CIENCIAS DE LA COMPUTACIÓN PARA EL AULA

Anexos para docentes

2º CICLO SECUNDARIA

ANEXO I

ANIMACIONES

En este anexo se trabaja con Alice, una herramienta gráfica que permite construir animaciones a partir personajes que se colocan en un escenario y a los que se les dan indicaciones.

Las primeras actividades funcionan como una introducción al entorno y la metodología de trabajo y, gradualmente, se introducen conceptos clave de programación motivados por consignas para que los estudiantes, en grupos, puedan incorporarlos a sus proyectos.

Anexo 1. Animación

En este primer capítulo trabajaremos con Alice, una herramienta gráfica que permite construir fácilmente animaciones a partir personajes que se colocan en un escenario y a los que se les dan indicaciones.

Las primeras actividades funcionan como una introducción al entorno y la metodología de trabajo y, gradualmente, se introducen conceptos clave de programación motivados por consignas para que los estudiantes, en grupos, puedan incorporarlos a sus proyectos. Además, se espera que se familiaricen con esos conceptos a través de una primera aproximación lúdica para poder utilizarlos en los capítulos siguientes.

Secuencia didáctica 1: Animación

- Alice y el Mundo
- Nuestros métodos
- Secuencialidad, simultaneidad y repetición
- Variables e interacción con el usuario
- Condicionales

Secuencia didáctica 1: Animación

Objetivos

- Familiarizarse con el entorno de programación Alice y apropiarse de la herramienta a partir de una producción de su propio interés.
- Conceptualizar un programa como una serie de instrucciones, en este caso, para los personajes, objetos, etc., que participarán de la animación.
- Conocer y utilizar adecuadamente la definición de métodos, la secuenciación, la simultaneidad, la repetición simple y la alternativa condicional, además de la definición de variables.

Actividad 1: Alice y el Mundo

Objetivos

- Aproximarse a la noción de programa, entendiéndolo como como una serie de guiones con instrucciones para los elementos de la escena.
- Familiarizarse con el funcionamiento de algunos aspectos básicos de la herramienta Alice, como el armado de la escena, los métodos y la elaboración de programas sencillos.

Modalidad de trabajo

Grupal (3 o 4)

Materiales

Computadoras
Alice

Desarrollo

A lo largo de esta actividad, los estudiantes ejecutarán por primera vez el lenguaje de programación Alice y empezarán a familiarizarse con su interfaz. En esta primera actividad presentaremos el proyecto a los estudiantes y daremos tiempo a los grupos para que exploren la herramienta. Esta exploración es el objetivo central de la actividad, razón por la cual las consignas son breves y muy generales para que, por un lado, los grupos puedan dedicar tiempo a familiarizarse con el entorno realizando diferentes pruebas y, por otro, para que estén motivados por sus intereses más que por una directiva del docente.

Comenzamos la clase contándoles a los grupos que realizarán una **animación**. Les preguntamos cómo creen que se hace una película, y rescatamos los distintos elementos que ellos consideren que intervienen en el proceso: actores, director, luces, cámaras, escenografía, etc. Destacamos especialmente que (en la mayoría de los casos) antes de comenzar la filmación se escribe un **guion**, es decir, un texto que dice detalladamente, no solo lo que deben decir los personajes, sino también cómo deben moverse e interactuar entre ellos, además de indicaciones relevantes para la cámara, las luces, el decorado y los objetos de utilería. Durante la filmación, quien dirige la película se encarga de que todas estas indicaciones sean reproducidas y llevadas a cabo.

Anunciamos que vamos a hacer algo muy similar en la computadora, dado que armaremos una escena con personajes a los que les daremos instrucciones para que las ejecuten, y también podremos dar indicaciones de luz o de cámara.

La herramienta que utilizaremos es Alice, un programa que pueden descargar desde https://www.alice.org/.

¿Qué es Alice?

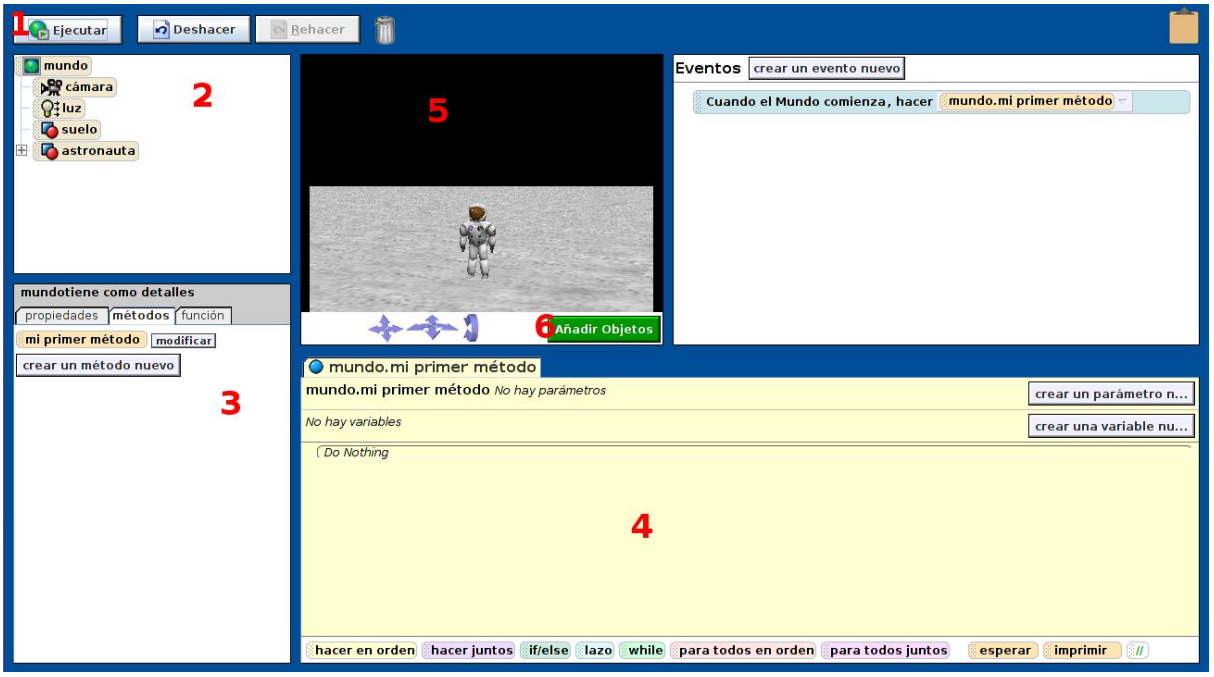
Alice es un lenguaje de programación en bloques orientado a la enseñanza, principalmente mediante la creación de animaciones y juegos. Es libre y gratuito.

Tutoriales de Alice

En el sitio web Dale Aceptar (www.daleaceptar.gob.ar), la Fundación Sadosky realizó más de 20 tutoriales (http://www.daleaceptar.gob.ar/contest/classes/3/) para estudiantes de secundario. El objetivo era que aprendieran de forma sencilla y amigable algunos conceptos básicos de Computación. Recomendamos chequear el material audiovisual para profundizar y complementar las actividades de este capítulo.

http://www.daleaceptar.gob.ar/

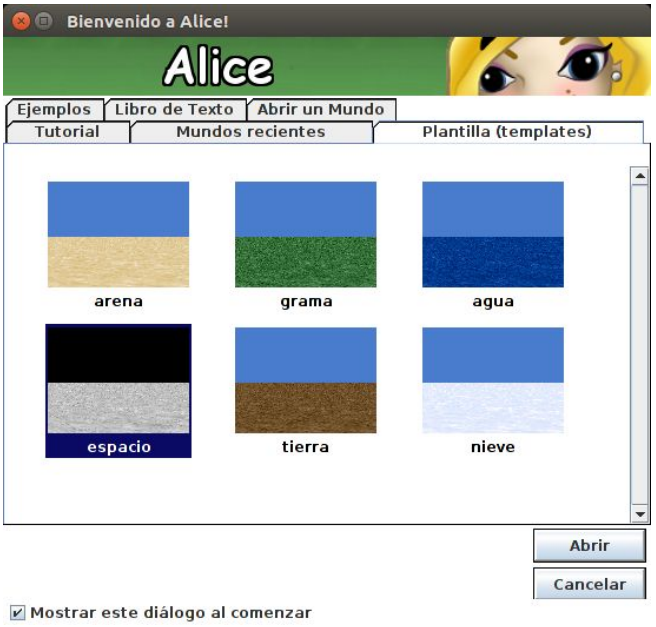
Invitamos a que los grupos abran Alice para conocer el entorno.
El entorno está organizado en distintas partes.



1. Ejecutar: el botón hace que se lleven a cabo las instrucciones del programa.
2. Árbol de objetos: aquí aparecen listados los objetos que forman parte de la escena (los personajes, las cosas, la cámara, etc.). Observamos que todos aparecen debajo de Mundo, por lo que utilizaremos esta palabra para referirnos a la escena con todos sus componentes.
3. Métodos: al seleccionar un objeto en el árbol, aquí aparecen sus métodos disponibles, es decir, las acciones que puede realizar. Los métodos consisten en bloques que podemos arrastrar y encajar para construir el programa en el sector respectivo.
4. Sector del programa: aquí colocamos los bloques para construir el programa.

5. Mundo: se trata de la escena en la que se ubican los personajes y objetos; es donde ocurre la acción.
6. Agregar objetos: este botón abre la biblioteca de objetos disponibles para que podamos elegir los que queramos agregar al Mundo.

Lo primero que debemos observar es que, para comenzar, se solicita elegir el escenario donde se realizará la animación, en la pestaña que se llama *Plantillas* (o *Templates*).

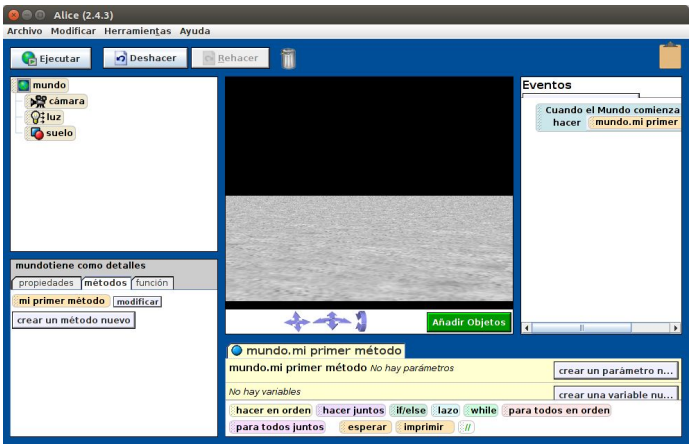


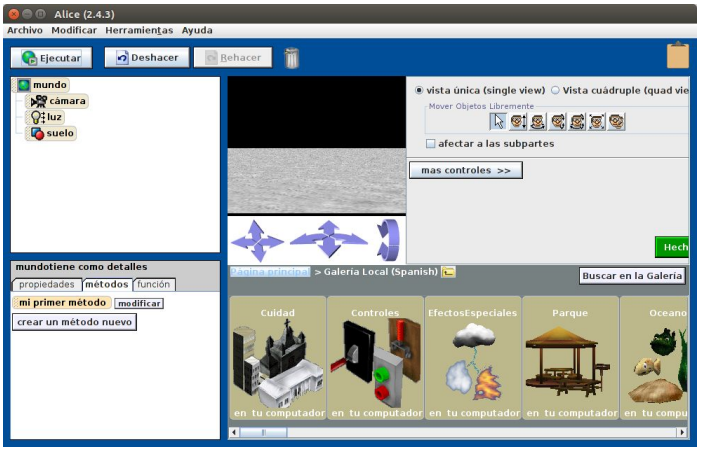
Es posible que el idioma esté configurado en inglés. En ese caso, pueden cambiarlo a español accediendo a *Edit → Preferences*, y en la pestaña *General,* elegir *Spanish* en la opción *Display language*, en la barra de menús.

Iniciamos la clase pidiéndole a los estudiantes que armen una escena en la que aparezcan dos personajes (recordar que en Alice los personajes son objetos) y un tercer objeto alejado de ellos. El objetivo es que ambos personajes se acerquen al lugar donde se encuentra el tercer objeto y luego tengan una conversación entre ellos.

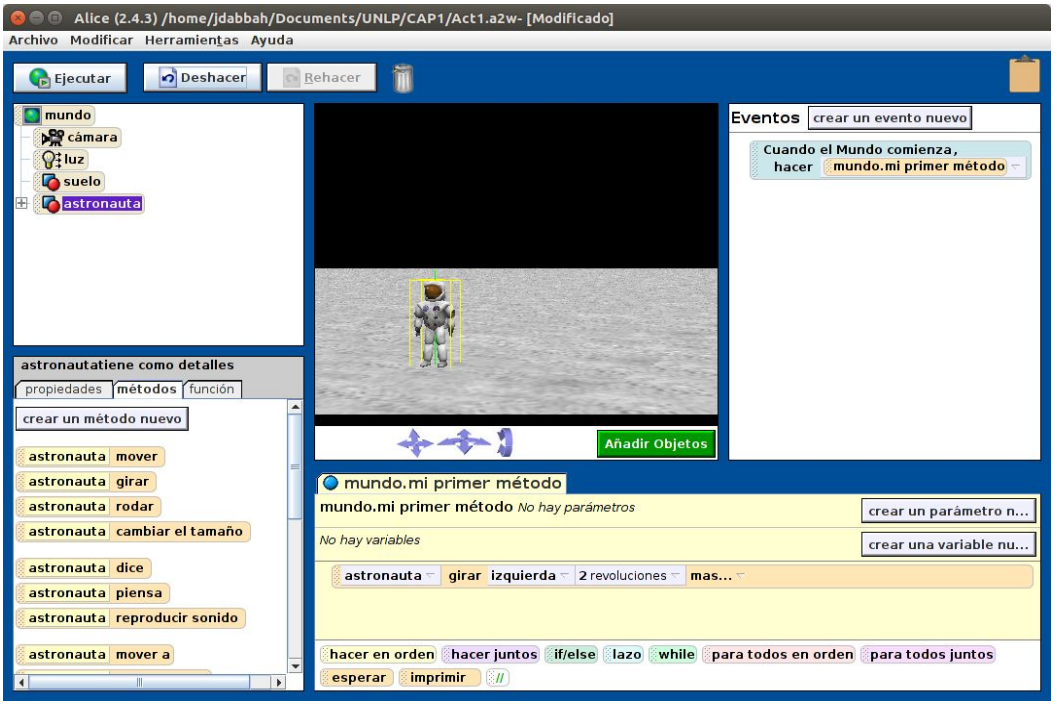
En el proyecto tenemos un ejemplo de una escena en el que están el astronauta y el flamenco en la luna y en el fondo se observa una carpa de franjas rojas y blancas.

Los estudiantes tienen que explorar cómo se agregan objetos al mundo. Para esto hay que rpesionar el botón *Añadir objeto.* Ahí se abre la *Galería* donde se podrá elegir entre los distintos objetos disponibles, agrupados en clases. Una vez que están agregados, los objetos pueden moverse arrastrándolos en la escena y pueden modificarse con los botones del cuadro superior derecho. Para regresar a la ventana anterior, se debe presionar el botón *Hecho*.





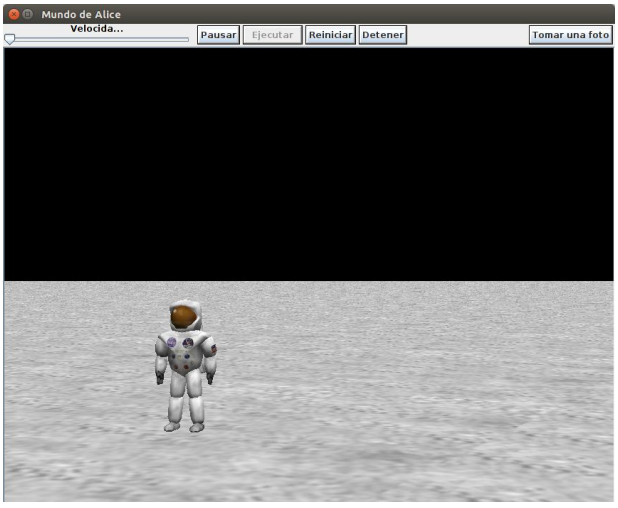
Luego, deben descubrir que cada objeto tiene métodos. Es decir, bloques que pueden arrastrarse al sector de programa, donde se va armando el guion de la animación, para darle indicaciones a un personaje.



Una vez que se agrega un objeto al mundo, aparece enumerado en el árbol del cuadro superior izquierdo, dependiente de `mundo`. Si lo seleccionamos, se despliegan sus métodos disponibles, que podemos arrastrar al sector del programa, donde aparecen como un bloque.

Luego deben ver qué hacen algunos de los métodos más importantes, como los que permiten que los personajes se muevan, digan frases, etc.

- Que al presionar el botón *Ejecutar*, se llevan a cabo las instrucciones y se produce la animación.



Si se presionamos el botón *Ejecutar*, veremos que se abre una nueva ventana y los objetos llevan a cabo los métodos que arrastramos al programa, produciendo de modo que se produce una animación. En esta ventana, además, contamos con botones para detener o reiniciar la animación, pausarla, cambiarle la velocidad o guardar una fotografía de lo que está sucediendo.

Como siempre, damos tiempo para que los estudiantes exploren el entorno y estamos atentos a las preguntas que puedan surgir en los grupos para orientarlos si lo necesitan. Observamos su avance en el dominio de la herramienta. En este sentido, debemos intentar lograr un balance entre el tiempo dedicado a construir el mundo agregando personajes y objetos y el momento en que comiencen a explorar los métodos y armar las primeras

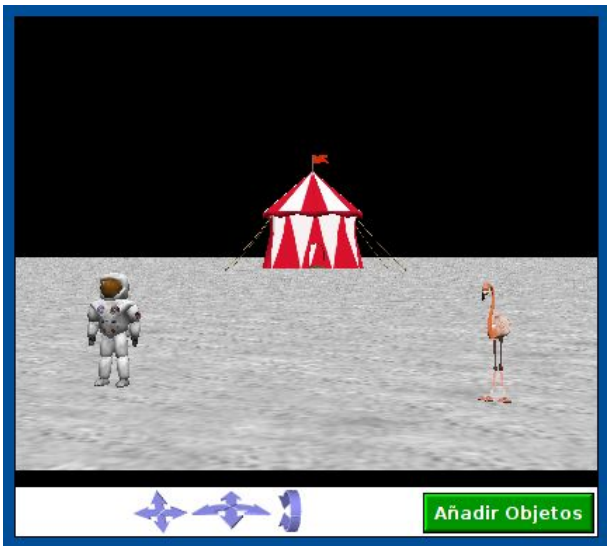
animaciones. Es fundamental dedicar tiempo a lo primero para fortalecer la motivación en el uso y la exploración de la herramienta. Sin embargo, si observamos que algún grupo no avanza a la siguiente etapa, los invitamos explícitamente a experimentar con los distintos métodos, recordándoles que continuaremos trabajando con el proyecto en las siguientes clases y que podrán completarlo luego.

Para tener en cuenta al trabajar con Alice:

- Para llevar un objeto a otro existen varias opciones:
    - Utilizar primero el método `girar para encarar a` (o `girar apuntar a`) y luego el método `mover`, eligiendo como dirección `adelante`. Esta es la manera recomendada, dado que no mueve a los personajes en sentido vertical, sino que solo los desplaza sobre el suelo.
    - `mover a` y `acercarse a` llevan un objeto hacia el centro del otro, que suele estar en su interior y, por lo tanto, a diferentes alturas. Luego, producen que los personajes se entierren en el suelo o se eleven de él. La diferencia entre ambos es que el primero produce el recorrido completo hasta el otro objeto, mientras que en el segundo se puede indicar una distancia menor.
- Si queremos utilizar un valor numérico que no aparece en la lista de opciones posibles, podemos presionar *otros…* Se abrirá un teclado numérico en el que podremos escribir el número que deseemos.
- Al elegir un objeto en los menús de los métodos, veremos que se despliega un menú en el que aparecen mencionadas algunas partes de estos objetos y, en primer lugar, `the entire ____`. Veremos en la actividad siguiente que los objetos están compuestos por partes que se pueden animar individualmente, pero si queremos tratar al objeto como un todo, debemos seleccionar `the entire ___`.
- Podemos realizar una copia de los bloques que ya tenemos en el programa haciendo clic sobre ellos con el botón derecho y eligiendo *hacer una copia.* De la misma manera, podemos borrarlos, habilitarlos o deshabilitarlos.

Cuando todos los grupos hayan armado su escena y agregado algunos métodos a sus personajes, hacemos una puesta en común para poner en claro los siguientes conceptos, que son centrales en la herramienta y a los que nos referiremos en las actividades siguientes.

- El mundo. En Alice, todos los personajes y objetos (incluyendo la escenografía, la cámara y la luz), forman parte del ***mundo***.

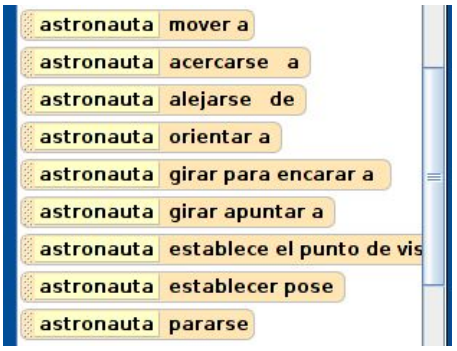
- 

- 

Los objetos añadidos al mundo del ejemplo son el astronauta, la flamenco y la carpa.

- Los objetos. En Alice, llamamos ***objeto*** a todo aquello que puede recibir instrucciones. El ejemplo más evidente son los personajes (que pueden ser personas, animales o, incluso, cosas), como así también la cámara y la luz.
- Los métodos. En Alice, las instrucciones que les podemos dar a los objetos para que las realicen durante la ejecución se llaman ***métodos***. Por ahora, observamos que todos los objetos tienen un conjunto limitado y muy parecido de métodos, que involucran movimiento. Más adelante construiremos nuestros propios métodos para complejizar el comportamiento de los personajes.

- 

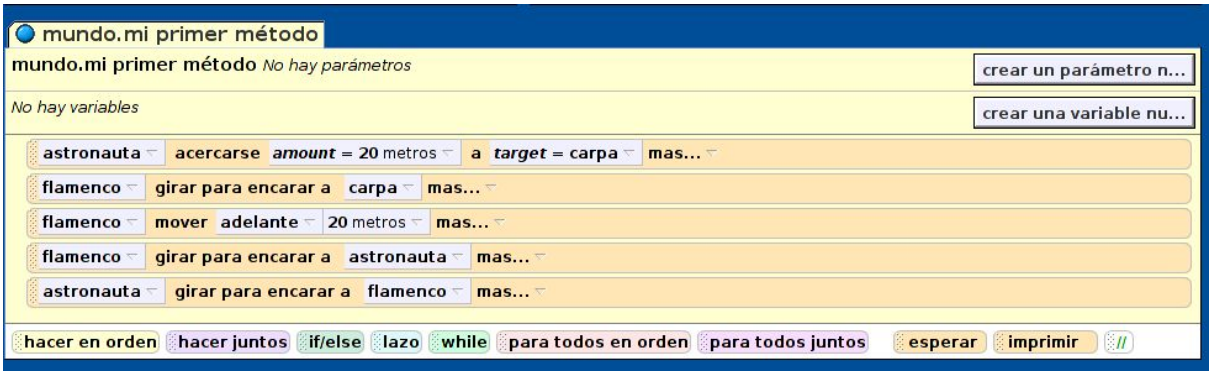
- 

- 

- 

Todos los objetos, incluida la cámara, comparten una serie de métodos que podemos utilizar en nuestras animaciones.

- El programa. Por ahora, lo conceptualizaremos como el guion de nuestra animación, es decir, como una serie de indicaciones para los distintos personajes u objetos que se pueden elegir de entre algunas posibles y que nosotros elegimos cómo combinar en base a lo que nos propongamos hacer. En esta línea, la ejecución de un programa es análoga a la situación de rodaje: todos los objetos y personajes realizan (ejecutan) estas indicaciones y vemos el resultado en la ventana de la animación.



El programa está constituido por una serie de métodos que se ejecutan en orden. Éste hace que los personajes del ejemplo avancen hacia la carpa y queden frente a frente.

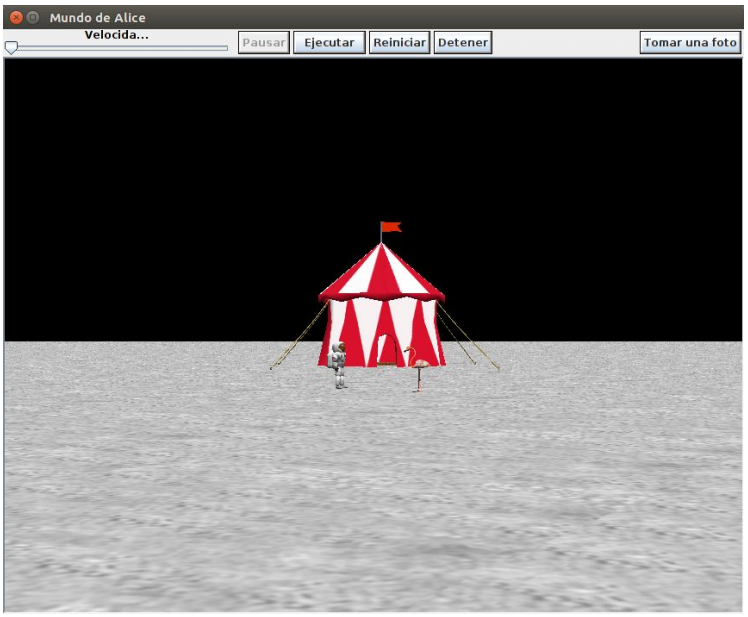
Una solución posible a esta primera consigna.
Observamos que el método `dice` agrega, además del texto, una duración (*duration)*. Para acceder a estas propiedades opcionales, debe hacerse clic sobre *más…*
También observamos un bloque `esperar`. Esto no es un método de ningún objeto, sino que se encuentra en la parte inferior de la pantalla, cerca del borde derecho, junto con otros bloques como *imprimir* o *//*.

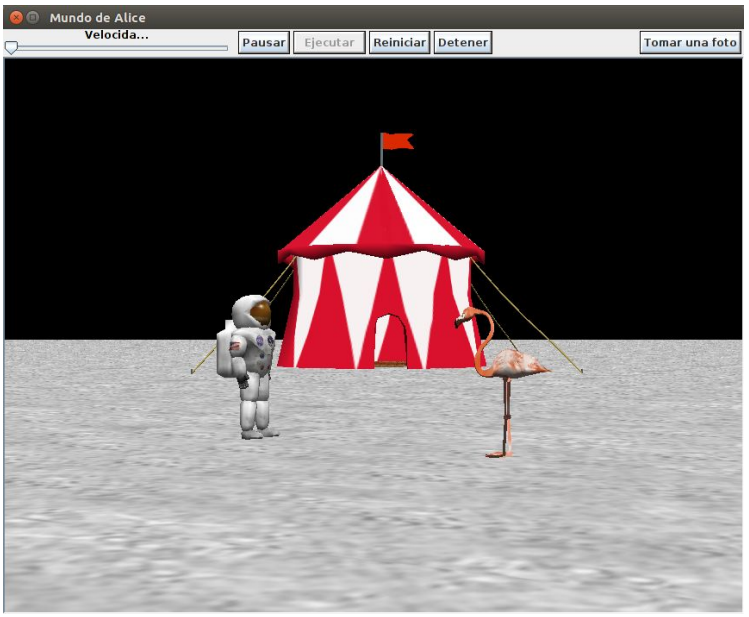
A continuación, les pedimos a los estudiantes que hagan que la cámara realice un recorrido hacia los personajes antes de que estos comiencen a hablar.

El objetivo de esta segunda consigna es poner en evidencia que la cámara, al ser un objeto del mundo, se controla con métodos, al igual que los personajes. Esperamos que los estudiantes exploren estos métodos para poder utilizarlos a medida que su proyecto se vaya desarrollando.

Para tener en cuenta al trabajar con Alice:

- Todas las instrucciones suelen tener un menú *más* donde se ofrecen distintas opciones. Si bien estas opciones dependen de cada instrucción en cuestión, suelen permitir establecer la duración, la velocidad y el estilo.
  Podemos invitar a los grupos a que lo exploren.





Vista antes y después de que se ejecute el método cámara acercarse 20 metros a carpa. Como ya dijimos, la cámara puede moverse con los mismos métodos que los objetos, para lograr que la escena se vea de distintas maneras.

Cierre

Cerramos esta actividad observando la relación que existe entre los personajes de un guion y los objetos del mundo de Alice, como así también entre las indicaciones que aparecen en aquel y los métodos de este. En este mismo sentido, dejamos en claro que la secuencia de métodos que construyeron los grupos, y que hasta ahora habíamos pensado como un guion constituye, en realidad, un **programa**, es decir, una serie de **instrucciones destinadas a una computadora para que las lleve a cabo** (en este caso, mover los personajes, colocar los globos de diálogo, etc.). Utilizaremos esta palabra de ahora en adelante.

Terminamos mostrando el bloque de comentario, que nos permite introducir texto entre las instrucciones de nuestro programa y pedimos a los grupos que lo utilicen para escribir qué va sucediendo sucede en la animación, es decir, los fragmentos del guión guion a medida que van avanzando.


// El astronauta y la flamenco avanzan hacia la carpa
astronauta acercarse amount = 20 metros a target = carpa mas...
flamenco girar para encarar a carpa mas...
flamenco mover adelante 20 metros mas...
// Los dos giran para quedar frente a frente
flamenco girar para encarar a astronauta mas...
astronauta girar para encarar a flamenco mas...
// Se acerca la cámara
cámara acercarse amount = 20 metros a target = carpa mas...
// Conversan
astronauta dice Bienvenida a la Luna. duration = 2 segundos mas...
esperar 0.5 segundos
flamenco dice Muchas gracias. Es un placer estar acá. duration = 2 segundos mas...

Actividad 2: Nuestros métodos

Objetivos

- Observar que pueden crearse nuevos métodos en Alice y utilizarlos para definir acciones más complejas.
- Reconocer las ventajas de la definición de métodos para la construcción y la comprensión de los programas.

Modalidad de trabajo

Grupal (3 ó 4)

Materiales

Computadoras
Alice

Desarrollo

El principal objetivo de esta actividad es presentar la noción de métodos propios como un recurso para enriquecer las posibilidades de los objetos, es decir, tener en cuenta que podemos agregarles nuevos comportamientos definiendo un método apropiado a partir de los que ya existen, de una manera particular.

Les pedimos a los estudiantes que logren que uno de los personajes haga siempre el mismo gesto antes de empezar a hablar (por ejemplo, levantar los dos brazos, saludar, girar la cabeza a ambos lados).

Durante el desarrollo de la consigna, esperamos que los estudiantes observen que:

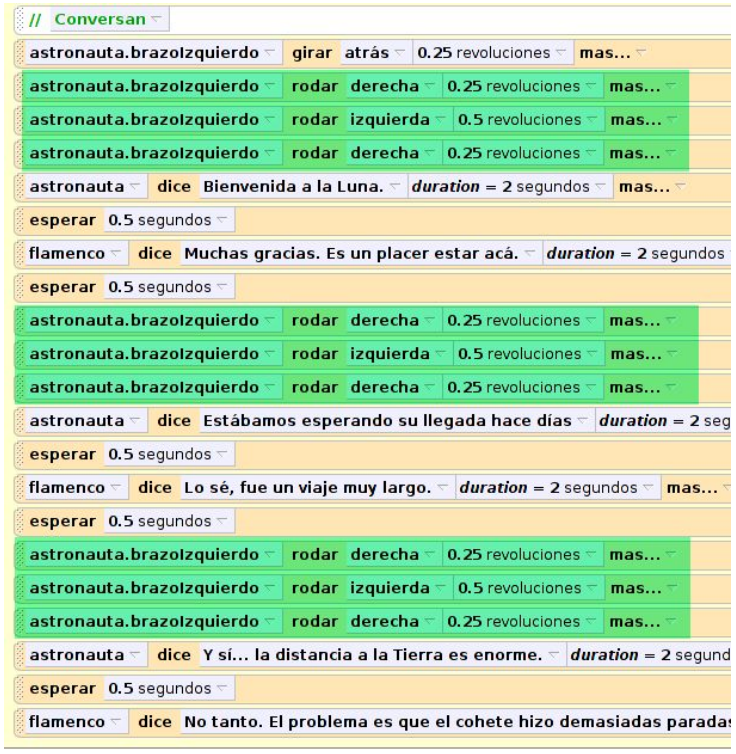
- Los objetos tienen subpartes a las que se accede en el árbol de objetos a través del símbolo **+** a la izquierda del nombre y que estas son también objetos, por lo que cuentan con métodos asociados para ser animadas (y, además, pueden contener subpartes).
- Deberán animar las subpartes de los objetos, para que estos puedan realizar movimientos en el lugar con distintas partes de su cuerpo.
- Deberán usar las mismas secuencias de instrucciones en distintos lugares de la animación (las que controlan el gesto en cuestión).

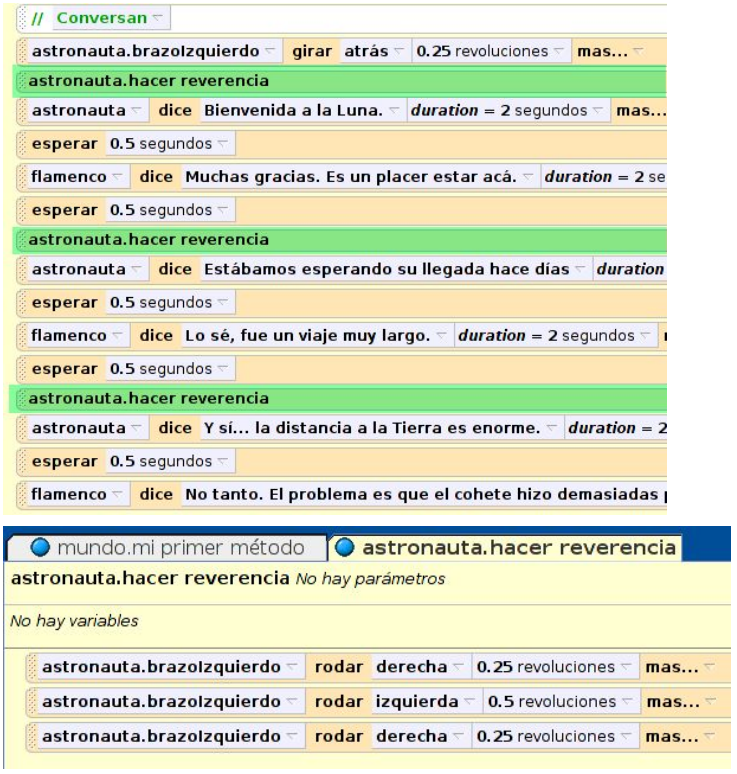
Después de la exploración pertinente, cuando todos los grupos hayan observado estos aspectos, planteamos que la **definición de métodos propios** es una solución adecuada para estos problemas, pues estos métodos:

- Condensan en un bloque la especificidad del manejo de las subpartes ya que permiten representar muchas instrucciones específicas como una única instrucción del programa que controla la animación. Debemos destacar que la longitud del

programa no es una virtud en sí misma, pero sí lo es su claridad, a la que definitivamente contribuye el hecho de que se cuente con una única instrucción en vez de una secuencia de instrucciones que solo adoptan sentido cuando se ejecutan todas juntas. En este sentido, es importantísimo el nombre que elegimos para el método. Si bien esto no afecta al funcionamiento del programa, es fundamental para que el programa pueda ser interpretado por las personas que trabajen con él, es decir, todas las personas que quieran entenderlo, arreglarlo o mejorarlo, hoy o en el futuro.

- Permiten escribir una única vez una serie de instrucciones y después usarla tantas veces como queramos, simplemente colocando un bloque en el programa. Lo primero se denomina ***definición*** del método, y a sus apariciones en el programa se las designa como ***invocación***, ***llamada*** o ***uso***.





Observamos cómo podemos utilizar un método para agrupar un grupo de instrucciones que, entre todas, tienen un efecto en particular. Por ejemplo, en este caso, el astronauta siempre hace una reverencia antes de hablar. Luego, definimos el método `hacer reverencia`, que consiste en las tres instrucciones para mover el brazo, y después lo usamos como una instrucción más en nuestra animación. Además de ahorrarnos trabajo repitiendo instrucciones, este método permite que el programa de la animación sea más fácil de comprender y, por lo tanto, de comunicar a otras personas, para compartirlo, arreglarlo o mejorarlo.

Mostramos cómo se crea un método y, además, observamos que este se agrega a los métodos básicos de los personajes. Damos tiempo para que todos los grupos puedan definir el método que describe el gesto que hace el personaje antes de hablar.

Vemos el botón *Crear un método nuevo*, que nos permite agregarle un método definido por nosotros al personaje seleccionado. Además, observamos que, después de creado, este método aparece disponible junto a los métodos del personaje que ya habíamos visto antes.
Si presionamos el botón *modificar*, se abre en el sector del programa una pestaña donde podemos arrastrar instrucciones, como veníamos haciendo hasta ahora, para construir la definición del método.

Para tener en cuenta al trabajar con Alice:

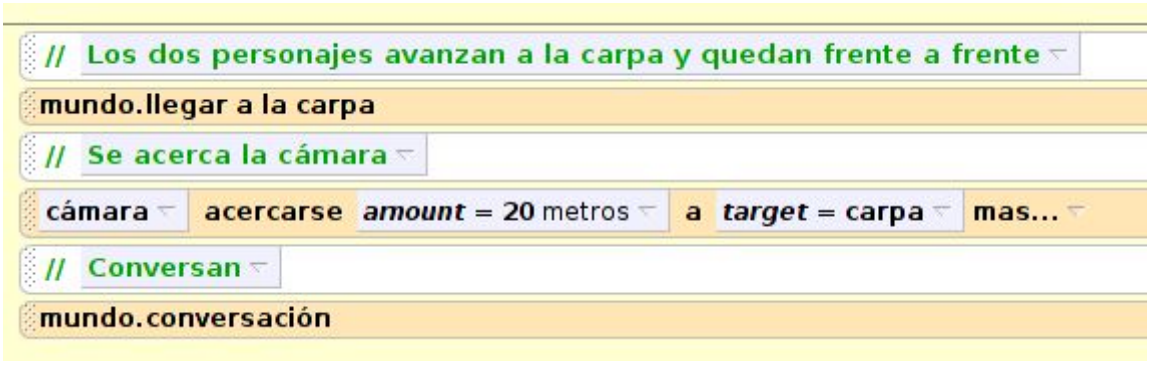
- Portapapeles de instrucciones: En la esquina superior derecha hay un portapapeles que sirve para **copiar y pegar instrucciones**. Es decir, si arrastramos una instrucción sobre él, luego podemos extraer copias para colocar en cualquier lugar del programa, simplemente haciendo clic sobre el ícono y arrastrando. Esto es particularmente útil para mover instrucciones entre distintas pestañas (o sea, distintas definiciones de métodos).
  Para poder aprovecharlo mejor, es necesario hacer un truco de modo de salvar la limitación de que solo puede contener una instrucción o bloque por vez y usarlo para copiar más de una: podemos encastrar todas las instrucciones que queramos mover dentro de alguno de los bloques que aparecen en el pie de la pantalla (como por ejemplo, `hacer en orden`, aunque puede ser cualquiera pues el bloque no formará parte del programa final). Una vez armado el bloque, si lo arrastramos al portapapeles, se copiará con todo su contenido y luego podremos arrastrarlo a otras pestañas, donde sacaremos las instrucciones para colocarlas donde necesitemos y eliminaremos el bloque vacío.
- Para **acceder a la definición de los métodos** propios: debemos presionar el botón *modificar*, que aparece junto al nombre del método en los métodos del personaje. Esto abre una pestaña para completar, modificar o ver la definición del método.

Como conclusión, podemos observar que, después de definir el método, el personaje *sabe hacer* algo nuevo y, para mejor, es algo que a nosotros nos resulta útil para nuestra animación. De hecho, a partir de aquí podemos proponer como estrategia de trabajo pensar qué instrucciones necesitaríamos que tuvieran nuestros personajes y definir aquellas que no aparezcan como métodos ya definidos. Con estas nuevas instrucciones definidas, entonces,

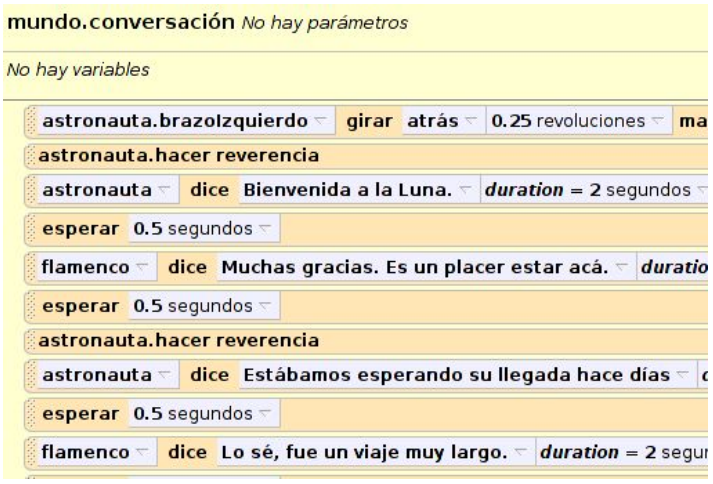
debería ser mucho más fácil construir la animación y, seguramente, resulte mucho más claro qué está haciendo el programa.

Con esta misma idea, les proponemos a los grupos que piensen si no hay algunas partes de sus programas que podrían conformar distintos métodos. Para esto, les podemos sugerir que identifiquen secuencias de instrucciones que, todas juntas, producen una situación o momento particular de la historia. Si necesitan ayuda para identificar estas partes, podemos pedirles que nos cuenten su animación con sus propias palabras (o que observen los comentarios que fueron agregando) y reconozcan en el programa qué instrucciones se corresponden con cada uno de los momentos que mencionan en su narración. También debemos preguntarles qué nombres les pondrían a esas instrucciones y motivarlos a que elijan nombres lo más representativos posible.
Es importante observar que, en Alice, todos los métodos deben estar definidos como asociados a algún personaje. Cuando involucran acciones de un único personaje para agregarle capacidades (como la reverencia del astronauta), tiene sentido asociarlos a este personaje. Sin embargo, cuando involucran más de uno o cuando no sea una acción puntual del personaje sino una situación más general, conviene definirlos en el mundo para poder encontrarlos fácilmente.

Por ejemplo, en nuestra animación, podemos agrupar en un método las instrucciones que llevan los personajes hacia la carpa y, en otro, la conversación.



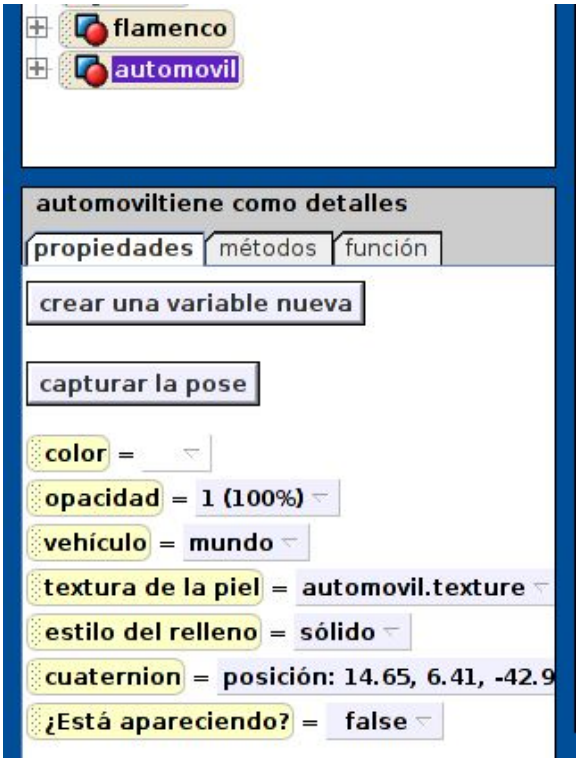
<epi> Programa principal de la animación, donde cada situación fue reemplazada por un método



Definiciones de los métodos invocados en el programa principal. Son las mismas instrucciones que antes, pero agrupadas en métodos con un nombre que deja en claro qué hacen en conjunto. Observamos que están definidos como métodos del mundo.

A continuación, les pedimos a los estudiantes que logren lo siguiente: durante la conversación, aparece un objeto que produce una interrupción y llama la atención de los personajes.

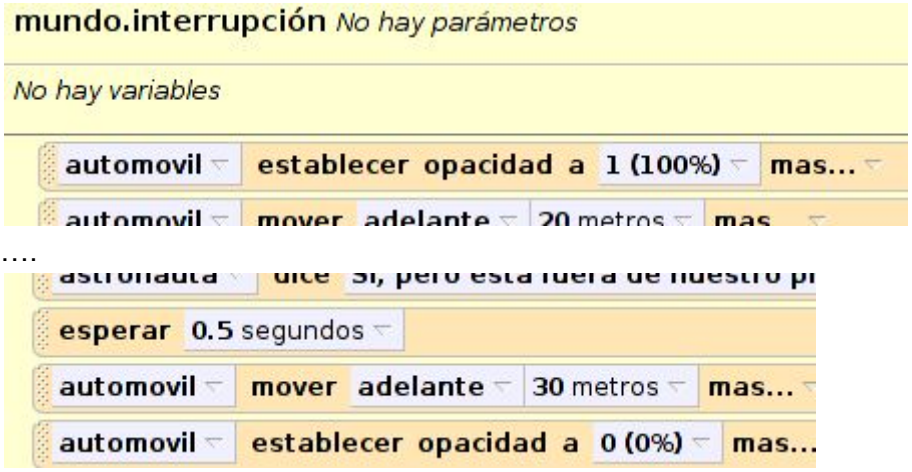
Para resolver esta consigna, los estudiantes deberán descubrir alguna manera de hacer aparecer objetos en la escena durante la animación. Para esto, probablemente utilicen alguna de las siguientes alternativas: colocar el objeto *fuera de cuadro*, es decir, por fuera del ángulo de visión de la cámara y moverlo hacia adentro para hacerlo aparecer; si quieren que aparezca espontáneamente en algún lugar en vez de ingresar, deberán operar con la propiedad `opacidad` o con la propiedad `¿Está apareciendo?` Estas propiedades se encuentran, justamente, en la pestaña *propiedades* del objeto seleccionado, al lado de la pestaña de métodos.



Al seleccionar un objeto, junto a la pestaña de *métodos*, vemos la pestaña de *propiedades*, donde se pueden controlar algunas características del objeto. En particular para esta consigna nos interesan dos:

- `opacidad` (`opacity`): expresa, mediante un porcentaje, qué tanto se puede ver a través del objeto: 0% significa que el objeto es perfectamente transparente (o invisible), mientras que 100% indica que el objeto es totalmente opaco, es decir, no deja ver lo que tiene atrás.
- `¿Está apareciendo?` (`Is showing?`): indica si el objeto es visible o no (para esto, utiliza los valores `true` (verdadero) y `false` (falso). Puede pensarse como una segunda instancia de opacidad: si un objeto tiene establecida la opacidad al 50% y `¿Está apareciendo?` en `true`, será visible con una apariencia semitransparente, mientras que si está en `false` directamente será invisible.

Lo interesante es que, si arrastramos estos bloques al espacio del programa, obtenemos la instrucción `establecer [propiedad] a...`, es decir, una instrucción que permite alterar la propiedad en cuestión durante el programa. Luego, para hacer aparecer un objeto, debemos configurar su opacidad (o `¿Está apareciendo?`) como invisible al comienzo de la animación y después utilizar la instrucción `establecer opacidad a 100%`, por ejemplo.



….

Observamos que, en la primera instrucción, hacemos aparecer el automóvil modificando su opacidad (lo que le da un efecto de fundido, pues es gradual) y lo hacemos desaparecer sacándolo de la escena, al hacerlo avanzar por fuera del límite de cuadro. De todas maneras, establecemos la opacidad en 0% para que no aparezca por error en futuros movimientos de cámara.

Cierre

Cerramos esta actividad con una breve discusión a propósito de la utilidad de los métodos (es muy probable que, al principio, los estudiantes no estén convencidos de esto). La primera pregunta que debemos hacer es si la definición de métodos es imprescindible para la confección de programas. La respuesta es que no, pues, en lugar de haber procedido como lo hicimos en la actividad, podríamos haber reemplazado todos los usos de un método por una copia de las instrucciones que conforman su definición y habríamos obtenido un programa equivalente.

Entonces, ¿cuál es la ventaja? Ya observamos que programar utilizando métodos es más cómodo, pues en general evita tener que escribir repetidas veces algunas instrucciones; sin embargo, como ya dijimos, su principal beneficio es la posibilidad de escribir programas más claros, es decir, programas más fáciles de entender. Además, permite organizar mejor la tarea de construcción del programa, pues nos obliga a resolver de a un problema por vez. Esta idea de ir construyendo una solución mayor a partir de las de pequeños problemas es fundamental en computación y constituye una estrategia perfectamente válida para resolver problemas en general (que, de hecho, usamos sin darnos cuenta todo el tiempo).

Actividad 3: Las cosas en orden
Secuencialidad, simultaneidad y repetición

Objetivos

- Comparar la ejecución secuencial y la simultánea, e incorporarlas a las animaciones.
- Conocer la estructura de repetición simple y aprovecharla en las animaciones.

Modalidad de trabajo

Grupal

Materiales

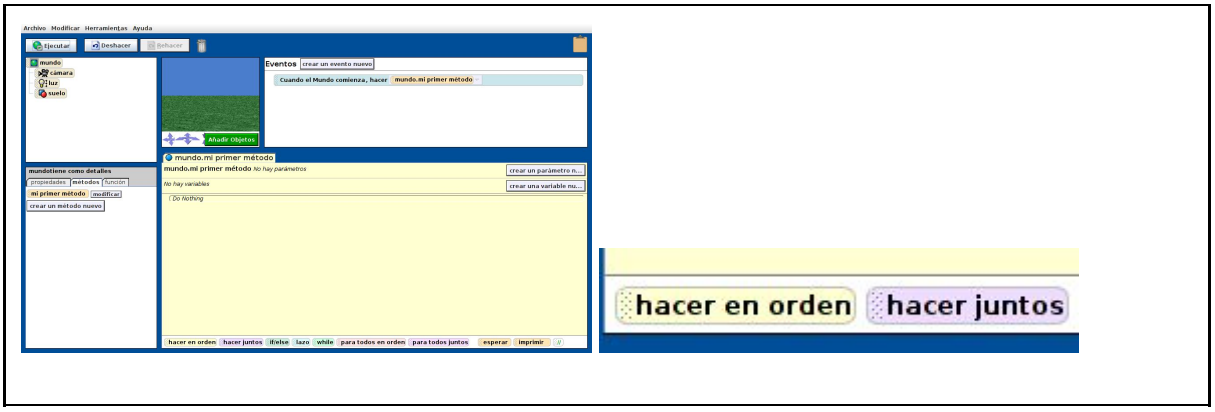
Computadoras
Alice

Desarrollo

Si bien la ejecución de un programa es por defecto secuencial (es decir, en orden), también existen bloques para indicar que una serie de instrucciones debe ejecutarse simultáneamente o ejecutarse más de una vez. El objetivo de esta actividad es poner de manifiesto la necesidad de modificar el orden secuencial y mostrar las herramientas que pueden usarse para conseguirlo.

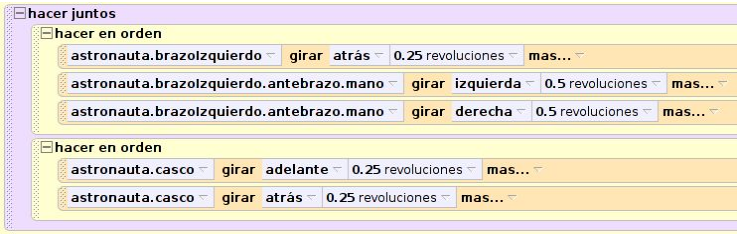
Les pedimos a los estudiantes que realicen la siguiente consigna: “Antes de la conversación, los personajes se saludan con un saludo especial que solo conocen ellos, como por ejemplo, chocar las manos en alguna posición en particular o hacer, al mismo tiempo, alguna secuencia de movimientos”.
https://www.youtube.com/watch?v=7VvKO65Lbp0

El objetivo de esta consigna es que los estudiantes reconozcan que, en general, todos los programas que construyeron se ejecutan de manera secuencial, es decir, las instrucciones se ejecutan en el orden en el que fueron escritas y de a una por vez. Esto impide representar acciones que se realicen simultáneamente, como, por ejemplo, que los dos personajes levanten el brazo y acerquen las manos al mismo tiempo para chocarlas. Para esto, esperamos que observen que, al pie del espacio del programa, entre los bloques disponibles hay uno llamado `hacer en orden` y otro denominado `hacer juntos`.



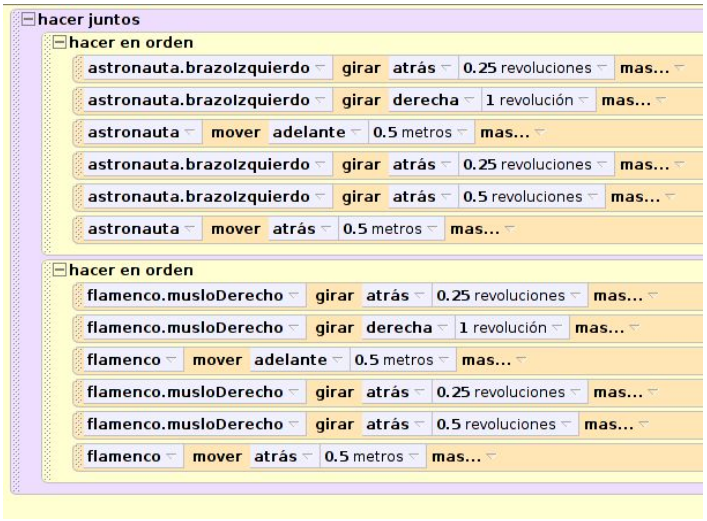
Los bloques `hacer en orden` y `hacer juntos`, debajo del sector del programa

Es probable que, al comenzar a experimentar con el nuevo bloque, los grupos obtengan resultados inesperados. Esto se debe a que todas las instrucciones dentro de `hacer juntos` se ejecutan simultáneamente y ninguna en orden. Luego, si hace falta más de una instrucción para realizar una acción en particular (por ejemplo, primero levantar el brazo y luego girar la mano para saludar) y queremos que se ejecuten en orden, pero al mismo tiempo que otra (por ejemplo, mover la cabeza), no podemos colocarlas todas juntas, pues sucederían las tres al mismo tiempo. Para solucionar este problema existe el bloque `hacer en orden`, que podemos combinar con `hacer juntos`, como se ve a continuación.

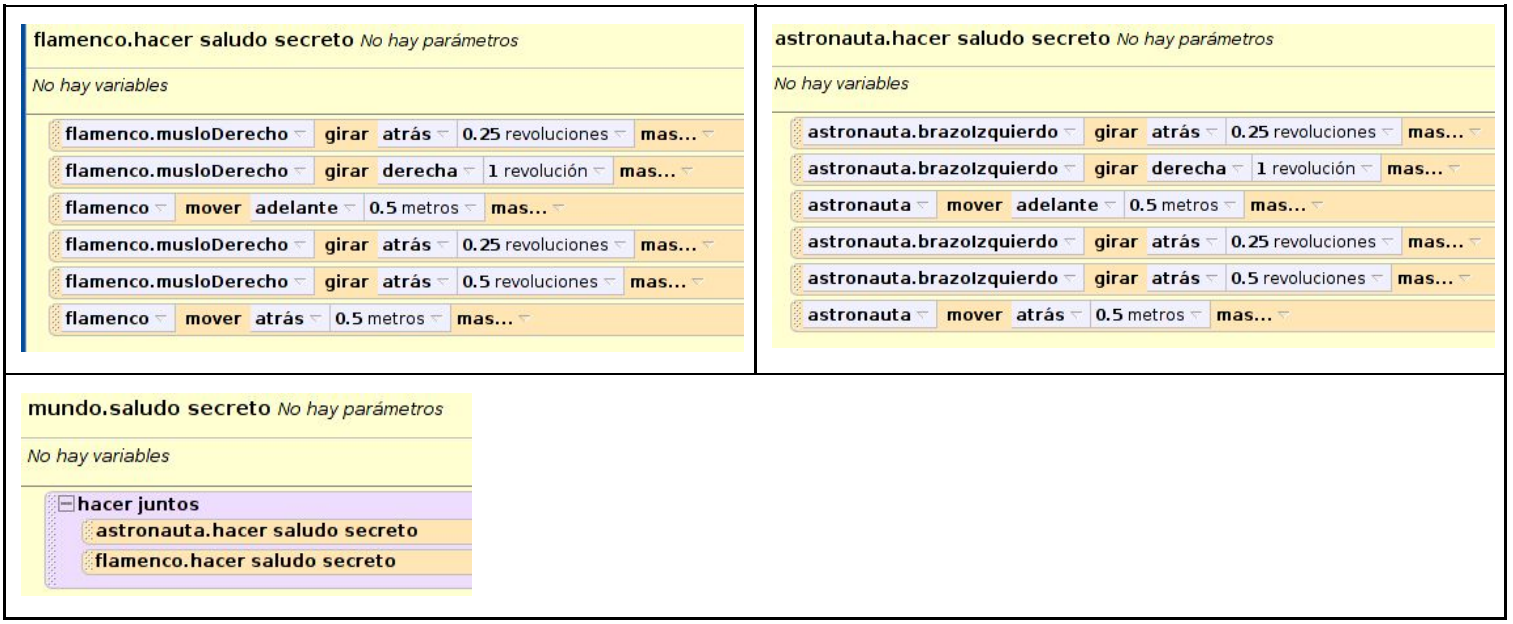


El primer programa hace que el astronauta gire el brazo y ningún efecto notable sobre el resto de las articulaciones, pues se están ejecutando simultáneamente instrucciones contrarias (como girar a la izquierda y a la derecha a la vez). En cambio, si queremos que el astronauta salude y mueva la cabeza al mismo tiempo, podemos construir un programa como el segundo: el primer bloque `hacer en orden` contiene la secuencia para que salude (levantar el brazo y girar la mano a ambos lados) y el segundo, la que hace que el astronauta baje y suba la cabeza. Al estar ambos contenidos en un mismo bloque `hacer juntos`, el resultado del programa es que estas dos secuencias se ejecutan simultáneamente.

Además del bloque `hacer en orden`, otra forma de resolver este problema (y que en general preferiremos, pues contribuye a la claridad del programa) consiste en definir métodos con cada una de las secuencias que deben ejecutarse en orden. Por ejemplo, nuestro saludo secreto podría ser el siguiente.



Sin embargo, el contenido de cada bloque `hacer en orden` se corresponde con una acción puntual de cada personaje. Luego, podríamos definir un método en cada uno que contenga esta secuencia, como se ve a continuación, y utilizarlos para definir el saludo de manera más clara. Insistimos con esta práctica para que los estudiantes identifiquen qué secuencias pueden convertirse en métodos y los definan.



La claridad del programa mejora al definir métodos para indicar secuencias de instrucciones

Cuando todos los grupos hayan programado su saludo secreto, interrumpimos el trabajo en la computadora y hacemos una puesta en común a propósito de lo que descubrieron y fuimos discutiendo individualmente:

- El bloque `hacer juntos` permite indicar que una serie de métodos se ejecutan al mismo tiempo, mientras que `hacer en orden` indica que deben ejecutarse uno después del otro, como estábamos acostumbrados a ver en nuestros programas.
- El bloque `hacer en orden` colocado dentro del bloque `hacer juntos` permite indicar que una secuencia de métodos debe ejecutarse en simultáneo con otros.

Esta es la posibilidad más interesante que ofrece el bloque, aunque probablemente también una de las más confusas. Podemos detenernos en ella y mostrar algunos ejemplos escribiendo algunas instrucciones en el pizarrón. Para esto, proveemos un ejemplo muy sencillo:

Los puntos verde y naranja deben llegar al mismo tiempo a la estrella atravesando el “laberinto”. Contamos con las instrucciones ←↑→↓, es decir, avanzar un paso en cada una de las cuatro direcciones.
Si escribimos el siguiente programa

```
Hacer Juntos [
  verde ↓
  verde →
  naranja ↓
  naranja ←
]
```

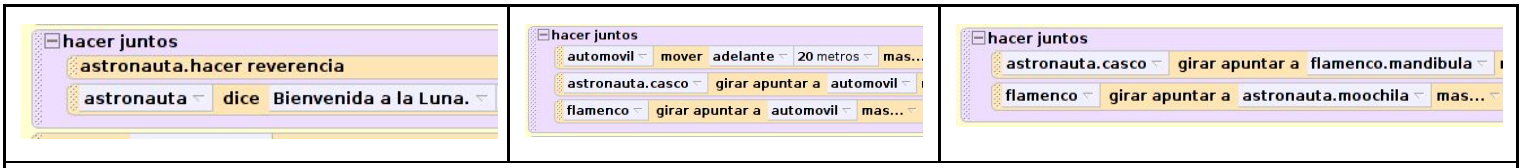
esto hará que los puntos se muevan en diagonal (pues se ejecutarán al mismo tiempo las instrucciones de desplazarse hacia abajo y hacia los costados), con lo que chocarían contra las paredes del laberinto. Lo que queremos es que, al mismo tiempo, cada uno primero baje y luego se desplace horizontalmente. Esto se puede escribir con el siguiente programa, utilizando el bloque `hacer en orden`:

```
Hacer Juntos [
  Hacer en orden [
    verde ↓
    verde →
  ]
  Hacer en orden[
    naranja ↓
    naranja ←
  ]
]
```

Podemos ejecutar ambos en el pizarrón, si hace falta, para que quede más claro.

- Como destacamos al final de la actividad anterior, se advierte la importancia de definir métodos con nombres adecuados para ordenar la elaboración y la comprensión del programa.

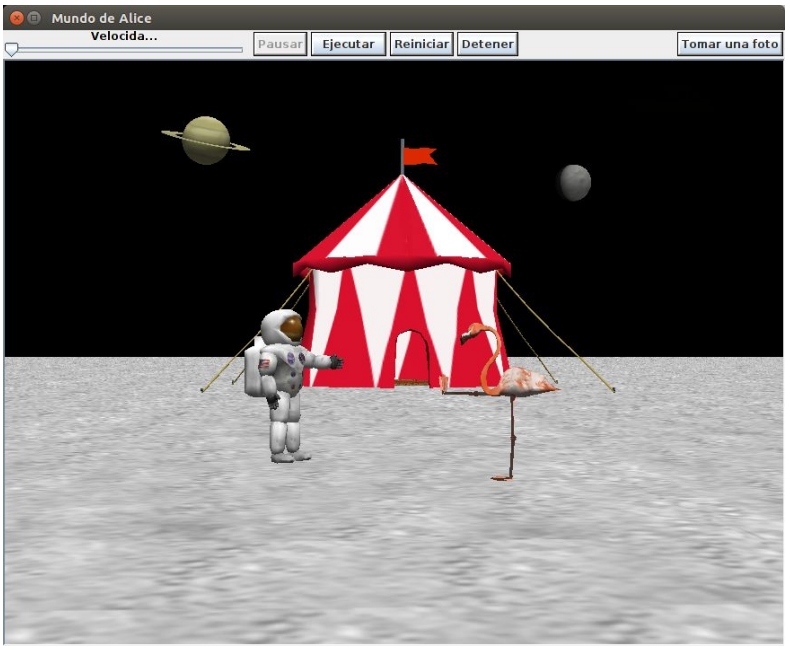
A medida que los grupos se sientan cómodos con este bloque y descubran cómo pueden aplicarlo para mejorar sus animaciones, probablemente quieran utilizarlo en otros lugares para que la acción se vea más fluida, para lo que les damos tiempo. Sin ir más lejos, en nuestro ejemplo, existen varias situaciones en las que podemos utilizar este bloque para mejorar la calidad de la animación.



Tres ejemplos donde podemos utilizar el bloque `hacer juntos` para mejorar la fluidez de la animación y darle un aspecto más realista. En el primer ejemplo, dentro del método `conversación`, ubicamos la reverencia y el parlamento en un mismo bloque, para que sucedan al mismo tiempo. En los otros dos, en el método `interrupción`, hacemos que los personajes se muevan los dos al mismo tiempo, en vez de tener que esperar a que termine uno para que el otro comience. En el primer caso, ambos giran hacia el automóvil al mismo tiempo, y en el segundo, retoman su posición original cuando este se va.

La segunda consigna de la actividad tiene como objetivo motivar el uso de bloques de repetición y forzar una situación en la que deba utilizarse el bloque `hacer juntos`. En este caso, los estudiantes tienen que animar el fondo de la escena. Un truco utilizado por los animadores para evitar dedicar demasiado tiempo a animaciones muy largas pero que no son muy importantes visualmente consiste en repetir un segmento cíclicamente.

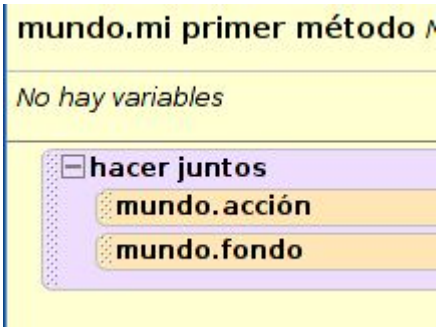
A partir de esta consigna esperamos que los grupos coloquen algunos objetos en el fondo de su escena y los animen con algunos comandos. Para esto deberán resolver primero el problema de que esta animación debe suceder en simultáneo con la animación de los personajes que ya tenían.



Comenzamos con un momento de exploración, para que todos los grupos descubran la dificultad mencionada y esbocen una solución, a partir de lo discutido en la consigna anterior. Hacemos una breve interrupción y, entre todos, comentamos las distintas propuestas. Recordamos que ya pudimos resolver fácilmente la necesidad de que dos cosas ocurran al mismo tiempo utilizando un bloque `hacer juntos`, pero que hacerlo directamente produce un programa muy poco claro y difícil de leer.

Hacemos referencia una vez más a las ventajas de escribir programas definiendo métodos propios: debemos concluir que una buena solución es armar un método donde se programe la animación de los personajes y otro para el fondo. De esta manera, quedarán separadas las definiciones y solo se unirán en el programa principal, que constará de un bloque `hacer juntos` y dos métodos en su interior. Al finalizar esta puesta en común, les indicamos a los grupos que continúen con el programa y lo organicen teniendo en cuenta lo que discutimos. En el ejemplo, vemos los métodos `mundo.acción` y `mundo.fondo` ubicados dentro de un bloque `hacer juntos`, en el cuerpo del programa principal.

Todavía no avanzamos a la definición de `mundo.fondo`, pues para esto debemos presentar el bloque de repetición. Dado que la traducción elegida (lazo o ciclo) es poco significativa, deberemos señalarlo especialmente e indicarles a los grupos que experimenten con su funcionamiento y lo incorporen a sus animaciones si lo necesitan. No nos interesa forzar su uso, pero podemos sugerirlo para generar fácilmente movimiento permanente en el fondo.



Programa de ejemplo, separado en los métodos `mundo.acción` y `mundo.fondo`, combinados en un bloque `hacer juntos`.

El método `mundo.acción` contiene el programa que controla los personajes, tal como estaba antes en `mundo.mi primer método`, mientras que `mundo.fondo` hace que se muevan los objetos del fondo. En particular, en su definición observamos:

- El bloque `hacer juntos` que contiene todos los otros bloques, pues queremos que todos los objetos del fondo se animen al mismo tiempo.
- El bloque `lazo o ciclo [ ] veces` puede indicar una repetición finita, como en el caso de la bandera, en el que repite tres veces su contenido en orden, o puede indicar una repetición infinita, como en el caso de la Luna o de Saturno. Si se decide utilizar este último bloque hay que prestar atención a que, si la ejecución es en orden secuencial, todas las instrucciones que aparezcan por debajo de él no se ejecutarán nunca. Por eso lo utilizamos dentro de un bloque `hacer juntos`.

- El método `girar a la velocidad` hace que la velocidad del movimiento sea constante (en vez de acelerarse al principio y detenerse al final), lo que permite encadenar sucesivos giros sin que se note una detención y un arranque entre ellos.

Damos tiempo a los grupos para que completen y mejoren sus animaciones, mientras recorremos los grupos y los orientamos con algunas de las ideas que usamos en nuestro ejemplo.

Cierre

Concluimos esta actividad observando que, si bien, en general, las instrucciones se dan para ser seguidas en orden, esto no tiene por qué ser necesariamente así. Más aún, existen casos en los que necesitamos que ciertas acciones se hagan en simultáneo o que, al terminar de ejecutarse, se repitan en vez de continuar con las siguientes.

Actividad 4: Variables e interacción con el usuario

Objetivos

- Incorporar el concepto de variable como una herramienta para conservar datos durante la ejecución de un programa.
- Presentar algunas herramientas básicas para interactuar con el usuario.

Modalidad de trabajo

Grupal (3 ó 4)

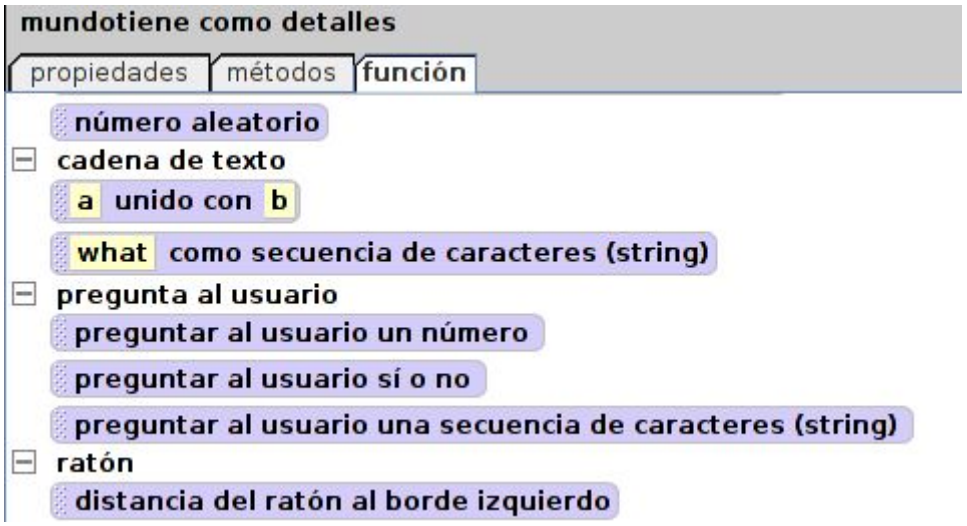
Materiales
Computadoras
Alice

Desarrollo

En esta actividad, los estudiantes comenzarán a agregarles interactividad a sus animaciones, a partir de hacerle alguna pregunta al usuario y utilizar la respuesta en los diálogos de los personajes. Además, les pediremos que utilicen la respuesta más de una vez, de manera que necesiten crear y asignar variables para almacenar datos durante la ejecución. Para reforzar esta conceptualización, interrumpiremos el trabajo en las computadoras para realizar una breve explicación.

Les planteamos a los estudiantes la primera consigna: “En algún momento, la animación debe preguntarle algo al usuario para que luego aparezca en los diálogos de los personajes. Por ejemplo, puede preguntarle el nombre para que, después, uno de los personajes lo salude por su nombre”.

Para poder completar esta consigna, los estudiantes deberán explorar la categoría *pregunta al usuario* en las funciones del mundo, como vemos a continuación.



Dado que, hasta ahora, nunca fueron necesarios los bloques de esta pestaña y que se usan de una manera muy particular, podemos sugerirles que la exploren durante un breve tiempo y, después, mostraremos cómo son sus especificidades. De paso, podemos aprovechar para aclarar que en Alice se denomina *cadena de texto, cadena de caracteres* o *string* (en inglés) al texto.

Una primera observación importante es que los bloques de esta pestaña no se pueden arrastrar directamente al área del programa. Esto se debe a que representan algún valor (por ejemplo, vemos en la imagen un número aleatorio, una cadena de caracteres unida con otra o algún valor ingresado por el usuario) y, por lo tanto, no son instrucciones en sí mismos sino que deben ser usados dentro de otras instrucciones. Podemos mostrar, por ejemplo, cómo hacer para que un personaje le pregunte el nombre al otro y que el usuario pueda escribir un nombre para que el personaje responda. Preferimos mostrarlo primero, dado que la construcción no es tan intuitiva como con los bloques con los que hemos trabajado hasta ahora.

En este caso, el bloque `preguntar al usuario [ ]` hace que se abra una ventana que dice “Ingrese un nombre” y un campo para ingresar texto. A su vez, el bloque representa el texto ingresado por el usuario, por eso podemos colocarlo en el bloque `dice`, en el mismo lugar donde antes colocábamos un texto fijo para que la flamenco lo diga.

Les indicamos a los estudiantes que la respuesta del usuario debe aparecer varias veces en la conversación de los personajes. Ahora planteamos un nuevo desafío que implica reutilizar la respuesta del usuario. Por ejemplo: el nombre ingresado por el usuario tiene que ser dicho a modo de respuesta por el personaje, pero también deberá formar parte del saludo que formulará el personaje que hizo la pregunta. Es decir, en el ejemplo anterior, si cuando el astronauta preguntó el nombre el usuario ingresó “Flavia”, la flamenco deberá

decir “Flavia”, a lo que el astronauta podría responder “Bienvenida a la Luna, Flavia”. La limitación es que solo se debe preguntar el nombre una vez, para no molestar al usuario.

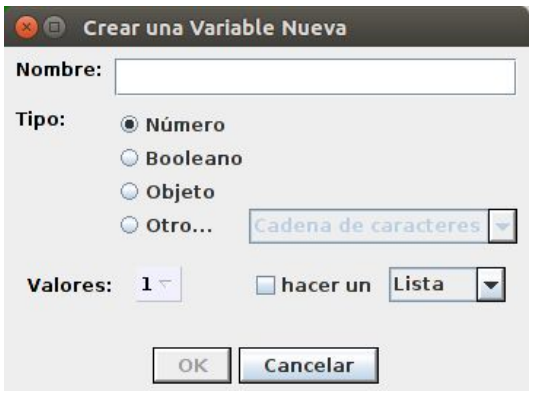
Damos unos minutos para que los grupos identifiquen la necesidad de conservar el texto ingresado por el usuario para poder utilizarlo después en cualquier instrucción del programa, no solo en la de la pregunta.

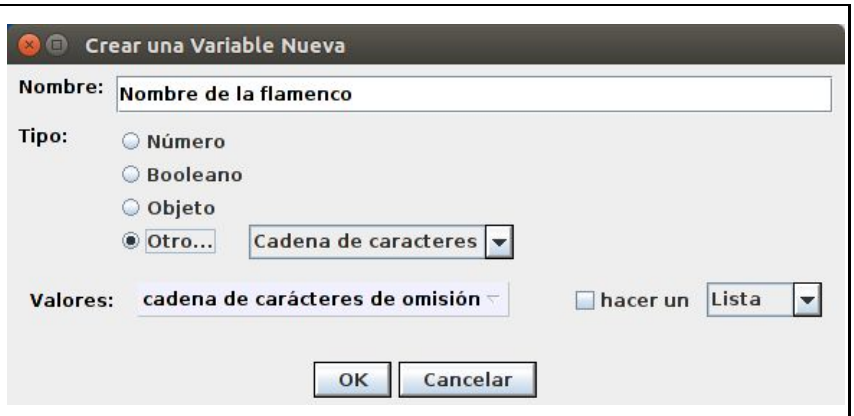
A modo de introducción, proponemos la siguiente dramatización, tomando el ejemplo que mostramos anteriormente, en el que alguien ocupará el rol de la flamenco y otra persona el del usuario. Al usuario se le preguntará el nombre, lo escribirá en un papel y se lo entregará a la flamenco (tal como dice la instrucción), y esta lo leerá en voz alta. Al finalizar, tirará el papel, pues la instrucción finalizó. Preguntamos, entonces, si será posible recuperar el nombre que había introducido el usuario para usarlo en una instrucción posterior, y observamos que no. Preguntamos, entonces, cómo se podría solucionar este problema, para explicitar la necesidad de algún tipo de almacenamiento. En esta instancia, diremos que contamos con una cajita para guardar el papel en el que escribió el usuario y que, además, le pondremos un nombre para referirnos a ella cuando lo necesitemos. Preguntamos qué nombre le podríamos poner para que quede claro su contenido; una buena respuesta podría ser “Nombre de la flamenco”. Repetimos la dramatización, pero cuando el usuario contesta, guarda el papel en la cajita. Cuando la flamenco va a hablar, preguntamos dónde debe buscar la información que necesita, para recordar que a la cajita la habíamos rotulado con un nombre y que esa era la manera que teníamos de referirnos a ella. Luego, la flamenco debe pedir la cajita “Nombre de la flamenco”, y, cuando se la demos, tendrá que abrirla, sacar el papel, leerlo, volver a guardarlo en la cajita y devolverla.

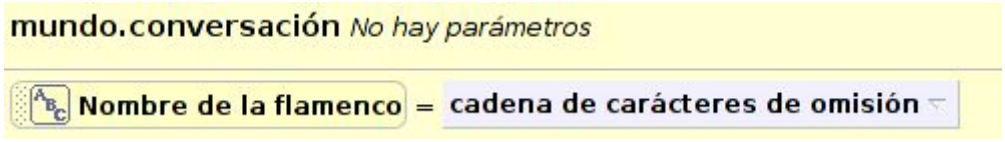
Contamos ahora, finalmente, que en Alice existen *cajitas* para guardar datos y se llaman **variables**. Mostramos a continuación cómo hacer para crear una variable que reproduzca la dramatización anterior.

1. Observamos que las variables pertenecen a los métodos. Por eso, en la parte superior de la definición del método aparece un botón *crear variable nu… (eva)*

2. Al presionarlo, se abre una ventana donde tenemos que completar el nombre que queremos darle a la variable (en este caso “Nombre de la flamenco”) y el tipo. Explicamos que, para Alice, las *cajitas* para almacenar texto son diferentes a las cajitas para almacenar números, que, a su vez, son diferentes de las cajitas para almacenar objetos, etc. Aquello que podemos guardar en una variable se denomina *tipo*; luego, debemos seleccionar “Otro → Cadena de caracteres”. Para terminar, observamos que la variable que creamos acaba de aparecer en la parte superior de la pestaña del método.





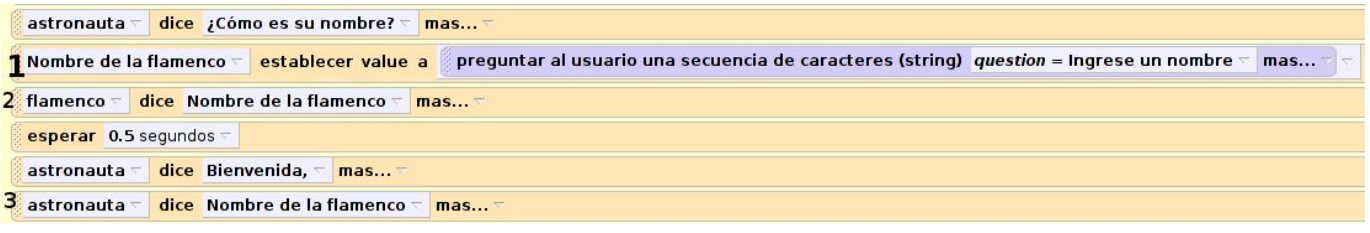


3. El próximo paso es utilizar las variables / cajitas en el programa. Observamos qué sucede cuando arrastramos una variable al área del programa: se genera un bloque `establecer value a` con el nombre de la variable y un espacio para completar con un valor.

   Preguntamos qué se imaginan que hará este bloque, en especial, relacionándolo con la dramatización de la cajita. El objetivo es concluir que esta instrucción *guarda* un valor en la cajita. Podemos agregar que esta operación, en computación, se denomina ***asignación***. Preguntamos, entonces, qué es lo que queremos almacenar en esta variable y de dónde saldrá esta información, para descubrir que debemos asignarle el resultado de la pregunta al usuario.
4. La siguiente pregunta es cómo utilizamos el valor que está almacenado en la cajita / variable. Para esto, mostramos que basta con arrastrar el nombre de la variable al lugar donde queramos usar su valor.

Podemos construir el siguiente ejemplo para ilustrar los puntos que acabamos de comentar.



- En el bloque 1 asignamos la respuesta del usuario a la variable *Nombre de la flamenco*.
- En el bloque 2 se usa el valor de la variable para que sea dicho por la flamenco.
- En el bloque 3 se vuelve a utilizar este valor, esta vez, para que lo diga el astronauta.

Solo resta presentar la operación de unión (o concatenación) de cadenas de caracteres para poder combinar, en una sola cadena, texto fijo escrito por nosotros con texto ingresado por el usuario. Por ejemplo, en el caso que mostramos, el astronauta debería decir “Bienvenida, Flavia” en un mismo globo y no en dos separados y, por lo tanto, es necesario unir estos dos textos para colocarlos dentro de un único bloque `decir`.

Preguntamos a los grupos si ya experimentaron con esto, y en caso de que no lo hayan hecho, damos la pista de que se trata de una función del mundo (al igual que `preguntar al usuario...`) y les indicamos que recuerden con qué tipo de datos queremos operar. El bloque en cuestión es `[a] unido con [b]`, al que le colocamos dos cadenas que queramos unir (una en el lugar de *a* y la otra en el de *b*) y que produce la cadena que es el resultado de unirlas. Como comentamos al principio, este bloque produce un valor (a partir de dos valores) y, por lo tanto, solo puede ser usado dentro de otros bloques. Veamos cómo podemos usarlo en el ejemplo anterior.

En este momento, todos los grupos deberían estar en condiciones de incorporar este tipo de interactividad a sus animaciones. Damos tiempo para que lo hagan y estamos atentos a las dificultades que puedan surgir.

Cierre

Para cerrar esta actividad podemos comentar que el concepto de variable es muy amplio y muy extendido en la programación. En particular, podemos recordar que, como vimos, en Alice existen variables de otros tipos, lo que nos permite almacenar durante la ejecución cualquier valor que necesitemos, como números o incluso objetos de la galería. Además, comentamos que la necesidad de almacenar información durante la ejecución de un programa surge casi constantemente y, por lo tanto, las variables se usan muchísimo. En este sentido, invitamos a los grupos a tenerlas en cuenta en el futuro.

Actividad 5: Condicionales

Objetivos

- Presentar la alternativa condicional como una herramienta para construir programas que no se ejecuten siempre igual.

Modalidad de trabajo

Grupal (3 ó 4)

Materiales

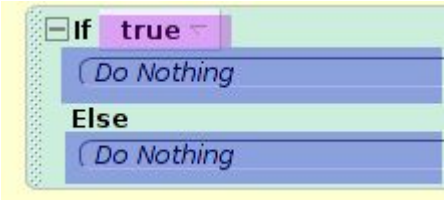
Computadoras
Alice

Desarrollo

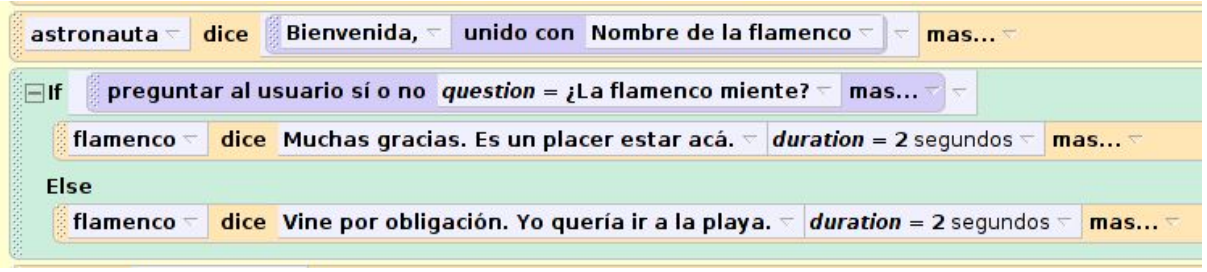
La propuesta para esta actividad es que los estudiantes continúen incorporando a sus animaciones interacciones con el usuario, ahora para decidir el curso de la historia. Para esto, deberán hacer preguntas que se respondan por sí o por no, y, en función de las respuestas, continuar con el programa de una u otra manera. Esperamos, entonces, que observen que los programas que construyeron anteriormente producían siempre la misma animación. Por el contrario, en este caso, el desarrollo de un mismo programa debe poder ser distinto en diferentes ejecuciones, según las respuestas del usuario. Para salvar esta limitación, entonces, presentaremos el bloque de alternativa condicional.

Les planteamos a los estudiantes la siguiente consigna: “Observar que en la sección *pregunta al usuario* existe la posibilidad de preguntar por sí o por no. Antes de que hable un personaje, hacerle una pregunta de este tipo al usuario y que la conversación continúe en función de su respuesta”.

En la actividad anterior vimos cómo incorporar respuestas del usuario; por lo tanto, si es necesario, podemos sugerirles a los grupos que revisen sus trabajos. De todos modos, la dificultad de la consigna surge del hecho de que, hasta ahora, todos los programas que escribieron los estudiantes se ejecutaban siempre igual, es decir, el resultado de todas las ejecuciones era siempre exactamente el mismo. Sin embargo, la nueva consigna propone romper esta limitación de modo que, si en distintas ejecuciones el usuario toma decisiones distintas, la animación resulte distinta. Dejamos un tiempo para que exploren el entorno y, sobre todo, reconozcan esta dificultad. Cuando todos los grupos hayan observado que no han visto ninguna manera de que la animación varíe entre diferentes ejecuciones, presentamos el bloque `if/else` (*si / si no*, en castellano) y mostramos que se construye a partir de tres partes fundamentales:

	• El bloque marcado con violeta, al que solemos llamar *condición*, es una pregunta que se responde con sí o con no (o verdadero o falso, *true* o *false* en inglés). • Observamos que los dos sectores marcados con azul permiten encastrar bloques con instrucciones, de la misma manera que los bloque `hacer juntos` o `hacer en orden`. • Si la respuesta a la pregunta fue sí (verdadero, *true*), se ejecutan los bloques del primer sector azul, mientras que si fue no, los del segundo. Dado que a partir del resultado de la condición existe una bifurcación en las ejecuciones posibles del programa, a cada uno de estos sectores se lo denomina *rama*.

Ahora sí, dejamos tiempo para que los grupos exploren el funcionamiento del bloque y vean cómo pueden integrarlo a sus animaciones. Podemos sugerirles que experimenten primero con una bifurcación sencilla, como por ejemplo “¿El personaje miente?” y que, en base a esto, se produzca una u otra respuesta, como vemos a continuación:



Una observación, que también retoma una discusión de la actividad anterior, es que el bloque con la pregunta está colocado directamente en la condición del bloque `if/else`. Esto es análogo a cuando hacíamos `decir [preguntar al usuario]...` pues, recordamos, el bloque de la pregunta, además de hacer que se abra la ventana para escribir la respuesta, representa el valor ingresado. Por este motivo, debemos colocarlo en los bloques que esperan ser completados con valores (como `decir` o un condicional). De la misma manera que antes, si quisiéramos volver a utilizar esta respuesta en algún momento, deberíamos almacenarla en una variable. Esto se haría exactamente de la misma manera, excepto que, en el momento de crear la variable, deberíamos especificar que sea de tipo booleano, dado que este es el tipo de los datos para representar si / no (verdadero / falso, *true* / *false*).

Cuando los grupos se hayan familiarizado con esta nueva construcción, interrumpimos el trabajo para hacer una puesta en común. Podemos pedir a algunos grupos que nos cuenten qué alternativa incorporaron y que nos muestren cómo lo programaron en Alice. Entonces, aprovechamos para reconocer entre todos correspondencias entre ideas expresadas en la lengua natural y bloques programados en Alice. En particular, nos interesa observar:

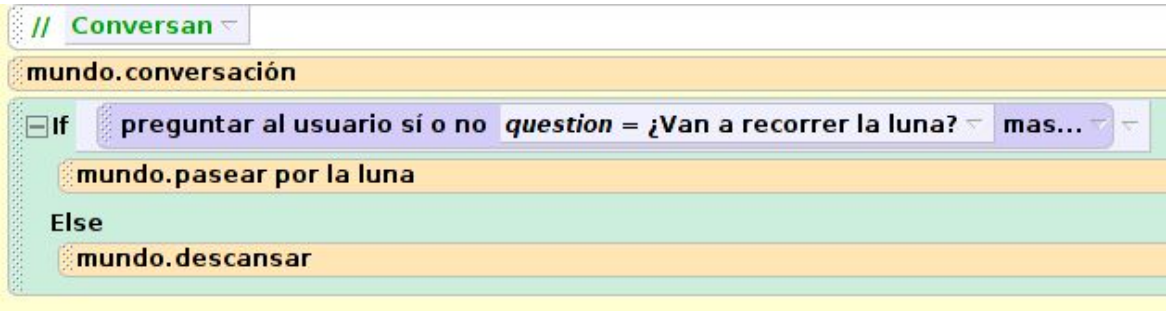
- Que el bloque expresa una ejecución supeditada a una condición: si sucede tal cosa, debe suceder esta otra; si no, debe suceder esta otra. De hecho, este es exactamente su significado en inglés: si (if) _____, _____, si no (else).
- Que la condición es una pregunta que se responde por sí o por no. En este caso es evidente, pues es exactamente la pregunta que hicimos, pero existen otras condiciones que pueden ser interesantes, como por ejemplo: ¿El flamenco está cerca del astronauta?, ¿El usuario se llama Pepe?, etc. Pedimos a los grupos que enuncien condiciones posibles en sus programas y digan qué debería suceder en caso de que fueran verdaderas o falsas.

Una vez que se hayan familiarizado con el nuevo bloque, les proponemos la segunda consigna, que continúa en esta misma dirección. La consigna consiste en hacerle una(s) pregunta(s) (de sí / no) al usuario y que el curso de la historia dependa de esta(s) pregunta(s). Prestar atención a la definición de métodos adecuados para que el programa sea claro y pueda comprenderse cómo se encadenan las distintas opciones.

Esta nueva consigna no plantea ninguna novedad técnica; simplemente propone utilizar la nueva herramienta para enriquecer las animaciones de manera que los grupos puedan experimentar con sus historias. A lo que debemos sugerirles que presten atención, es a la definición de métodos. De la misma manera que hicimos en la actividad correspondiente cuando separamos cada uno de los momentos de la escena en un método, ahora esperamos que definan uno para cada alternativa posible de la historia. De esta manera, al llegar al bloque de la alternativa condicional no habrá demasiadas instrucciones dentro de cada rama, sino más bien un único método que deje en claro de qué se trata ese momento de la animación. Observemos que, como las bifurcaciones pueden producirse en cualquier momento de la historia, puede ser necesario colocar el condicional en cualquiera de los métodos definidos.

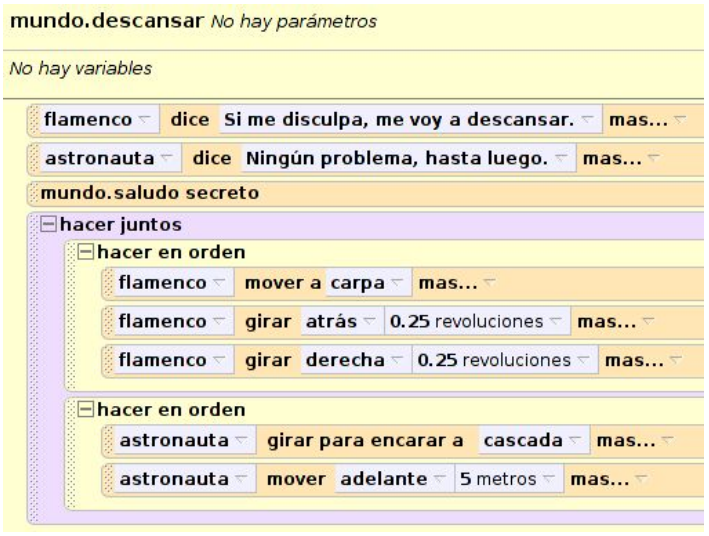
Para tener en cuenta al trabajar con Alice:

- Haciendo clic con el botón derecho sobre una instrucción, método o bloque aparece la opción *deshabilitar*. Si la elegimos, veremos que el bloque en cuestión queda rayado y gris, lo que indica que no se ejecutará durante la animación. Esto puede resultar útil durante el desarrollo, por ejemplo, para “saltearse” la primera parte y avanzar directamente a la escena que se está ajustando.

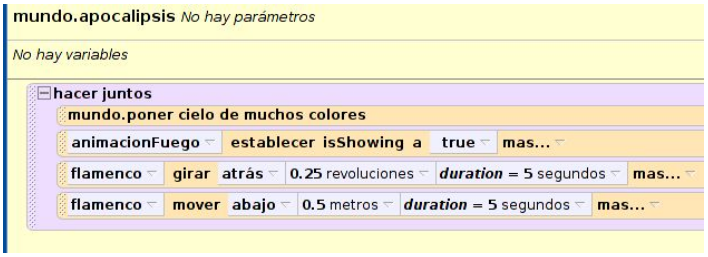


`mundo.acción`: Después de la conversación, ocurre la primera bifurcación, en la que podemos elegir si los personajes van a pasear por la luna o la flamenco se va a descansar. Observamos que definimos un método para cada alternativa.

`mundo.pasear por la luna`: Si elegimos que los personajes vayan a pasear por la luna, encuentran una cascada en la que la flamenco se baña. Ahí podemos elegir otra vez si el baño es peligroso o no. Observamos que la nueva alternativa condicional aparece al final del método que habíamos definido para esta situación (`mundo.pasear por la luna`) y puede incluir nuevos métodos que representen nuevas situaciones (como `mundo.apocalipsis` y `mundo.chapuzón`).



`mundo.descansar`: Si elegimos no ir a pasear por la luna, los personajes se saludan y la flamenco se acuesta dentro de la carpa. Fin de la animación.



`mundo.apocalipsis`: Si el agua lunar resulta peligrosa, el cielo cambia de color y la flamenco se hunde.

`mundo.chapuzón`: Si el agua no es peligrosa, el astronauta acompaña a la flamenco y nadan juntos en la cascada.

Cierre

Para concluir, observamos que en esta actividad profundizamos lo que habíamos comenzado en la anterior: que el usuario pueda controlar algún aspecto del comportamiento del programa. Esto se parece mucho más a cómo se comportan los programas que utilizamos a diario (por ejemplo, un navegador de Internet no abre siempre la misma página ni un videojuego muestra siempre el mismo recorrido; todo depende de cómo nosotros interactuemos con ellos).

ANEXO II

PROGRAMACIÓN TEXTUAL

Los entornos de programación como Alice presentan grandes ventajas a la hora del aprendizaje: su propuesta lúdica permite que los estudiantes exploren su creatividad, el sistema de bloques encastrables evita errores y dificultades, etc. No obstante, para escribir programas complejos, suelen utilizarse lenguajes de programación textuales. En ellos, los programas se construyen escribiendo texto y no encastrando bloques.

En este anexo se introduce distintas actividades y proyectos para ser desarrollados usando el lenguaje textual Python. Actualmente es un lenguaje muy utilizado, y su filosofía hace hincapié en una sintaxis que favorece el desarrollo de programas que, al ser leídos, puedan comprenderse con relativa facilidad.

Anexo 2: Programación textual

Los entornos de programación como Alice presentan grandes ventajas a la hora del aprendizaje: su propuesta lúdica permite que los estudiantes exploren su creatividad, el sistema de bloques encastrables evita errores y dificultades, etc. No obstante, para escribir programas complejos, suelen utilizarse lenguajes de programación textuales. En ellos, los programas se construyen escribiendo texto y no encastrando bloques.

En este anexo se introduce distintas actividades y proyectos para ser desarrollados usando el lenguaje textual Python. Actualmente es un lenguaje muy utilizado, y su filosofía hace hincapié en una sintaxis que favorece el desarrollo de programas que, al ser leídos, puedan comprenderse con relativa facilidad.

El anexo está dividido en dos partes: en la primera, se presenta el lenguaje y se realiza una aproximación a la resolución de problemas que involucran el procesamiento de secuencias de datos; en la segunda, se propone un proyecto para realizar un sintetizador de voz.

Parte 1: Procesamiento de secuencias

- **Secuencia didáctica 1**: Lenguajes textuales y secuencias de datos
  - **Actividad 1**: Adivina, adivinador
  - **Actividad 2**: ¿Podremos buscar más rápido?
  - **Actividad 3**: Secuencias en Python
- **Secuencia didáctica 2**: Programamos con imágenes
  - **Actividad 1**: Programamos en blanco o negro
  - **Actividad 2**: Programamos en color
  - **Actividad 3**: Hacemos nuestros filtros

Parte 2: Proyecto de Sintetizador

- **Secuencia didáctica 3**: Programas que suenan
  - **Actividad 1.A**: ¿Cómo es el sonido?
  - **Actividad 1.B**: ¿Cómo se representa el sonido?
  - **Actividad 2**: Manipulamos sonido
  - **Actividad 3**: Generamos sonidos
  - **Actividad 4:** Generamos melodías

Parte I: Procesamiento de secuencias

Las computadoras son, en esencia, máquinas preparadas para trabajar con grandes cantidades de datos, y una de las maneras más usuales en las que estos se presentan es en forma de **secuencias**. Gran parte de la información que pasa por nuestras computadoras consiste en secuencias de datos: como ya vimos, un texto es una secuencia de caracteres, y una imagen, una secuencia de píxeles; en el capítulo siguiente veremos que también el sonido se representa en forma de secuencia.

En este capítulo, conoceremos algunas estrategias o **algoritmos** para resolver problemas usuales que involucran trabajar con secuencias. Además, como gran novedad, los programaremos usando Python, que a diferencia de Alice, no está basado en bloques encastrables sino que se trata de un lenguaje completamente textual.

Terminaremos el capítulo aplicando estas nuevas herramientas de manipulación de secuencias para hacer programas que generen o modifiquen imágenes y permitirle a los estudiantes experimentar con ellas.[1]

[1] Este recorrido y la mayoría de las actividades están adaptadas de “Propuesta de planificación anual para Tecnologías de la Información, 4to año de la NES (TI4), CABA”, disponible en http://bit.ly/2HeXjJN

Secuencia didáctica 1: Lenguajes textuales y secuencias de datos

Los entornos de programación como Alice, que utilizamos en el primer capítulo de este manual, presentan grandes ventajas a la hora del aprendizaje: su propuesta lúdica permite que los estudiantes exploren su creatividad, el sistema de bloques encastrables evita errores y dificultades. No obstante, para escribir programas complejos, suelen utilizarse lenguajes de programación completamente textuales.

En esta secuencia didáctica, los estudiantes escribirán sus primeros programas en Python, un lenguaje textual que resulta flexible y amigable para programar. Lo harán a la vez que resuelven problemas relacionados con secuencias de datos, una manera muy simple pero muy poderosa de representar grandes cantidades de información en Python (como por ejemplo, todas las palabras de un libro o los píxeles de una imagen). Además del aspecto técnico, trabajaremos con algoritmos generales para algunos de estos problemas.

Objetivos

- Presentar la estructura de repetición condicional
- Conocer la existencia de lenguajes de programación textuales.
- Introducir el lenguaje Python, a partir de su comparación con Alice.
- Presentar e implementar los algoritmos de búsqueda lineal y binaria
- Presentar la estructura de Python para representar secuencias: las listas.
- Presentar el recorrido sobre secuencias (ciclos for) y utilizarlo para resolver problemas.
- Presentar y comenzar a ejercitar la noción de parametrización.
- Escribir e implementar un algoritmo para encontrar un máximo.

Actividad 1 | SD1

Adivina, adivinador

Objetivos

- Presentar la estructura de repetición condicional.
- Trabajar con un algoritmo de búsqueda lineal.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras
- Alice
- Proyector (opcional).

Desarrollo

El resultado de esta actividad será el punto de partida de la siguiente, en la cual comenzaremos a trabajar con Python, un lenguaje de programación textual. Dado que los primeros pasos en este tipo de lenguajes pueden ser muy ásperos, proponemos primero escribir un programa no trivial en Alice y, en la actividad siguiente, observar cómo podemos "traducirlo" a nuestro nuevo lenguaje, aprovechando la riqueza de la nueva herramienta.

Los primeros minutos podemos emplearlos, si resulta necesario, para hacer un rápido repaso de Alice. Luego les proponemos que, trabajando en grupos de dos personas, resuelvan la siguiente consigna, como un juego sencillo:

> **Consigna**:
> *El programa le pedirá al jugador que elija un número entre 1 y 10. El usuario **no** debe decirle al programa de qué número se trata. Luego, el mismo programa debe tratar de adivinar el número elegido. Para ello, el programa podrá ir haciendo preguntas al usuario, como "¿el número que elegiste es X?", donde "X" representa algún número entre 1 y 10.*

Al igual que en proyectos anteriores, no será especialmente relevante qué personajes u otros elementos escenográficos empleen los estudiantes en su solución; dejaremos que los elijan según sus gustos y preferencias, y nos preocuparemos únicamente por ayudarlos a lograr el comportamiento esperado.

Existen varias formas o **estrategias** para resolver el problema, ya que el número elegido por el usuario se puede adivinar siguiendo estrategias variadas. Una de las más sencillas es ir preguntando uno por uno por cada uno de los números, en orden ascendente:

- ¿El número que elegiste es 1?
- ¿El número que elegiste es 2?
- ¿El número que elegiste es 3?
- …

En el momento en que el usuario responde afirmativamente a una de las preguntas, el juego termina y dice “el número que elegiste es *Z*”.

Si intentamos escribir en Alice un programa que siga esta estrategia, notaremos que no hay una única forma de hacerlo. Por ejemplo, si tenemos que usar alguna variable, podríamos darle diferentes nombres; o quizás decidamos usar un bloque de repetición en lugar de repetir explícitamente alguna instrucción. Es común que existan distintos programas para una única estrategia, que difieren en aspectos meramente accidentales. A la estrategia que elegimos para resolver el problema se le llama usualmente **algoritmo**, mientras que un programa que aplica esa estrategia es una **implementación** del algoritmo. Esto significa que pueden existir muchas maneras de implementar un mismo algoritmo. Como veremos más adelante en esta actividad, incluso es posible escribir implementaciones para un mismo algoritmo en lenguajes de programación distintos.

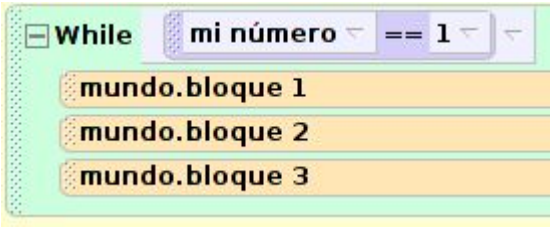
El algoritmo que acabamos de proponer para resolver el problema del adivinador suele recibir el nombre de **búsqueda secuencial**. Un primer intento para implementarlo en Alice puede consistir en una **repetición simple** (bloque `lazo`) que repita 10 veces la cantidad de instrucciones que se encuentran en su interior, de manera que cada ejecución del lazo prueba con un número distinto. Para poder escribir esta solución será necesario utilizar otras tres herramientas que se trabajaron en los capítulos anteriores, aunque con algunas particularidades:

- Cada vez que se ejecuta la repetición, el adivinador debe recordar qué número le toca preguntar a continuación, de manera que cada vez pregunte por un número distinto. Para esto puede utilizarse una **variable** `Número a probar`, que comienza valiendo 1 y a la cual se le va sumando 1 cada vez que se ejecuta el lazo. En este caso, y a diferencia de los que sucedió en el capítulo anterior, es importante que el valor almacenado en la variable cambie en cada iteración del ciclo. Para eso, aparece la instrucción `aumentar`.
- Para cada número, se le debe preguntar al jugador si es o no el que había pensado originalmente. Para esto puede utilizarse la **función** `preguntar al usuario sí o no`. Por cuestiones técnicas (que mencionamos cuando vimos la creación de variables), en Alice los valores (y por lo tanto, las variables que los almacenan) tienen un tipo de dato asociado (número, texto, objeto, etc.) y las operaciones están preparadas trabajar con uno de ellos en particular. Por ejemplo, la operación `unido con` espera que los dos valores sean de tipo texto. Por este motivo, para poder concatenar el valor de `número a probar` con “El número que pensaste….”, primero es necesario aplicarle la operación `como secuencia de caracteres (string)`, para, justamente, expresar el número como un texto.
- Por último, el comportamiento del programa debe ser distinto según cuál sea la respuesta del usuario; para lograr esto podemos usar una **alternativa condicional** (bloque `if/else`) dentro del cuerpo de la repetición.

```
mundo.adivinador ( )
  número a probar = 1
    Mago adivinador dice Pensá un número entre 1 y 10
    esperar 2 segundos
    Lazo o Ciclo 10 veces
      If ( preguntar al usuario sí o no ( ¿El número que pensaste es unido con ( ( ( int número a probar ) como secuencia de caracteres (string) ) unido con ? ) ) )
        Mago adivinador dice ( ¡Adiviné! El número que pensaste es unido con ( ( int número a probar ) como secuencia de caracteres (string) ) )
      Else
        aumentar número a probar de 1 en 1
```

Esta solución presenta un problema, que esperamos que los grupos observen: aunque el programa haya adivinado el número, seguirá preguntando por todos los números hasta el final, ya que la cantidad de veces que se ejecuta la repetición (diez) está fija. Para poder resolver este inconveniente, entonces, presentamos la **repetición condicional** (bloque `while`), de manera de poder interrumpir la repetición cuando sea adivine el número.

La repetición condicional es una estructura muy frecuente en los lenguajes de programación para indicar repeticiones. A diferencia de la repetición simple, en la que la cantidad de veces que debe repetirse un fragmento del programa es un número fijo que está dado de antemano, en este caso, la decisión de repetir o no un conjunto de instrucciones está dada por una condición. El bloque `while` con el que contamos en Alice para expresar esta estructura funciona de la siguiente manera:



Bloque `while` en Alice.

Primero se evalúa la condición. Si el resultado es falso, se ignora el cuerpo del bloque y se continúa con la ejecución del resto del programa. Si el resultado es verdadero, se ejecuta el cuerpo del ciclo. Al finalizar, se evalúa la condición nuevamente y se repite el comportamiento de antes: si es falso, se continúa con el programa, si es verdadero, se vuelve a ejecutar el cuerpo del ciclo y, después, a evaluar la condición.
Esto produce que el cuerpo del ciclo se repita **mientras** que la condición sea verdadera (o hasta que la condición sea

	falsa). Justamente, éste es el significado de la palabra *while* en inglés: mientras.

Existen varias maneras de modificar el programa para poder utilizar una repetición condicional; a continuación presentamos una de ellas. Si reemplazamos el bloque `lazo` por un bloque `while`, el contenido de la repetición ya no se ejecuta siempre, sino que antes se verifica si se cumple una determinada condición. En este caso, nos gustaría que la repetición solo se ejecute si el programa todavía *no* adivinó el número del usuario. Por lo tanto, tenemos que recordar qué fue lo que respondió el usuario la vez anterior. Para eso podemos usar una variable de tipo booleano; recordemos que las variables de tipo booleano almacenan un *valor de verdad*, que puede ser verdadero (`true`) o falso (`false`). Dado que la variable recuerda si el programa ya adivinó el número, el nombre elegido para ella es `adivinó`, mientras que su valor inicial es `false`.

```
mundo.adivinador ( )
  número a probar = 1
  adivinó = false
  Mago adivinador dice Pensá un número entre 1 y 10
  esperar 2 segundos
  While ( no adivinó )
    If ( preguntar al usuario sí o no ( ¿El número que pensaste es unido con ( ( ( int número a probar ) como secuencia de caracteres (string) ) unido con ? ) ) )
      adivinó establecer value a true
      Mago adivinador dice ( ¡Adiviné! El número que pensaste es unido con ( ( int número a probar ) como secuencia de caracteres (string) ) )
    Else
      aumentar número a probar de 1 en 1
```

Esta estrategia, que consiste en ir observando valores y probar para cada uno si es el buscado hasta que lo encontramos, se conoce como **algoritmo de búsqueda lineal**. Podemos preguntarnos en qué situaciones de la vida cotidiana la aplicamos, para observar que es muy frecuente: cuando buscamos una carta en un mazo, cuando buscamos en un manojo de llaves la correcta para abrir una puerta, cuando elegimos en un perchero de venta de ropa una prenda que nos guste, etc. .

Cierre

Para terminar la actividad, podemos recordar que cuando introdujimos la alternativa condicional, pudimos construir programas (animaciones) que no siempre se ejecutaban de la misma manera. En esta misma dirección, la repetición condicional es aún más poderosa: nos permite escribir programas que repitan una serie de instrucciones sin necesidad de conocer cuántas veces debe hacerse esta repetición, simplemente expresando una condición bajo la cual debe suceder. De hecho, podemos preguntarnos “¿Qué pasará si siempre se cumple la condición de la repetición condicional?”, para observar que siempre se repetirá la ejecución de las instrucciones del ciclo y, por lo tanto, el programa no terminará nunca. Esta es una de las razones por las cuales los prorgamas que usamos a veces dejan de responder (o, como decimos informalmente, “se cuelgan”).

Actividad 2 | SD1

Adivina, adivinador, ahora como texto.

Objetivos

- Conocer la existencia de lenguajes de programación textuales.
- Trabajar con un entorno de desarrollo orientado a Python.
- Introducir el lenguaje Python, a partir de su comparación con Alice.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras
- Alice
- Python (preferentemente PyCharm Edu).
- Proyector (opcional).

Desarrollo

Después de haber terminado el programa de la actividad anterior, finalmente buscaremos que los estudiantes tengan un primer contacto con un **lenguaje de programación textual**: Python[2], ya que lo usaremos de ahora en adelante para abordar algunos desafíos de programación para los que Alice resulta demasiado limitado. Este primer contacto puede parecer desalentador al principio, principalmente, por los siguientes motivos:

- Los lenguajes textuales, al no contar con bloques para construir visualmente la estructura de programa, tienen una sintaxis muy estricta. Esto significa que, para que los programas estén correctamente formados y puedan ser interpretados, la manera en la que están escritos debe respetar reglas muy precisas y específicas. Por este motivo, una equivocación tan simple como omitir un símbolo puede producir que el programa ni siquiera pueda ser ejecutado para observar su comportamiento.
- No se cuenta con la motivación lúdica de poder ver directamente a los personajes en 3D ejecutando los programas.

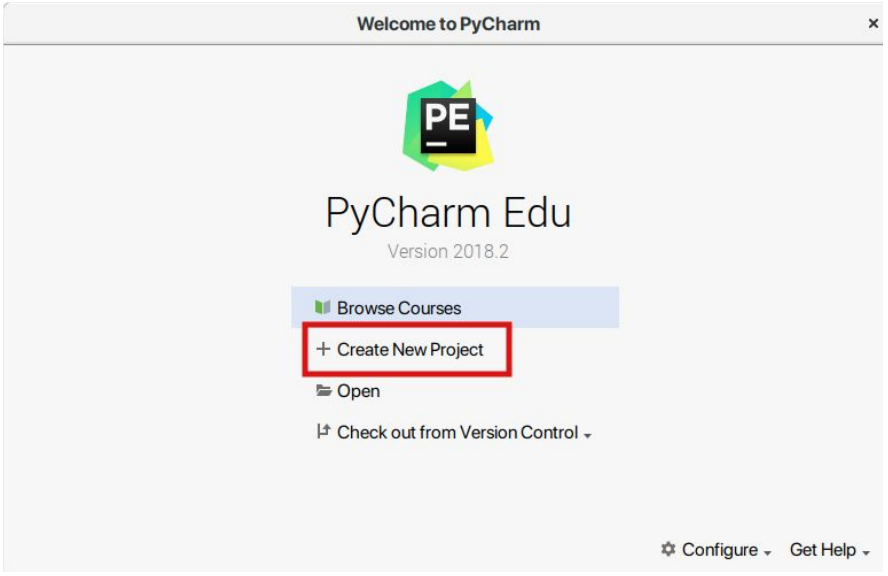
Por lo tanto, al dar los primeros pasos tendremos que ser especialmente pacientes ante estas dificultades, teniendo presente y recordando a nuestros estudiantes que están aprendiendo a utilizar una herramienta muy poderosa, con la que luego encararemos proyectos muy interesantes, como un procesador de imágenes y de sonidos. Por otro lado, podemos observar que con Alice no es posible producir software similar al que ellos usan en

[2] Actualmente existen dos versiones diferentes de Python: Python 2 y Python 3, que si bien son similares, no son enteramente compatibles. Para el desarrollo de este manual, elegimos trabajar con la primera, con lo cual, cuando digamos Python estaremos haciendo referencia a Python 2.

sus netbooks, celulares, tablets y otros dispositivos. Sin embargo, todas estas aplicaciones están construidas con lenguajes textuales (incluso con el que trabajaremos nosotros).

También es importante resaltar que, más allá de diferencias en la apariencia, los conceptos fundamentales que trabajaremos son los mismos. Como ya mencionamos, debemos estar atentos a las dificultades que puedan surgir al comienzo, teniendo en cuenta será necesario aprender una nueva forma de escribir los programas donde es posible cometer errores de sintaxis, pero donde todas las herramientas que utilizamos anteriormente para pensar y construir programas siguen siendo válidas.

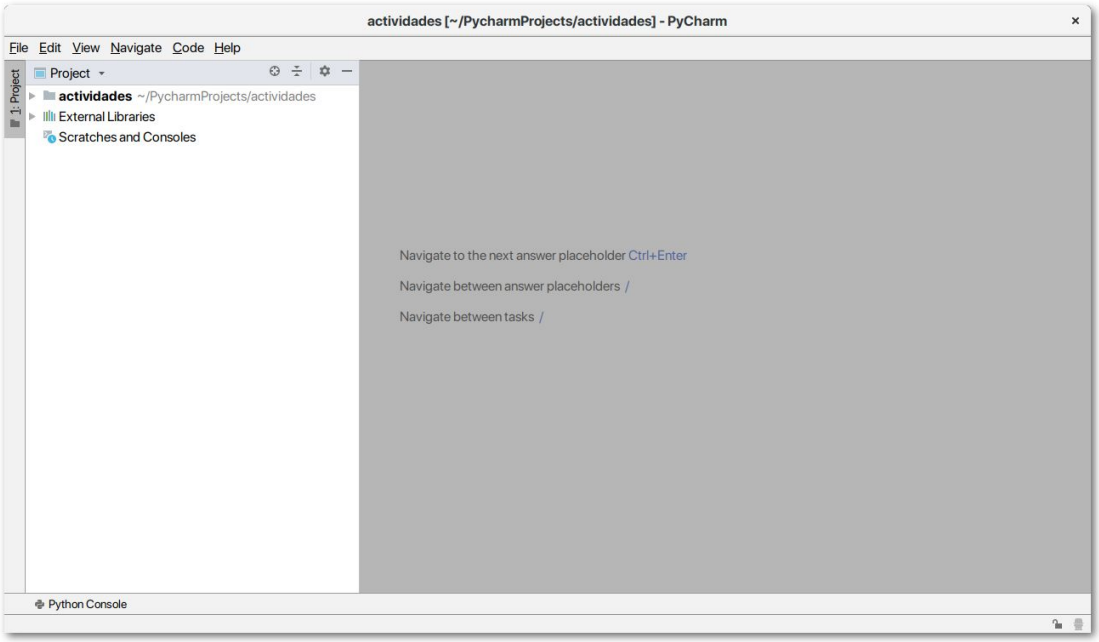
Todo el trabajo que venimos haciendo en Alice lo hicimos dentro de un **entorno de programación**, es decir, una pieza de software que permite tanto escribir y modificar programas como ejecutarlos y ver sus resultados. Aunque un programa de Alice no puede ejecutarse fuera de su entorno de programación, este no es el caso para los programas de Python, lo cual amplía las posibilidades de lo que puede hacerse con el lenguaje. No obstante, también existen entornos de programación que simplifican la tarea de escribir y probar nuestros programas. El desarrollo de las actividades de este manual está escrito utilizando un entorno con fines didácticos llamado PyCharm Edu.[3] El entorno se encuentra en inglés, pero siguiendo atentamente las instrucciones, el idioma no debería resultar un obstáculo para utilizar las funcionalidades necesarias para escribir y ejecutar nuestros programas.



Pantalla inicial de PyCharm Edu

[3] Existe una gran variedad de entornos de programación para trabajar con Python, por lo que podemos cambiarlo si existe otro con el que nos sentimos más cómodos. Más aún, los programas de Python son simples archivos de texto que contienen las instrucciones, por lo que no es necesario usar un entorno de programación para modificarlos; se puede usar cualquier *software* editor de textos, incluyendo algunos especializados en la edición de programas (por ejemplo, Atom, Geany o Notepad++). En ese caso, para probar los programas es necesario contar otro *software* llamado intérprete, que se encarga de ejecutar las instrucciones.

Al iniciar PyCharm Edu, se presentará una pantalla con varias opciones. Elegiremos la opción "*Create New Project*" (Crear proyecto nuevo). Se nos pedirá que indiquemos dónde queremos crear el proyecto, y luego se abrirá una ventana dividida en dos paneles principales.



El panel izquierdo (con fondo blanco) contiene la lista de los archivos de nuestro proyecto, mientras que en el panel derecho (con fondo gris) veremos las instrucciones de nuestros programas.

Una de las características interesantes de este entorno es que, además de escribir y ejecutar programas, podemos probar instrucciones "sueltas" usando la **consola** de Python. Invitamos a los estudiantes a abrir el entorno y comenzar a familiarizarse con las instrucciones del lenguaje usando la consola. Si contamos con un proyector, podemos ejecutar nosotros las instrucciones y pedir a los estudiantes que lo repliquen en sus computadoras; de lo contrario, tendremos que dar indicaciones muy claras para que ellos puedan hacerlo.

Para abrir la consola, hacemos clic en el botón "*Python Console*" que aparece en la parte inferior. Se abrirá un panel en la parte inferior del entorno, dividido en dos sectores; nosotros trabajaremos con la parte izquierda. Los símbolos `>>>` en color verde indican el lugar donde podemos escribir instrucciones para luego ver sus resultados.

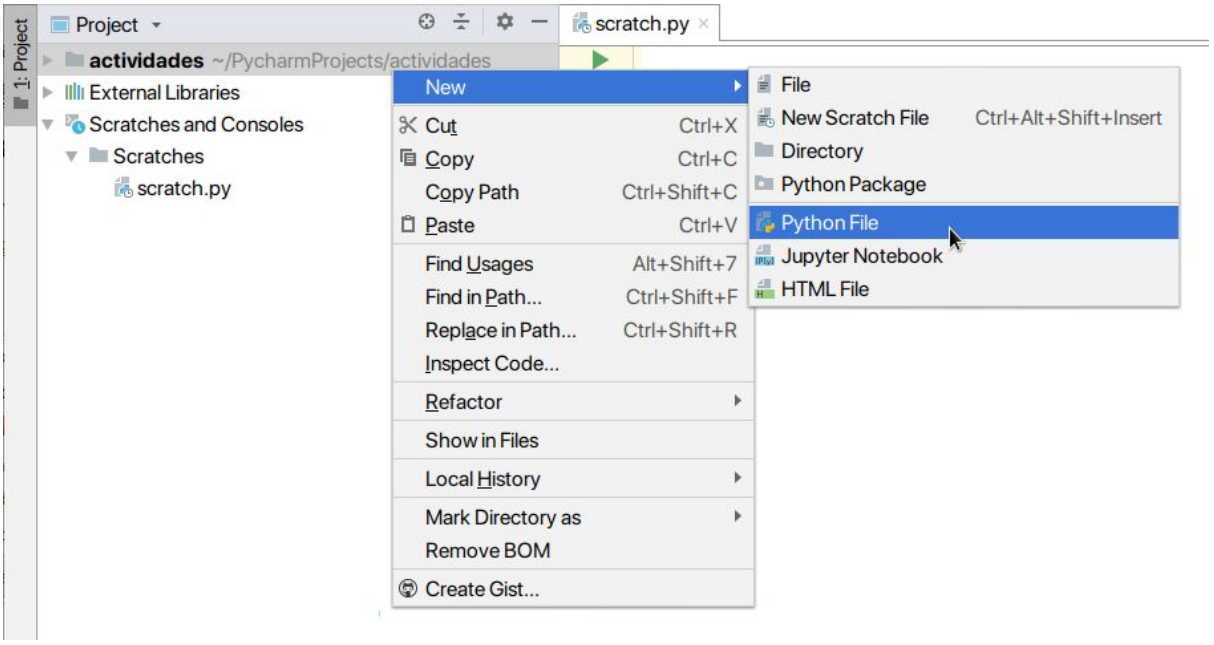
La consola de Python

Ya no tenemos personajes para que “digan” cosas al usuario; en lugar de eso tendremos que *imprimir* los resultados por pantalla, lo cual puede hacerse con la instrucción `print`.

```
>>> print "Hola, mundo. Esta es mi primera instrucción en Python."
Hola, mundo. Esta es mi primera instrucción en Python.
>>> print 4 + 12
16
>>> print 20 > 13
True
```

En este ejemplo, se imprimieron tres tipos distintos de valores: un fragmento de texto, un número (que es el resultado de realizar una operación matemática entre otros dos números) y un valor de verdad o *booleano* (que resulta de preguntarse si un número es o no mayor que otro).

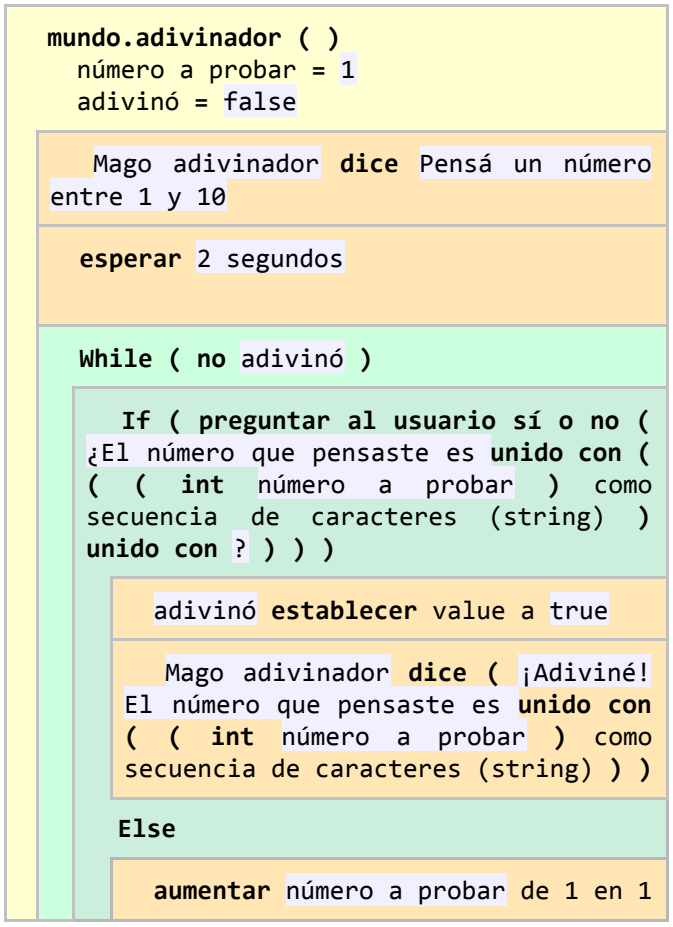
La consola es útil para probar instrucciones de a una, pero si queremos escribir un programa que contenga muchas instrucciones, será mejor que lo hagamos en un archivo con tal fin. Para crear un nuevo archivo, hacemos clic con el botón derecho del ratón en la carpeta de nuestro proyecto (en el panel izquierdo), y elegir las opciones *New* > *Python File* (Nuevo > Archivo de Python) en el menú que se despliega. Tras brindar un nombre para nuestro archivo, se abrirá en el panel derecho y podremos empezar a escribir un programa.



Creación de un nuevo archivo de código

Con el objetivo de comenzar a conocer este nuevo lenguaje, proponemos ”traducir” a Python la solución al problema del adivinador. De ser posible, la idea es realizar la programación de manera interactiva y con un proyector, mostrando en pantalla las instrucciones a medida que las vamos escribiendo, pero dejando que sean los estudiantes quienes vayan sugiriendo cuál debería ser la siguiente idea a incorporar al programa, guiándose por la solución ya escrita en Alice. Ante cada nueva instrucción, mostraremos la

similitud con las instrucciones de Alice. Una solución posible es la que se muestra a continuación.




```
1  # -*- coding: utf-8 -*-
2  import time
3
4  numero_a_probar = 1
5  adivino = False
6
7  print "Pensá un número entre 1 y 10"
8  time.sleep(2)
9
10 while not adivino:
11       respuesta = raw_input("¿El número que
   pensaste es " + `numero_a_probar` + "?")
12    if respuesta == "sí":
13        adivino = True
14        print "¡Adiviné! El número que pensaste
   es " + `numero_a_probar` + "."
15    else:
16        numero_a_probar = numero_a_probar + 1
```

Versión en lenguaje Python de la solución previa al problema del adivinador,[4] y comparación con el programa escrito en Alice.

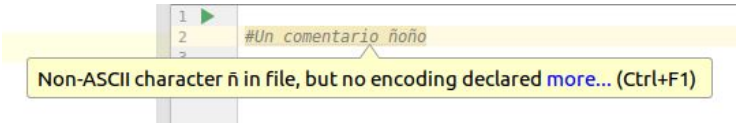
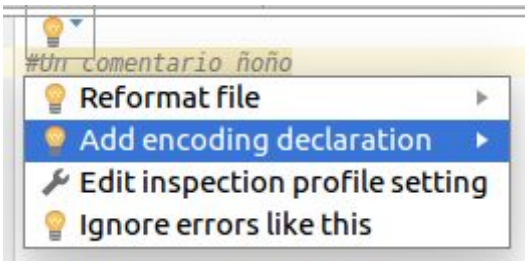
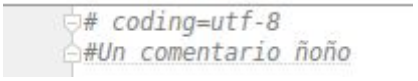
Las primeras dos líneas del programa se encuentran allí por razones técnicas. La línea 1 permite que se puedan utilizar ciertos caracteres, como letras acentuadas o signos de interrogación de apertura. La línea 2, por su parte, permite que en nuestro programa se utilicen procedimientos de la **biblioteca** `time`. Una biblioteca es un conjunto de métodos[5] ya implementados que podemos utilizar en nuestros programas; esta biblioteca en particular incluye diversas funcionalidades relacionadas al manejo del tiempo, y la utilizaremos en la

[4] Los números al principio de cada línea no forman parte del programa. La mayoría de los editores de texto muestran numeradas las líneas del programa para poder hacer referencia a ellas de manera sencilla. Cabe destacar que, como se ve en el ejemplo, en el programa pueden intercalarse líneas en blanco para hacerlo más sencillo de leer.

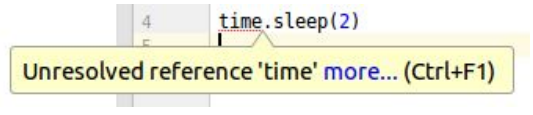
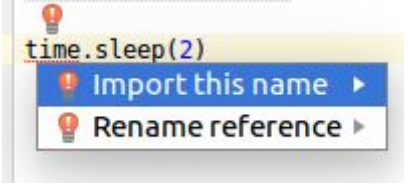
[5] Python utiliza la palabra función (function) en vez de método. Existe una diferencia, pero por el momento no haremos la diferencia para subrayar la similitud con Alice.

línea 8 del programa, al invocar el procedimiento `sleep(2)` para que el programa nos dé dos segundos para pensar el número antes de comenzar a intentar adivinarlo.

IDE Tip
Una enorme ventaja de los entornos de programación es que, en general, detectan errores a medida que vamos escribiendo el programa y nos permiten solucionarlos (casi) automáticamente.
Por ejemplo, si utilizamos un carácter que no está permitido con la codificación estándar de caracteres (como la ñ): la línea se resalta y si pasamos el cursor por encima nos muestra un mensaje explicando el error (está en inglés, pero podemos identificar que el problema es la ñ y que hay un problema con el encoding -codificación-). Además, aparece una lamparita que, si la apretamos, nos ofrece distintas soluciones, como por ejemplo, agregar la declaración de la nueva codificación (*Add encoding declaration*). A estas opciones también podemos acceder si colocamos el cursor de texto sobre el error y presionamos Alt + Enter.

	
La línea resaltada, el mensaje de error y las opciones que nos ofrece para solucionarlo.	
Al aceptar la sugerencia de la lamparita, se agrega la línea con la declaración de la codificación y desaparece el error.	

IDE TIP
De la misma manera que con la codificación, el entorno nos señala si nos falta importar una biblioteca.

El mensaje “Unresolved reference” es un error frecuente que significa que estamos utilizando algo (en este caso, una biblioteca) que no definimos en ningún lado (es decir,

hay una palabra para la que no se puede resolver a qué está haciendo referencia). Una de las opciones posibles para solucionarlo es importar una biblioteca que lo defina (por eso, *Import this name* es una sugerencia posible).

Continuando con el texto del programa, `while condición:` indica el comienzo de una repetición condicional. En Alice, este era un bloque especial, porque podía contener otros bloques en su interior. El equivalente en Python se logra agregando cuatro espacios al comienzo de las instrucciones que están dentro de la repetición, lo cual permite marcar visualmente que tienen menor jerarquía. A esto se lo conoce como **sangrado** (o **indentación**, del inglés *indentation*).

```
while numero < 10:
    numero = numero + 1
    print numero
print "Terminé"
```

[Marcar:
· el encabezado (`while numero < 10:`),
· la condición (`numero < 10`),
· el cuerpo (las dos líneas que siguen),
· el indentado o sangrado (el espacio en blanco al principio de las líneas del cuerpo),
· "Instrucción por fuera de la repetición" para `print "Terminé"`]

Podemos decir, entonces, que una repetición condicional tiene dos partes. La primera línea es el **encabezado**, donde se indica que es una repetición condicional (mediante la palabra clave *while*), se escribe la condición y se termina con `:`. La segunda parte es el **cuerpo** de la repetición, formado por las instrucciones con sangrado, que serán las que se repetirán durante la ejecución.

IDE TIP
El entorno también nos ayuda a colocar las indentaciones correctas

```
while 1 == 2:
    |
```

```
8   while 1 == 2:
9   print "hola"
    Indent expected
```

En el primer caso, vemos que automáticamente coloca el cursor para seguir escribiendo dentro del cuerpo del ciclo (es decir, con una indentación) mientras que en el segundo detecta el error y lo marca.
Sin embargo, podemos manejar la indentación a mano (podemos agregar cuatro espacios o utilizar la tecla *tab*) y debemos ser conscientes de ella, dado que a veces la sintaxis de un programa puede ser correcta pero no refleja lo que queremos expresar.

```
while hayQueSaludar:
    print "Hola"
print "Chau"
```

```
while hayQueSaludar:
    print "Hola"
    print "Chau"
```

Ambos fragmentos son sintácticamente correctos, pero no se comportan de la misma manera: mientras que en el segundo `print "Chau"` forma parte de la repetición (pues aparece indentado), en el primero se ejecuta una sola vez al final.

El símbolo igual (=) se utiliza para almacenar un valor dentro de una variable; así, por ejemplo, la instrucción `mi_variable = 15` hace que el valor `15` se almacene en una variable llamada `mi_variable`. En cambio, para verificar si dos elementos son iguales (lo cual es útil para expresar una condición), se utilizan dos símbolos igual (`==`), como en la línea 13.

Para hacerle una pregunta al usuario, usamos el método `raw_input`, similar a `preguntar al usuario` en Alice. La diferencia es que no podemos preguntar por sí o no, sino que la respuesta del usuario siempre viene en forma de texto; por eso, lo que hacemos en las líneas 10 y 11 es guardar la respuesta en una variable llamada `respuesta` y luego verificar si la misma fue `"sí"`.[6]

¿Por qué el texto va entre comillas?

¿Cuál es la diferencia entre `print hola` y `print "hola"`? ¿Y entre `print mi_variable` y `print "mi_variable"`?
Recordemos que estamos trabajando con un lenguaje donde todos los elementos que forman el programa se escriben como texto. Por definición, cuando no usamos comillas, nos estamos refiriendo a elementos del programa (entonces `print hola` y `print mi_variable` imprimen por pantalla el valor de las variables `hola` y `mi_variable`, respectivamente). Por el contrario, cuando queremos incorporar un texto en particular para que el programa trabaje con él, debemos encerrarlo entre comillas (entonces, print "hola" y print "mi_variable" escriben directamente *hola* y *mi_variable* en la consola). Esta distinción es fundamental y puede ser muy confusa, por eso el entorno nos ayuda coloreando los textos entrecomillados.

La alternativa condicional comienza con el encabezado `if condición:` (observamos la palabra clave *if* y los dos puntos para cerrarlo) y cada rama se indica con sangrado. La primera corresponde a cuando la condición es verdadera, mientras que la rama para cuando la condición es falsa se indica con el encabezado `else:` y más instrucciones con sangrado.

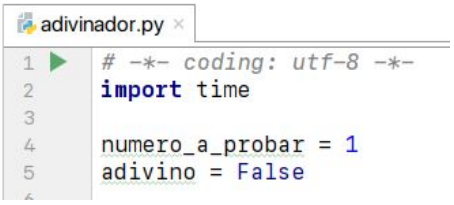
```
if respuesta == 2:
   print "Felicitaciones, ganaste"
   puntaje = 100
else:
   print "Volvé a intentar"
   puntaje = -1
```

Marcar, al igual que en el while:
· el encabezado (`if laRespuestaEsCorrecta:`)
· la condición (respuesta == 2)
· el sangrado para cada rama
· la rama verdadera (print felicitaciones y puntaje = 100)
· la palabra clave else y : para indicar la rama falsa
· la rama falsa (print volvé a intentar y puntaje = -1)

[6] Debemos prestar atención con este punto, ya que la comparación entre fragmentos de texto en Python es muy estricta y esto puede causar problemas. En la solución dada como muestra, el programa espera que el usuario responda exactamente `"sí"`, por lo que si la respuesta es `"si"` (sin tilde) o `"SI"` (en mayúsculas), el programa hará de cuenta que dijimos que no.

Para terminar con el programa, cabe señalar dos últimas diferencias. Una de ellas es que la manera de sumar uno a una variable en Python es diferente a como se hace en Alice (ver la línea 16 del programa). Por otra parte, el símbolo de comilla invertida (`) que se usa en las líneas 11 y 14; usar este símbolo en Python para encerrar una expresión equivale a usar, en Alice, la función `[algo] como secuencia de caracteres`; el símbolo más (+), por su parte, colocado entre dos secuencias de caracteres, es el equivalente de la función de Alice `[a] unido con [b]`.

Para ejecutar el programa, hacemos clic en el pequeño símbolo verde de “play” ubicado junto a la primera línea. Esto abrirá una consola de Python en la que podremos ir respondiendo las preguntas del programa y viendo los resultados.

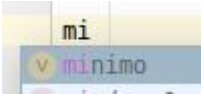


Luego de haber programado el adivinador entre todos, explicando cada una de las líneas del programa, y de haber experimentado su ejecución, pedimos a los estudiantes que copien el código en sus propias computadoras, abran el intérprete y lo ejecuten por sus propios medios. Estamos atentos para ayudarlos ante cualquier dificultad que pueda surgir.Cuando todos hayan podido hacerlo, les pedimos que modifiquen el programa de manera que el usuario pueda decidir entre qué par de números va a elegir el número que el programa debe adivinar. El objetivo es que comiencen a familiarizarse con el lenguaje; les damos el tiempo suficiente para que puedan experimentar. El nuevo desafío requerirá realizar dos preguntas al usuario, guardar las respuestas en dos variables y utilizarlas como las cotas del rango de números en el cual buscar. A continuación se muestra una posible solución.

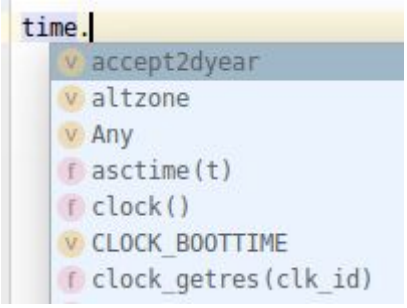
```
1  # -*- coding: utf-8 -*-
2  import time
3
4  minimo = int(raw_input("¿Desde qué número será la búsqueda?"))
5  maximo = int(raw_input("¿Hasta qué número será la búsqueda?"))
6  adivino = False
7
8  print "Pensá un número entre el " + `minimo` + " y el " + `maximo`
9  time.sleep(2)
10
11 numero_a_probar = minimo
12
13 while not adivino:
14    respuesta = raw_input("¿El número que pensaste es el " + `numero_a_probar` + "?")
15    if respuesta == "sí":
16        adivino = True
17        print "¡Adiviné! El número que elegiste es el " + `numero_a_probar - 1`
18    else:
19        numero_a_probar = numero_a_probar + 1
```

Variante del programa adivinador, permitiendo al usuario elegir el rango de números.

IDE TIP: Autocompletar
En general, los editores de texto orientados a programación (como el que incluye nuestro entorno de referencia para escribir los programas), cuentan con una función de autocompletar, es decir, a medida que escribimos, nos sugiere palabras que son correctas para utilizar en ese lugar del programa y que comienzan con (o contienen) lo que ya escribimos.
Además de ahorrarnos esfuerzo de tecleo, es particularmente útil para evitar errores de escritura. Por ejemplo, si en vez de `minimo` escribimos `minmo`, será un error del programa pues la variable `minmo` no está definida (y probablemente nos cueste detectar que se debe a que nos faltó escribir una i); en cambio, si escribimos `mi` y aceptamos la sugerencia de `minimo`, no estaremos cometiendo este error.

	A medida que escribimos, el editor nos sugiere palabras del programa que podríamos estar queriendo escribir.

Además, también es una manera rápida de conocer qué cosas están definidas y disponibles para utilizar en el programa. Por ejemplo, si en el adivinador escribimos `time.`, el editor nos enumera todas las funciones que contiene la biblioteca time, pues son las palabras que podríamos escribir a continuación.



La instrucción `int` que encierra a los primeros dos `raw_input` se utiliza para que la computadora interprete que los valores ingresados por el usuario son numéricos. Si no, la computadora los interpretará como texto.

Cierre

Cerramos con una reflexión que servirá como puntapié inicial para la próxima actividad, donde continuaremos trabajando con el lenguaje Python para tratar de mejorar la estrategia usada por el programa para adivinar el número, de forma que pueda hacerlo planteando menos preguntas al usuario. Debatimos en grupo algunas preguntas para motivar la noción de **eficiencia**, pensando cuántas preguntas necesita hacer el programa a sus usuarios, especialmente en el **peor caso** posible.

- ¿Cuántas preguntas tiene que hacer el programa para adivinar si el usuario elige el número 10?

- ¿Se les ocurre una alternativa? Por ejemplo, ¿será mejor cambiar el orden de las preguntas para ir de mayor a menor, empezando por el 10 y terminando por el 1?
- ¿Qué pasa si cambiamos por esta última estrategia y el usuario elige el 1?
- En el peor caso, con cualquiera de las dos estrategias, el programa debe hacer diez preguntas para adivinar el número que eligió el usuario. ¿Existirá una estrategia que realice menos pasos, sin importar qué número elija el usuario?

A la última pregunta, no le daremos por ahora una respuesta por sí o por no, sino que la dejaremos planteada a modo de inquietud, para retomarla en la próxima actividad al abordar el algoritmo de búsqueda binaria.

Actividad 3 | SD1

¿Podremos buscar más rápido?

Objetivos

- Conocer un algoritmo eficiente de búsqueda: la búsqueda binaria.
- Implementar en Python una búsqueda binaria.
- Adquirir una noción elemental de **eficiencia algorítmica**.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras
- Python (preferentemente PyCharm Edu).

Desarrollo

En la actividad anterior, los estudiantes realizaron un programa sencillo que les permitía encontrar un número (el que había pensado el usuario del programa) dentro de una secuencia de números (los números del 1 al 10).

Comenzamos la actividad dedicando unos minutos a preguntar a los estudiantes cómo creen que un sitio como YouTube o Spotify hace para encontrar tan rápidamente la película o video que se le pide. Para tener una idea de la dificultad de la tarea, se recuperará la discusión realizada en la actividad anterior, acerca de cuántos números debía evaluar el programa hasta acertar el elegido por el usuario, en el peor de los casos. Una de las conclusiones era que, independientemente de si se recorren los números “hacia arriba” o “hacia abajo”, en el peor caso el usuario podría elegir el extremo opuesto y, por ende, se debería preguntar por todos los números del rango hasta adivinar.

Una de las preguntas que quedó abierta es si existía alguna estrategia que permitiese encontrar el número elegido por el jugador sin tener que barrer todo el conjunto de posibilidades. En el caso de que el espacio de búsqueda sean sólo 10 números no parece un problema serio tener que preguntar por todos. Pero, ¿si fueran 100?, ¿o 1000? ¿Y si fueran 100 millones? Eso nos llevaría a tener que revisar, cien, mil y cien millones de posibilidades respectivamente.

Preguntamos, entonces, a los estudiantes si se les ocurre algún contexto en donde haya que buscar entre tantas cosas. ¿Alguna vez tuvieron que buscar entre miles o millones de elementos? Algunas posibles respuestas sobre búsquedas en soportes físicos pueden ser buscar una palabra en un diccionario de papel, un nombre en una guía telefónica o en una agenda. También pueden mencionar realizar una búsqueda en un buscador web, un video, un usuario en una red social, etc. Si no surgieran entre los ejemplos mencionados, podemos agregar que en estos últimos casos, las búsquedas suceden entre millones, incluso, miles

de millones o billones, de elementos. Por ejemplo, un buscador web debe buscar entre más de mil millones de sitios web para determinar cuál es la lista de los que se corresponden con las palabras deseadas.

- ¿Cómo hace para hacer la búsqueda en tan pocos milisegundos?
- ¿Cada vez buscará uno por uno entre todos los sitios?
- ¿Existirá una manera más **eficiente**?

En esta clase presentaremos una estrategia de búsqueda más eficiente: la búsqueda binaria. Para motivar su abordaje, proponemos buscar un número en una secuencia de números ordenados. Cada número estará escrito en un papelito y tapado por un pequeño vaso plástico;[7] además, los números estarán ordenados de menor a mayor.

Secuencia de números ordenados de menor a mayor y ocultos debajo de un vaso.

Se les pedirá a las/los estudiantes que determinen si el número 8 está o no en la secuencia de números pero asegurando que no van a revisar debajo de todos los vasitos.

- ¿Qué opinan de empezar por el vaso del medio?
- Si justo es el 8 listo. ¿Y si no? ¿Qué se puede hacer?

Para comenzar con la demostración daremos vuelta el vaso del medio, descubriendo el número 5, el cual no es el número buscado. Realizamos nuevamente la segunda pregunta, recordando que la secuencia de números está ordenada de menor a mayor, por más que no se sepa qué números hay debajo de cada vaso. Por lo tanto, como el 8 es mayor que el 5, seguramente el 8 no podrá estar debajo de ninguno de los 3 vasos a la izquierda del 5 ya que estos deberían ser menores que 5. Por lo tanto, si el 8 estuviera presente en la secuencia, lo estaría debajo de alguno de los 3 vasos a la derecha del 5. Habiendo llegado a esta conclusión se pueden quitar de la mesa, o separar hacia un costado, los 3 vasos que cubren al 1, al 2, al 3 y al 5, obteniendo así una nueva secuencia de 3 números: 8-13-21.

- ¿Cómo podemos seguir?
- ¿Se podrá repetir el proceso en esta nueva secuencia de 3 elementos?

Ahora serán los estudiantes quienes deban continuar el proceso levantando el vaso que cubre al 13, verificar que no es el 8 y descartar los vasos que están a su derecha, quedando

[7] Como alternativa, también podemos simplemente poner los papeles dados vuelta. El objetivo de hacer esto es simular lo que hace una computadora: revisar de a un número por vez. A diferencia de los seres humanos, que podemos ver varios números a la vez y decidir en un golpe de ojo si el número está o no, la computadora no “ve” sino que para cada posición se fija cuál es el valor que allí está.

tan solo el vaso que cubre al 8. Para terminar, levantarán este vaso concluyendo que el 8 sí está en la secuencia de números presentada.

- Si en vez del 8 hubiera habido un 7, ¿hubiera cambiado algo? ¿Qué?
- ¿Tuvimos que revisar todos los números de la secuencia?

Si en lugar del 8 hubiera estado el 7, el procedimiento hubiera sido análogo. La única diferencia hubiera sido que la respuesta sobre si el 8 estaba o no habría sido negativa.

- ¿Se podría hacer algo similar con el problema de adivinar el número elegido por el usuario?
- ¿Si cambiamos la pregunta que hace el programa? ¿Cuál podría ser?

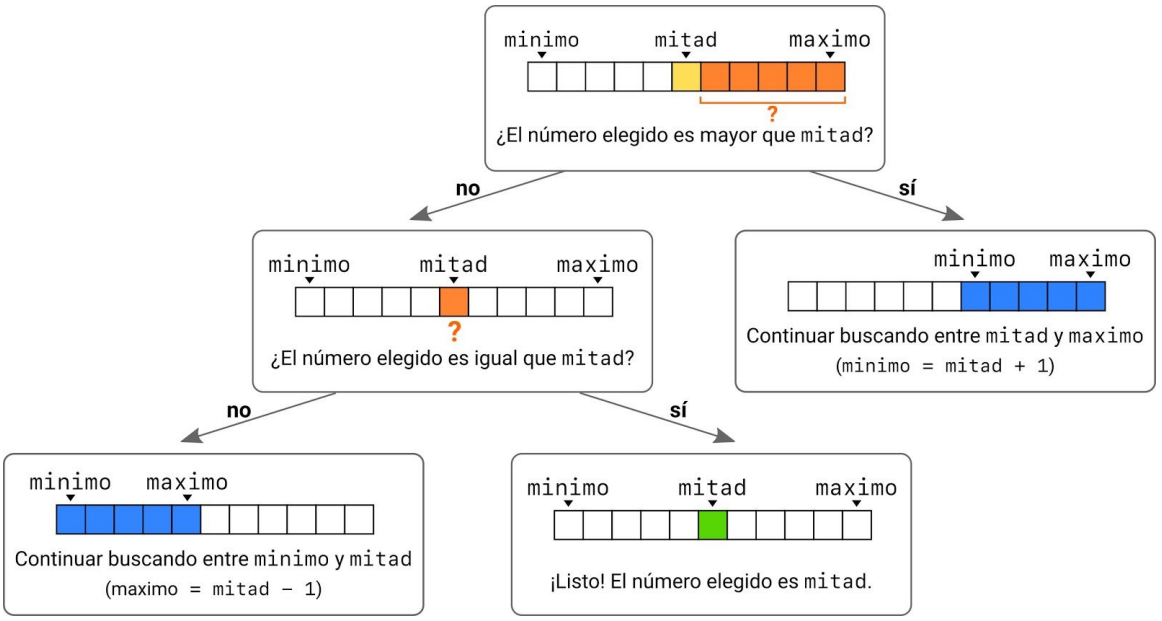
En este punto, iremos guiando la discusión para llegar a que, de modo similar a como hicieron recién, se podría preguntar si el número elegido por la computadora es mayor (o menor, es análogo) al número que eligió el usuario. De este modo, se descartan todos los números mayores -en caso negativo- o todos los números menores o iguales -en caso afirmativo.

- ¿Qué número elegirían para preguntar como primera opción entre el 1 y el 100?
- ¿Y por cuál seguirían preguntando si el usuario responde que el número que eligió es mayor?

Conviene anotar esta idea en el pizarrón y trabajarla con un ejemplo en donde seamos nosotros quienes pensemos un número y lo escribamos en un papel que nadie pueda ver hasta que no hayan adivinado.[8]

El algoritmo que acabamos de presentar, que sirve para buscar un elemento dentro de una secuencia de elementos, recibe el nombre de **búsqueda binaria**, nombre que proviene de que en cada paso, la secuencia se divide en dos partes y solo se continúa buscando en una de ellas. A continuación se muestra una representación esquemática del algoritmo de búsqueda binaria.

[8] En http://www.csfieldguide.org.nz/en/teacher/chapters/algorithms.html#searching, una guía de trabajo sobre temas de Cs. de la Computación elaborada por el equipo que diseñó las actividades CSUnplugged, se puede jugar a una versión interactiva de este juego (con los números ordenados y sin ordenar). La explicación y la consigna están en inglés, pero son muy similares a lo que desarrollamos hasta ahora.



Representación esquemática del algoritmo de búsqueda binaria

A continuación, buscaremos que los estudiantes puedan ver en funcionamiento el algoritmo de búsqueda binaria programándolo en Python. Como programarlo desde cero puede resultar difícil, propondremos que completen los espacios con "..." del siguiente programa incompleto.

```
# -*- coding: utf-8 -*-
import time

minimo = 1
maximo = 1000
adivino = False

print "Pensá un número entre " + `minimo` + " y " + `maximo`
time.sleep(2)

while minimo <= maximo and not adivino:
  medio = (...) / 2

  respuesta = raw_input("¿El número que pensaste es " + `medio` + "?")
  if respuesta == "sí":
     adivino = ...
     print "¡Adiviné! El número que pensaste es " + `...`
  else:
     respuesta = raw_input("¿El número que pensaste es mayor que " + `medio` + "?")
     if respuesta == "sí":
        minimo = ... + 1
     else:
        maximo = ... - 1
```

Programa incompleto de búsqueda binaria

La siguiente es una solución correcta para el desafío propuesto:

```
# -*- coding: utf-8 -*-
import time

minimo = 1
maximo = 10
adivino = False

print "Pensá un número entre " + `minimo` + " y " + `maximo`
time.sleep(2)

while minimo <= maximo and not adivino:
  medio = (minimo + maximo) / 2

  respuesta = raw_input("¿El número que pensaste es " + `medio` + "?")
  if respuesta == "sí":
     adivino = True
     print "¡Adiviné! El número que pensaste es " + `medio` + "."
  else:
     respuesta = raw_input("¿El número que pensaste es mayor que " + `medio` + "?")
     if respuesta == "sí":
        minimo = medio + 1
     else:
        maximo = medio - 1
```

Programa de búsqueda binaria

A continuación, les pedimos que modifiquen el programa para que pueda decir *cuántas preguntas* en total se le hicieron al usuario hasta haber adivinado el número que eligió. Esto pueden hacerlo agregando una variable que sea inicializada con el valor 0 y cuyo valor se incremente cada vez que se hace una pregunta.

Indicamos que experimenten con distintos números para tomar nota de qué valores da. Si se toma el rango 1-10, en ningún caso la cantidad de preguntas será 10: de hecho, la máxima cantidad de preguntas posibles en este caso es 7.

¿Qué ocurre con la cantidad de preguntas realizadas si el rango es de 1 a 100? ¿Y de 1 a 1000? Incluso para rangos altos como 1-1000, podrán apreciar que la cantidad de preguntas realizadas es siempre mucho menor a 1000, valor que sí se obtenía en el peor caso de la búsqueda lineal.

Contamos que, efectivamente, cuando se quiere buscar un elemento en una secuencia ordenada, como los números del 1 al 1000, esta estrategia de descartar la mitad de los elementos en cada paso es significativamente más eficiente que ir preguntando uno por uno. Una manera de verlo es que, haciendo tan solo una pregunta más, duplicamos la cantidad de números entre los que podemos buscar. La siguiente tabla puede servir para ejemplificar la diferencia entre la estrategia lineal y la estrategia binaria en el peor caso.

Rango de números	Cantidad de preguntas **Búsqueda lineal**	Cantidad de preguntas **Búsqueda binaria**

entre 1 y 10	10	7
entre 1 y 100	100	13
entre 1 y 1.000	1.000	19
entre 1 y 1.000.000	1.000.000	39
entre 1 y 1.000.000.000	1.000.000.000	59

Comparativa entre la cantidad de preguntas que se le hacen al usuario en el peor caso para los algoritmos de búsqueda lineal y búsqueda binaria

Cierre

Para concluir, mostraremos un caso concreto de aplicación del algoritmo de búsqueda binaria en un dominio que las/los estudiantes suelen utilizar con frecuencia: Google o, más en general, los buscadores web. Para ello, proponemos a los estudiantes leer el artículo "Destripando Google"[9] y responder las siguientes preguntas:

- ¿Cómo relacionan la forma en que un buscador busca en una serie de palabras con el algoritmo de búsqueda binaria?
- ¿Qué característica tienen todas las palabras que están a la izquierda de un nodo[10] particular en un árbol AVL? ¿Y las que están a la derecha?
- ¿Por qué es importante esta característica?

[9] Schapachnik, F. (2015). *Destripando Google*. https://elgatoylacaja.com.ar/destripando-google/
[10] Un nodo es un elemento de un árbol. En el ejemplo del artículo, cada palabra es un nodo.

Actividad 4 | SD1

Secuencias en Python

Objetivos

- Presentar la estructura de Python para representar secuencias: las listas.
- Trabajar con recorridos sobre secuencias (ciclos for).

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras
- Python (preferentemente PyCharm Edu)
- Proyector (opcional).

Desarrollo

Hasta ahora, en Python, hemos venido trabajando con elementos de manera aislada, ya sea que fueran números, fragmentos de texto o valores de verdad. Sin embargo, Python también nos brinda la posibilidad de agrupar múltiples elementos en una **secuencia** o lista y trabajar con ellos de manera conjunta. Esto es especialmente útil porque, como veremos en las actividades siguientes, son muchos los contextos en los que se trabaja con información organizada en forma de secuencia, como al manipular textos, imágenes, videos o sonidos.

Comenzaremos esta actividad invitando a los estudiantes a experimentar con la consola de Python, para familiarizarse con algunas operaciones básicas de secuencias. Si tenemos la posibilidad de usar un proyector, podemos acompañarlos escribiendo los comandos, aunque intentando asegurarnos de que ellos también los ejecuten en sus computadoras.

En Python, una secuencia de elementos se escribe listando cada uno de ellos separados por comas (`,`), y encerrando la lista completa entre corchetes (`[` y `]`). Lo primero que haremos será crear una secuencia de números y almacenarla en una variable. Por ejemplo:

```
>>> mi_secuencia = [6, 2, 5, 9, 6, 5, 1]
```

También podemos pedirle al intérprete que muestre por pantalla la secuencia que acabamos de crear:

```
>>> print mi_secuencia
[6, 2, 5, 9, 6, 5, 1]
```

La herramienta fundamental para trabajar con secuencias es el **recorrido** o `for`: ejecutar ciertas instrucciones para **cada uno** de los elementos de la secuencia, recorriéndolos en orden. Al igual que las repeticiones y los condicionales, los recorridos tienen un encabezado

y un cuerpo. En el encabezado indicaremos qué secuencia queremos recorrer y también un nombre que vamos a usar luego para referirnos al elemento de la secuencia que estemos mirando en cada paso. Por ejemplo:

```
>>> for elemento in mi_secuencia:
```

que significa “para cada *elemento* en *mi secuencia*:”. Tras escribir esto y pulsar la tecla Enter, el intérprete nos dará la posibilidad de agregar más líneas de código (debemos tener en cuenta el sangrado: algunos intérpretes lo colocan automáticamente y otros esperan que lo agreguemos nosotros). Podemos escribir:

```
...     print "Este es el número " + `elemento`
```

Luego deberemos pulsar Enter dos veces para indicarle al intérprete que ya terminamos de escribir el cuerpo del ciclo y que debe ejecutarlo. Obtendremos la siguiente respuesta:

```
Este es el número 6
Este es el número 2
Este es el número 5
Este es el número 9
Este es el número 6
Este es el número 5
Este es el número 1
```

Probamos algunas veces más, poniendo diferentes instrucciones dentro del cuerpo del recorrido, incluso poniendo más de una única instrucción. Podemos pedir sugerencias a los estudiantes e invitarlos a que prueben sus propias ideas en sus computadoras. Podemos sugerirles que experimenten con secuencias de textos, como por ejemplo:

`los_nombres = ["Ariel", "Betty", "Charly", "Diana"]`	

`for nombre in los_nombres:`
`  print "Hola " + nombre`
	`Hola Ariel`
	`Hola Betty`
	`Hola Charly`
	`Hola Diana`

`for nombre in los_nombres:`
`  print "Llegó una nueva persona"`
`  print "Hola " + nombre`
`  print "Chau " + nombre`
	`Llegó una nueva persona`
	`Hola Ariel`
	`Chau Ariel`
	`Llegó una nueva persona`
	`Hola Betty`
	`Chau Betty`
	`Llegó una nueva persona`
	`Hola Charly`
	`Chau Charly`
	`Llegó una nueva persona`
	`Hola Diana`
	`Chau Diana`

Observamos dos recorridos distintos sobre la lista de nombres: en el primero, solo imprimimos Hola y el nombre a continuación, mientras que en el segundo vemos que podemos agregar más de una instrucción en el cuerpo, incluso algunas que no hagan	

referencia al elemento de la secuencia que estamos considerando en ese paso (`print "Llegó una nueva persona"`)

Otra forma de acceder a los elementos de una secuencia es haciendo referencia explícita a su posición. Para esto, utilizamos el operador [_] y un número o **índice**. Es fundamental observar que el primer elemento de la secuencia tiene índice 0 (en vez de 1) y, por lo tanto, los índices válidos para una secuencia de n elementos son 0, 1, 2, …, n-1. Utilizar un índice inválido (es decir, que cae "por fuera" de la lista) produce un error de ejecución.

`los_nombres = ["Ariel", "Betty", "Charly", "Diana"]` `print "Mi mejor amiga se llama " + los_nombres[1]`	`Mi mejor amiga se llama Betty`
Un programa y su resultado. Vemos que `los_nombres[1]` representa al segundo elemento de la secuencia ("Betty"), pues, recordamos, los índices comienzan desde cero.	

Mostramos esto a los estudiantes y les proponemos que lo experimenten en sus consolas, a la vez que podemos sugerir algunos ejemplos similares y pedirles que predigan el resultado.

A continuación, la idea es que los estudiantes resuelvan, programando en Python, algunos problemas con secuencias. Esto les permitirá entrenar sus habilidades con el lenguaje mientras se familiarizan con esta nueva forma de organizar la información, que luego usaremos extensivamente en actividades y proyectos para representar imágenes y sonidos. Comenzaremos con un problema algorítmico para resolver *unplugged* y después implementaremos una solución en Python.

Consigna: pensar un algoritmo para resolver el siguiente problema.
Salimos a la calle y le pedimos, a cada persona con la que nos cruzamos, que nos diga una palabra al azar. Al final del día, queremos saber cuál fue la palabra más larga que nos dijeron.
Aclaración: debido a la cantidad de palabras que esperamos escuchar, no podemos anotarlas todas, aunque sí podemos llevar un registro limitado (por ejemplo, una pizarra pequeña, donde podemos escribir y borrar).

Damos tiempo para que los grupos piensen y pongan a prueba sus propuestas antes de hacer la puesta en común. A continuación presentamos una estrategia válida que no requiere más espacio que para una sola palabra:

Encontrar la palabra más larga:
- *Pedir una palabra y anotarla.*
  Para cada nueva palabra escuchada:
    Si la nueva palabra es más larga que la anotada:
      Borrar la palabra anotada y anotar la nueva palabra.
  Responder que la palabra maś larga escuchada es la palabra anotada.

Esta estrategia, si bien puede motivarse desde la intuición, tiene un detalle que hay que precisar: hace falta designar "a mano" una primera palabra. Una opción es pedir una

palabra especialmente para esto, como hicimos en la solución expuesta. Otra (y tal vez más abstracta) es utilizar una palabra que sea más corta que cualquier otra que vayamos a escuchar (y, por lo tanto, será borrada en el primer paso de la repetición). Para esto, debemos elegir la palabra sin caracteres (que escribimos **“”**), cuya longitud es 0 y, por lo tanto, menor que la de cualquier otra palabra.

Cuando cada grupo esté convencido de la estrategia presentada, les pedimos que escriban un programa para implementar el algoritmo que acabamos de discutir.

Consigna: escribir un programa para encontrar la palabra más larga de una secuencia de palabras. *Atención: crear una variable que contenga la lista de palabras de entre las cuales queremos buscar.* *Pista: experimentar con la función* `len`.
Atención: ¿Qué pasa si hacemos `len("año")`*? Es probable que el resultado no sea 3, debido a que la codificación de los caracteres ajenos al inglés (como la ñ, las vocales acentuadas, los signos de apertura, etc.), internamente, es más larga que la de los caracteres “comunes”. Para evitar este problema, debemos aclarar que la secuencia debe interpretarse como una secuencia de caracteres con una codificación en particular (Unicode). Esto lo hacemos anteponiendo una u a la cadena de texto, es decir, en el ejemplo anterior, deberíamos ejecutar* `len(u"año")`*. En este primer ejercicio podemos evitar estos caracteres si nos parece que se prestará a confusión.*

Los dejamos trabajar, insistiendo con que tomen de base la estrategia que produjimos. Como siempre, estamos atentos a las necesidades que surjan, a las que responderemos con preguntas y pistas para orientarlos, pensando en el significado en castellano de lo que escribimos en la estrategia y recordando qué herramienta de programación que ya vimos puede utilizarse para resolverlo. A su vez, motivamos a que elijan siempre nombres que expresen el significado de lo que están refiriendo.

A continuación presentamos una solución posible y las sugerencias para motivar algunos de sus elementos que creemos que serán los que revistan mayor dificultad.

```
palabrasEscuchadas = ["Buscamos", "una", "palabra", "extremadamente", "larga"]

masLargaHastaAhora = palabrasEscuchadas[0]

for palabra in palabrasEscuchadas:
   if len(palabra)  > len(masLargaHastaAhora):
       masLargaHastaAhora = palabra

print "La palabra más larga que escuchamos fue: " + masLargaHastaAhora
```

En la estrategia	En el programa	
Pedir una nueva palabra	`palabrasEscuchadas[0]`	`[0]` nos devuelve el primer elemento de la secuencia.

Para cada palabra escuchada	`for palabra in palabrasEscuchadas`	Usamos un recorrido de secuencias para procesar, de a una, todas las palabras
Si la nueva palabra...	`if ...`	Usamos una alternativa condicional, porque no siempre tenemos que borrar y escribir la nueva palabra
la nueva palabra es más larga que la antoada	`len(palabra) > len(masLargaHastaAhora)`	Utilizamos la función `len` y la comparación por `>` para decidir si la nueva palabra es más larga.
palabra anotada	`una variable llamada masLargaHastaAhora`	Cuando escribimos la estrategia, observamos que era necesario *recordar* o registrar algún valor. Vimos que, justamente, las variables cumplen esta función.

> *¿Qué hay que cambiar del programa anterior para que, a partir de una secuencia de números, encuentre el más grande de todos? ¿Es muy distinta la estrategia? ¿Y si tuvieran que buscarlo ustedes "a mano", cómo harían? ¿Cambiarían la estrategia?*

A partir de la última pregunta, observamos que al enfrentarnos a problemas análogos en la vida cotidiana, no solemos pensar de forma algorítmica. Si nos encontramos con un lista relativamente breve de números, nos podemos dar cuenta a simple vista cuál es el mayor de todos. No obstante, si los números son muchos, ya no resulta tan sencillo, y seguramente nos veamos obligados a ir mirándolos uno a uno. Al resolver el problema en la computadora, aunque los números sean pocos, no tendremos opción más que ir **recorriéndolos individualmente**. Para acostumbrarnos a pensar soluciones teniendo en cuenta esta restricción, agregamos la limitación de que la lista de palabras era desconocida a priori (o lo que es similar, tan grande que no podíamos leerla de una vez).

A continuación vemos un programa en Python que utiliza esta misma estrategia para encontrar el máximo en una secuencia de números. La variable `maximo` sirve para almacenar el elemento que "apartamos" de la lista, es decir, se corresponde con `laMasLargaHastaAhora`. Vamos recorriendo uno a uno los elementos de la secuencia, y si el `numero` que estamos viendo es mayor que el `maximo` que habíamos guardado (antes: si la nueva palabra es más larga que la palabra anotada), pasamos a almacenar este nuevo valor en la variable. Finalmente, imprimimos por pantalla el resultado.

```
1 # -*- coding: utf-8 -*-
```

```
2
3  mi_secuencia = [15,7,3,9,16,6,22,4]
4
5  maximo = mi_secuencia[0]
6
7  for numero in mi_secuencia:
8     if (numero > maximo):
9         maximo = numero
10
11 print "El máximo es " + `maximo`
```

Aquí es importante distinguir el rol que cumple la secuencia almacenada en la variable al comienzo del programa. Podemos notar que no es parte de la solución del problema: es lo que se conoce como sus **datos de entrada**. La mayoría de los programas que usamos a diario necesita datos de entrada para funcionar; por ejemplo, cuando escribimos palabras en un buscador o escribimos un mensaje en una aplicación, les estamos proporcionando datos de entrada. Sin embargo, los datos de entrada no suelen estar escritos dentro de los programas (como en nuestro caso), sino que llegan “desde afuera” cuando los utilizamos.

Sería bueno quitar los datos de entrada de nuestro programa, de manera de poder utilizarlo para cualquier lista que se nos ocurra. Una manera en que podemos hacerlo es convirtiendo nuestro programa en un método, análogamente a como hacíamos en Alice. En Python, un método (que, en la bibliografía oficial se llama *función*) se escribe con un encabezado como el que sigue:

```
def nombre_del_metodo():
```

Seguido por las instrucciones del método, escritas con sangrado.

Comenzaremos por mover a un método las instrucciones del programa, dejando fuera los datos de entrada. Así obtendremos:

```
1  # -*- coding: utf-8 -*-
2
3  mi_secuencia = [1,2,3,4,5,6,22,4]
4
5  def buscar_maximo():
6     maximo = mi_secuencia[0]
7
8     for numero in mi_secuencia:
9         if (numero > maximo):
10            maximo = numero
11
12    print "El máximo es " + `maximo`
```

Si eliminamos sin más los datos de entrada, es decir, la variable `mi_secuencia`, tendremos un problema. El programa dejará de funcionar, porque las instrucciones que hacen referencia a esa variable pasarán a ser incorrectas. La manera de resolverlo es reemplazándola por un un **parámetro** del procedimiento `buscar_maximo`. En Python, esto

se hace agregándolo entre los paréntesis del encabezado del procedimiento. Por ejemplo, si al parámetro lo llamamos `secuencia`, nos queda:

```
# -*- coding: utf-8 -*-

def buscar_maximo(secuencia):
   maximo = secuencia[0]

   for numero in secuencia:
       if (numero > maximo):
           maximo = numero

   print "El máximo es " + `maximo`
```

Observamos que, ahora, el programa (el cuerpo del método) está escrito en base a `secuencia`, es decir, una secuencia genérica que en principio desconocemos y no a `mi_secuencia` que era una secuencia en particular. Luego, cuando lo queramos usar, deberemos proveerle esta información, es decir, la secuencia sobre la que queremos que trabaje. Esto se consigue colocando este valor en el lugar donde aparece el nombre del parámetro en el encabezado, es decir, `buscar_maximo([1, 2, 55, 3, 4])` o, si está definida la variable, `buscar_maximo(mi_secuencia)`. Si bien en principio la utilidad de la parametrización puede no ser tan evidente, es fundamental en programación. Debemos observar que, ahora, contamos con un método que se puede usar en cualquier programa para calcular el máximo de cualquier secuencia, e incluso, podemos nombrarlo para que quede explícito qué hace.Para probar esta nueva versión del programa, podemos hacer clic derecho en el código y seleccionar "*Run File in Console*". Esto abrirá una **nueva consola interactiva** en la parte inferior de la pantalla en la cual previamente se ejecutó nuestro programa, lo cual quiere decir que el intérprete incorporó la definición de nuestro procedimiento. También, si ya tenemos una consola abierta (con otras definiciones) y queremos agregar la nueva (o, incluso, cambiar una definición anterior del mismo método por una corregida), podemos seleccionar las líneas de la definición y, haciendo clic derecho, elegir *"Execute selection in console"*. Esto ejecuta la selección en la consola que tenemos abierta y, por lo tanto, conserva todas las definiciones y recuerda las instrucciones que ejecutamos hasta el momento.

y podremos pedirle que lo ejecute con distintos valores de entrada. Por ejemplo, si escribimos

```
>>> buscar_maximo([99,4,345,67,12,298,23,80])
```

obtendremos el resultado:

```
El máximo es 345
```

IDE TIP:
Si queremos volver a ejecutar una instrucción en la consola, no hace falta que la escribamos de vuelta. Presionando la flecha hacia arriba, nos aparecen, una a una, las últimas instrucciones que ejecutamos. Más aún: si escribimos las primeras letras de la instrucción que buscamos y luego presionamos la flecha hacia arriba, nos lista las instrucciones que ejecutamos y que comienzan con las mismas letras que escribimos.

Consigna: ¿Cuál es la palabra más larga que aparece en El Quijote? (Observamos que podemos elegir otra obra literaria, o incluso, cualquier archivo de texto plano que contenga muchas palabras).
Sugerencia: parametrizar el programa que busca la palabra más larga.
Herramienta: contamos con la función `cargarListaDePalabras(nombreDelArchivo)` que, dado un archivo de texto, nos devuelve todas las palabras que aparecen en él.

Pregunta: ¿Y si quisiéramos calcular el mínimo? ¿Y la palabra más corta? ¿Y la que tiene más vocales? ¿Y la comida que más nos gusta?

Para calcular el mínimo, adaptamos el programa del máximo y reemplazamos la comparación por mayor (>) por comparar por menor (<). Para encontrar la palabra más corta, hacemos algo similar al comparar las longitudes. Para saber cuál es la que tiene maś vocales, dentro del ciclo, en vez de comparar la palabra nueva con la vieja por su longitud, deberíamos contar las vocales de cada una y quedarnos con la que tenga más.

Observamos que en todos estos casos la estrategia es muy similar.

Sin embargo, ¿para la comida favorita? En cada paso, deberíamos elegir entre un plato favorito y uno nuevo. ¿Pero esto siempre es posible? ¿Podemos comparar una torta de chocolate con un choripán? Si bien es muy subjetivo, nos interesa resaltar que la estrategia que vimos sirve solo cuando todos los elementos del universo con el que estamos trabajando son comparables entre sí por el criterio que nos interesa.

IDE TIP: Autocompletar (bis)
Cuando comenzamos a definir procedimientos con parámetros, la función de autocompletar nos ayuda todavía más. Por un lado, incluirá los procedimientos que ya definimos como sugerencias y, además, incluirá los parámetros. Esto es particularmente útil para saber a qué podemos hacer referencia y a qué no. Por ejemplo, si dentro del cuerpo del procedimiento `calcular_suma` escribimos `sec` nos sugerirá `secuencia`, mientras que si lo hacemos por fuera, no (pues los parámetros son información de entrada del procedimiento que los declara únicamente, no de todo el programa y, por lo tanto, solo están disponibles en sus definiciones). De la misma manera, si nos olvidamos de agregar un parámetro a la definición o de definir una variable (como en el ejemplo anterior, que primero teníamos la secuencia almacenada en una variable y después la eliminamos), estos no aparecerán en la lista de sugerencias, lo que debería advertirnos que algo falta en nuestro programa.

Cierre

En esta actividad comenzamos a observar la potencia de un lenguaje textual. Con una estructura muy simple, pudimos escribir programas para revisar cientos de miles de palabras. También, observamos la velocidad a la que trabajan las comptuadoras: este procesamiento fue casi instantáneo. Además, observamos cómo una misma estrategia o algoritmo puede utilizarse para resolver problemas que en apariencia son muy diferentes. Continuaremos por este camino en la próxima secuencia cuando utilicemos Python para hacer programas que manipulen imágenes y, en el próximo capítulo, sonidos.

Secuencia didáctica 2: Programamos con imágenes

Bajada

En esta secuencia retomaremos la representación digital de imágenes, tal como la vimos en el capítulo de representación de la información, para hacer programas que produzcan o modifiquen imágenes. Para esto, utilizaremos lo practicado sobre recorridos de secuencias, ya que pensaremos en las imágenes como una grilla o matriz de píxeles y a esta matriz como una secuencia de secuencias.

Nuevamente adquirimos un enfoque proyectual, por lo que proponemos consignas incrementales y personalizables para que los grupos experimenten conforme a sus inquietudes e intereses.

Objetivos

- Afianzar las nociones de representación de imágenes.
- Trabajar con estructuras bidimensionales (matrices).
- Presentar el concepto de función de un lenguaje de programación.
- Programar filtros de imágenes.

Actividad 1 | SD2

Programamos en blanco o negro

Objetivos

- Afianzar la noción de pixel
- Familiarizarse con el recorrido de estructuras bidimensionales, como la grilla que soporta una imagen

Modalidad de trabajo

Grupal

Materiales y recursos utilizados

- Computadoras con Python y nuestras bibliotecas instaladas

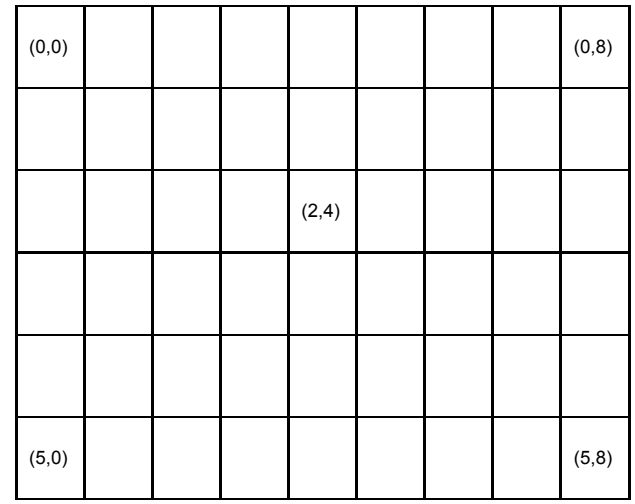
Bajada para el aula

En esta actividad realizaremos diversos algoritmos para dibujar imágenes simples pintando casilleros negros en un cuadriculado. Estos algoritmos utilizan, aunque de manera más compleja, las formas de recorrer secuencias trabajadas en las actividades anteriores, para poder recorrer una imagen como si se tratara de una secuencia de secuencias (o matriz) de píxeles.

Para estas actividades utilizaremos Python con una biblioteca especial para trabajar sobre imágenes. En el Anexo de instalación se encuentran las instrucciones necesarias para su instalación según el sistema operativo.
Podrán descargarse los archivos correspondientes a las actividades y sus soluciones en http://program.ar/descargas/TI4_clase8_codigo.zip

Comenzamos la actividad contando a los estudiantes que tendrán que realizar dibujos en blanco y negro pensando a la imagen como una cuadrícula blanca a la que se le colorearán determinados píxeles indicando su ubicación. Para eso, deberán importar la biblioteca `img`, que cuenta con métodos para trabajar sobre imágenes. Presentamos algunos que resultarán particularmente útiles:

- `img.nuevaByN(`*`filas, columnas`*`)` Este método creará una nueva imagen que deberá ser guardada en una variable. La imagen se creará con las dimensiones indicadas en los parámetros cantidad de filas y de columnas. Podemos preguntarnos qué rol cumplen *`filas`* y *`columnas`*, para recordar qué es un parámetro y cuál es el beneficio de crear métodos paramétricos.
- `img.mostrarByN(`*`imagen`*`)` Este método muestra en pantalla la imagen en blanco y negro que recibe como parámetro.
- `img.pintarPixel(`*`imagen, fila, columna`*`)` Este método pinta de negro un píxel determinado de una imagen ya creada, la cual recibirá como parámetro junto a la fila y la columna del píxel elegido. En este caso, debemos prestar atención a la importancia de utilizar parámetros válidos: por ejemplo, no se podrá pintar un píxel ubicado en una columna mayor a la cantidad de columnas totales.



Una imagen de 6 filas y 9 columnas. Observamos la posición de algunos de sus píxeles como (fila, columna).

Debemos recordar que, como estamos usando secuencias, las posiciones dentro de la cuadrícula comienzan en 0 tanto para filas como para columnas. Por ejemplo, el píxel ubicado en la esquina superior izquierda será el que tendrá valor 0 en la posición de las filas y valor 0 en la posición de las columnas. El píxel ubicado en el extremo superior derecho tendrá valor de fila 0 y valor de columna igual a la cantidad total de columnas - 1.

Para familiarizarse con el lenguaje y las instrucciones para pintar imágenes en blanco y negro, proponemos una primera consigna: pintar la fila del medio de una cuadrícula de 3 filas y 5 columnas.

Es posible que los grupos intenten resolver esta actividad pintando uno a uno los 5 píxeles de la fila central. Si bien esto no está mal, podemos proponerles que piensen si no sería conveniente utilizar alguna instrucción que permita hacer repeticiones. Por el momento no realizaremos un recorrido por filas y columnas, sino que se avanzará únicamente columna a columna (pues la fila que nos interesa es siempre la misma). Podemos pensar la siguiente estrategia para resolver el problema. Elegimos trabajar con una repetición condicional, dado que es la única forma que vimos en Python de generar repeticiones sin involucrar secuencias, es interesante preguntarse cuál es la condición de la repetición y qué debe agregarse al programa para que ésta funcione. Aparecen marcadas en negrita en la estategia.

Pintar fila central:

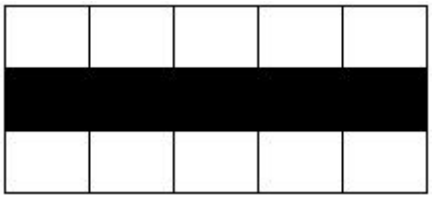
Crear una imagen en blanco de 3 filas x 5 columnas.
Definir la primera columna como la columna actual (columna número 0).
Mientras ***la columna actual sea menor que la cantidad total de columnas****:*
Pintar el píxel correspondiente a la columna actual de la segunda fila (fila número 1)
Aumentar en una unidad la columna actual
Al terminar de recorrer todas las columnas de la fila número 1, mostrar la imagen obtenida.

Observamos que para programar la idea propuesta deberemos utilizar la repetición condicional mediante la instrucción `while`. De todas maneras, presentaremos el código del programa en Python con el recorrido de la fila resuelto, para que solo tengan que completar el espacio punteado con la instrucción que se encarga de colorear el píxel deseado. El programa en Python es el siguiente:

```
import img
def pintarFilaCentral():
	imagen = img.nuevaByN(3, 5)
	columnaActual = 0
	while columnaActual < 5:
```

```
        .....................
        columnaActual = columnaActual + 1
    img.mostrarByN(imagen)
```

Para los grupos que necesiten ayuda, les indicamos que encuentren las correspondencias entre la estrategia que presentamos y el código del programa incompleto, para que observen que falta la instrucción que colorea el píxel. Para eso, deberán agregar `img.pintarPixel(imagen, 1, columnaActual)` con la imagen, la fila número 1 y la columna actual como parámetros. Observamos también que la repetición condicional terminará cuando la columna actual deje de ser menor a 5, es decir cuando sea igual a dicho valor y, por lo tanto, la última columna que se pintará será la número 4. Una vez más, recordamos que las columnas se cuentan desde 0 y por ende no existe la columna número 5, con lo cual, el comportamiento es correcto. Otra alternativa sería comparar por menor o igual; en este caso, deberíamos usar la cantidad de columnas menos uno. El resultado obtenido será similar al que vemos a continuación:



.Resultado del programa pintarFilaCentral: una imagen de 3x5 con la fila central pintada de negro

Para continuar con este ejercicio, reforzaremos la noción de parámetro que vimos en la secuencia anterior. Preguntamos qué pasaría si quisiéramos hacer el mismo dibujo, pero de largo 6, 10, 50, etc. para observar que deberíamos escribir un programa para cada uno. Sin embargo, el problema que estamos resolviendo es siempre muy similar, y de hecho, la estrategia no varía. Luego, podemos escribir un programa *más potente* (técnicamente: más general, en el sentido de que resuelve más problemas) que, si bien necesita que nosotros le proveamos el valor del largo, una vez que este valor está dado, es capaz de producir un tablero para cualquier largo. Éste es el significado que queremos resaltar: un parámetro es una entrada de información para el programa.

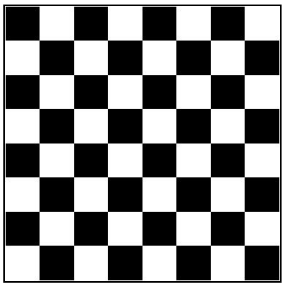
Para resolver este ejercicio, simplemente debemos observar que, si antes el largo era 5, ahora será el parámetro `largo`. Comparamos las definiciones de los dos métodos e invitamos a los grupos a que experimenten con ellos en la consola interactiva.

```
import img
def pintarFilaCentral():
    imagen = img.nuevaByN(3, 5)
    columnaActual = 0
    while columnaActual < 5:
        img.pintarPixel(imagen,          1,
        columnaActual)
        columnaActual = columnaActual + 1
    img.mostrarByN(imagen)
```

```
import img
def pintarFilaCentralParam(largo):
    imagen = img.nuevaByN(3, largo)
    columnaActual = 0
    while columnaActual < largo:
        img.pintarPixel(imagen,          1,
        columnaActual)
        columnaActual = columnaActual + 1
    img.mostrarByN(imagen)
```

A continuación, les pediremos a los estudiantes que realicen un programa para pintar un tablero de ajedrez. Es decir, una cuadrícula de 8 filas y 8 columnas con píxeles blancos y negros alternados. Insistimos en que, para comenzar, conviene pensar la idea principal en papel y luego utilizar la computadora. Con esto, buscamos que los grupos identifiquen que en este problema ya no será suficiente con recorrer únicamente de izquierda a derecha, sino que deberá recorrerse la cuadrícula entera variando tanto de fila como de columna.

Con esta actividad se buscará identificar el recorrido de una cuadrícula como un recorrido por filas dentro de las cuales se avanza columna a columna. Es posible realizar estos recorridos utilizando la instrucción `for` indicándole un rango o la instrucción `while` como en el ejemplo anterior indicando en este caso cuál es la condición que debe cumplirse para que el ciclo se detenga. Los resultados serán levemente diferentes.

Pintar tablero de ajedrez: *Crear una imagen en blanco de 8 filas x 8 columnas.* *Para cada fila de la imagen entre 0 y 8:* *Para cada columna de la fila actual entre 0 y 8:* *Si la fila es par y la columna es par pintar el píxel.* *Si la fila es impar y la columna es impar pintar el píxel.* *Al terminar de recorrer todas las filas y columnas mostrar la imagen obtenida.* Estrategia para resolver el problema.	 Resultado esperado del programa *pintarTableroDeAjedrez()*.

Como antes, presentamos un programa incompleto que los grupos deberán completar para conseguir el resultado esperado. Además, incluimos el método `img.esPar(`*`número`*`)` que tendrá como resultado un valor verdadero cuando el número sea par y falso cuando no lo sea. Vemos el programa incompleto a continuación:

```
import img
def pintarTableroDeAjedrez():
       imagen = img.nuevaByN(...)
       for fila in range(0, 8):
         for columna in range(0, 8):
              if (img.esPar(fila) and ....):
                     img.pintarPixel(imagen, fila, columna)
              elif (not(img.esPar(fila)) and not(img.esPar(columna))):
                     .........
       ..........
```

Esta solución incorpora algunas herramientas y usos novedosos, que nos interesa destacar para aprovechar en las actividades siguientes:

La instrucción `range(inicio, final)` se utiliza para indicar un rango de valores desde el primer parámetro hasta una unidad menos que el último. En concreto, `range(inicio, final)` produce la lista `[inicio, inicio+1, inicio+2, ..., final-1]`. Por ejemplo, `range(2, 5)` produce `[2, 3, 4]`. Por eso, al combinarlo con un recorrido `for` nos permite repetir el cuerpo del ciclo, una vez para cada valor del rango. En este caso, lo utilizamos para recorrer los valores de filas y de columnas.

Al anidar dos recorridos, podemos obtener todas las combinaciones posibles de los valores de los rangos especificados. En particular, en nuestro ejemplo, en cada repetición del primer `for` se fija el valor de la fila y, una vez que éste está elegido, se van sucediendo todos los valores de columnas. Cuando éstos se agotan, avanza la primera repetición, recorriendo todos los valores de columnas, pero ahora con un nuevo valor de fila.

`for fila in range(0, 3):` `  for columna in range(0, 3):` `  print "Estoy en la fila " + `fila``  `+ " columna " + `columna``	Estoy en la fila 0 columna 0 Estoy en la fila 0 columna 1 Estoy en la fila 0 columna 2 Estoy en la fila 1 columna 0 Estoy en la fila 1 columna 1 Estoy en la fila 1 columna 2 Estoy en la fila 2 columna 0 Estoy en la fila 2 columna 1 Estoy en la fila 2 columna 2

Un programa sencillo para ilustrar el efecto de los recorridos anidados y su resultado. Podemos compartirlo con los grupos y pedirles que predigan su comportamiento.

La instrucción **elif** se agrega a una alternativa condicional para analizar una nueva condición luego de que no se cumpliera una previa. Su nombre proviene de unir **else**, que sería equivalente a un "si no" con **if**, que representaría un nuevo "si".

`if condición1:`
`  bloque1`
`elif condición2:`
`  bloque2`
`else:`
`  bloque3`
	`if condición1:`
	`  bloque1`
	`else:`
	`  if condición2:`
	`  bloque2`
	`  else:`
	`  bloque3`

Dos programas equivalentes, uno utilizando la construcción `elif` y el otro únicamente `if` / `else`. Observamos que, en ambos: el `bloque1` se ejecuta cuando se cumple la `condición1`; el `bloque2` cuando no se cumple la `condición1`, pero sí la `condición2` (en el segundo caso, se ejecuta el bloque `else` asociado a la `condición1`, y como éste consiste en otra alternativa condicional, se evalúa la `condición2` y si su resultado es `True`, se ejecuta el `bloque2`); y el `bloque3` cuando ninguna de las dos es verdadera.	

El operador `not`, aplicado sobre una condición (o valor de verdad), produce la condición contraria (o sea, su negación). Es decir, `not(condición)` será verdadero cuando la condición sea falsa y viceversa. En nuestro programa, para saber si un número es impar (en este caso el número de fila o columna), aplicamos `not` sobre el método `img.esPar(`*`número`*`)`.

El operador `and` combina dos condiciones (o valores de verdad) para expresar una nueva, que es verdadera cuando ambas se cumplen. En inglés "and" quiere decir "y", y de hecho, esta es la interpretación más directa que podemos darle: `condición1 and condición2` será verdadera cuando se cumplan la `condición1` y la `condición2`, y falsa en cualquier otro caso.

Con estas aclaraciones, esperamos que completen el programa para producir una solución similar a la siguiente:

```
import img
def pintarTableroDeAjedrez():
       imagen = img.nuevaByN(8 ,8)
           for fila in range(0, 8):
               for columna in range(0, 8):
                      if (img.esPar(fila) and img.esPar(columna)):
                             img.pintarPixel(imagen, fila, columna)
                      elif (not(img.esPar(fila)) and not(img.esPar(columna))):
                             img.pintarPixel(imagen, fila, columna)
```

```
        img.mostrarByN(imagen)
```

Al igual que antes, para terminar, proponemos parametrizar el tamaño del tablero, aunque, por simplicidad, conservaremos la restricción de que sea cuadrado. Una solución es la siguiente:

```
import img
def pintarTablero(lado):
      imagen = img.nuevaByN(lado ,lado)
          for fila in range(0, lado):
              for columna in range(0, lado):
                    if (img.esPar(fila) and img.esPar(columna)):
                          img.pintarPixel(imagen, fila, columna)
                    elif (not(img.esPar(fila)) and not(img.esPar(columna))):
                          img.pintarPixel(imagen, fila, columna)
          img.mostrarByN(imagen)
```

Como actividad extra podemos proponerle a los grupos que dibujen una imagen en blanco y negro de su preferencia en una cuadrícula y luego hagan un programa en Python que la dibuje en pantalla.
Además, si todos lo completan, podemos pedir que nos entreguen, por un lado, el dibujo y por otro, el programa. Luego, entre todos, para cada dibujo, deberemos deducir qué programa lo produce. Esto es muy fácil de verificar: basta con ejecutar el programa candidato.

Cierre

Para cerrar la actividad comparamos los programas que hicimos en esta actividad con los que hicimos cuando trabajamos procesando secuencias de texto o de números. Concluimos que en ambos casos estamos recorriendo algún tipo de estructura: en los anteriores, la estructura es lineal, es decir, es una estructura en la que solo se puede avanzar “hacia adelante”. En cambio, en estos, la estructura es bidimensional, es decir, se puede recorrer “hacia adelante” pero también “hacia abajo”. Es por eso que en el código de los programas de esta actividad aparecen dos estructuras `for in` anidadas (es decir, una para recorrer las filas y, una vez elegida una fila, ir recorriendo las columnas), mientras que en los anteriores aparecía una sola.

Actividad 2 | SD2

Programamos en color

Objetivos

- Conocer las herramientas que utilizaremos para trabajar con imágenes en color.
- Repasar los fundamentos de la representación RGB.
- Escribir programas que modifiquen imágenes.

Modalidad de trabajo

Grupal

Materiales y recursos utilizados

- Computadoras con Python y nuestras bibliotecas instaladas

Bajada para el aula

En esta actividad y la siguiente retomaremos la representación RGB de imágenes para hacer programas que, a partir de los valores de color de los píxeles, produzcan efectos particulares, lo que popularmente conocemos como *filtros*. Para esto, primero haremos programas que generen algunas imágenes simples (colores plenos, barras de ajuste o colores aleatorios) para luego trabajar sobre fotografías propias, que cargaremos y modificaremos con nuestros programas. Propondremos algunos filtros para que los grupos elijan los que más les interesen. Además, los invitaremos a que experimenten modificando sus programas para obtener filtros nuevos que socializaremos al final de la actividad.

Primera parte: generando imágenes en RGB.

Las consignas de esta parte apuntan a que los grupos ejerciten el trabajo con la representación RGB, con problemas similares a los que trabajaron en la actividad anterior. Por eso, deberán escribir programas que generen distintas imágenes, no ya en blanco y negro, sino en color.

> *Primera consigna: definir el programa* `pintarLiso(filas, columnas)`*, que produce una imagen de las dimensiones indicadas toda de un mismo color (a elección) .*
> Aclaración: dado que en la actividad anterior parametrizamos los métodos, en esta consigna proponemos, de entrada, trabajar con parámetros para las dimensiones. Si viéramos que el manejo de los parámetros por parte de los grupos todavía no es del todo fluido, podemos replicar el mismo esquema de trabajo: hacer una primera definición con una dimensión fija y luego parametrizarla.

Si hiciera falta, podemos comenzar con un repaso del sistema de representación de colores en RGB[11]. El algoritmo de este programa es muy simple, pero podemos explicitarlo antes de comenzar:

*Pintar color liso (*ancho, alto*):*
Crear una imagen en blanco de alto *filas x* ancho *columnas.*
Para cada fila de la imagen entre 0 y alto*:*
Para cada columna de la fila actual entre 0 y ancho*:*
Pintar el píxel del color elegido
Al terminar de recorrer todas las filas y columnas mostrar la imagen obtenida.

Para esto no brindaremos código para completar, dado que es muy similar a los programas anteriores. Si algún grupo se sintiera muy inseguro para comenzar a escribir desde cero, les sugerimos explícitamente que tomen alguno de los programas que ya hicieron como modelo.

Para resolver este problema necesitan las versiones *a color* de las herramientas que utilizaron en la actividad anterior:

- `img.crearRGB(filas, columnas)` crea una imagen RGB de las dimensiones indicadas, con todos sus píxeles negros. Es análogo a `img.nuevaByN(filas, columnas)`.
- `img.colorearPixel(imagen, fila, columna, valorRojo, valorVerde, valorAzul)` cambia el color del pixel indicado por `fila` y `columna`, poniéndole los valores de rojo, verde y azul de los respectivos parámetros[12] en la imagen recibida en el primer parámetro. Es análogo a `img.pintarPixel(imagen, fila, columna)`, solo que ahora, por tratarse de imágenes en color, es necesario especificar, además, valores de rojo, verde y azul para indicar de qué color queremos pintar el pixel.
- `img.mostrarRGB(imagen)` muestra una imagen codificada en RGB. Es análogo a `img.mostrarByN(imagen)`.

Una solución posible, que genera una imagen del color (128, 128, 0) (es decir, un amarillo oscuro) tomando como modelo el último programa de la actividad anterior es:

```
import img
def pintarLiso(filas, columnas):
	imagen = img.crearRGB(filas, columnas)
	    for fila in range(0, filas):
	        for columna in range(0, columnas):
	            img.colorearPixel(imagen, fila, columna, 128, 128, 0)
	img.mostrarRGB(imagen)
```

Extensión posible: parametrizar el color, es decir, programar el método `pintarLisoParam(filas, columnas, valorRojo, valorVerde, valorAzul)` que nos permite pasar como parámetro, no solo las dimensiones sino también el color del que

[11] Proponemos solo un repaso. Hay actividades completas sobre este tema en el capítulo de representación de la información.
[12] Debemos recordar que el rango válido para estos valores es entre 0 y 255 inclusive.

queramos que se pinte.

Para hacer esta modificación basta con reemplazar la línea en la que se colorea el pixel por:

```
img.colorearPixel(imagen, fila, columna, valorRojo, valorVerde, valorAzul)
```

Segunda consigna: adaptar el programa `pintarTableroDeAjedrez` *de la actividad anterior a* `pintarTableroDeAjedrezColor(lado, color1, color2)`*, que produce un tablero de ajedrez pero donde las casillas (o* escaques) *no son blancas o negras sino de* `color1` *o* `color2` *.*
Aclaración: solo por generalidad, desarrollaremos el ejemplo a partir de la versión parametrizada del método que pinta el tablero de ajedrez en blanco y negro, pero a loos efectos de la actividad es indistinto.

Para resolver este problema, los grupos deberán conocer cómo representamos un color en nuestros programas (observemos que en este método indicamos cada color con un solo parámetro, mientras que el anterior, tomaba tres: uno para el rojo, otro para el verde y otro para el azul del color que queríamos indicar). Para indicar un color con un único valor, utilizamos una lista de tres valores, donde el primero representa el rojo, el segundo el verde y el tercero el azul. Por ejemplo, el color que tiene 255 de rojo, 128 de verde y 0 de azul (un anaranjado), lo representamos con la lista `[255, 128, 0]`.

Si quisieran acceder a cada elemento de la lista por separado, deberán utilizar el operador `[ _ ]` e indicar la posición de la lista. [0] para el rojo (recordemos que cuando contamos elementos de listas comenzamos en 0), [1] para el verde y [2] para el azul. En nuestro ejemplo, si `c` es la lista / color `[255, 128, 0]`, `c[0]` equivale a 255, `c[1]` equivale a 128 y `c[2]` a 0.

También les puede resultar útil el método `img.reemplazarPixel(imagen, fila, columna, color)` que cambia el color del pixel indicado por fila y columna, reemplazándolo por `color`. Observemos que, a diferencia de `img.colorearPixel`, este método toma el color como un parámetro (y por lo tanto, una lista) en vez de por sus tres componentes por separado.

`img.reemplazarPixel(im, 10, 10, [255, 128, 0])`	`img.colorearPixel(im, 10, 10, 255, 128, 0)`
`c = [255, 128, 0]` `img.reemplazarPixel(im, 10, 10, c)`	`c = [255, 128, 0]` `img.colorearPixel(im, 10, 10, c[0], c[1], c[2])`
Maneras equivalentes de establecer el color del pixel (10, 10) a (255, 128, 0). Observamos la diferencia entre los dos métodos: mientras que `img.colorearPixel` trabaja con las componentes del color por separado, `img.reemplazarPixel` lo considera un único valor que contiene a los otros tres.	

Para comenzar a programar, les decimos a los grupos que retomen el programa para pintar el tablero en blanco y negro, hagan una copia y vayan modificándolo para que cumpla con esta nueva consigna. Los guiamos para que observen qué alteraciones deberían hacerle al método original. A continuación presentamos las dos versiones:

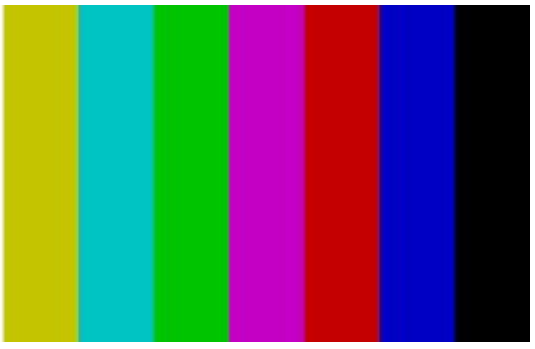
```
def pintarTableroDeAjedrez(lado):
    imagen = img.nuevaByN(lado , lado)
    for fila in range(0, lado):
        for columna in range(0, lado):
            if (img.esPar(fila) and
img.esPar(columna)):
                img.pintarPixel(imagen, fila,
columna)
            elif (not(img.esPar(fila)) and
not(img.esPar(columna))):
                img.pintarPixel(imagen, fila,
columna)



    img.mostrarByN(imagen)
```

```
def pintarTableroDeAjedrezColor(lado, color1,
color2):
    imagen = img.crearRGB(lado , lado)
    for fila in range(0, lado):
        for columna in range(0, lado):
            if (img.esPar(fila) and
img.esPar(columna)):
                img.reemplazarPixel(imagen,
fila, columna, color1)
            elif (not(img.esPar(fila)) and
not(img.esPar(columna))):
                img.reemplazarPixel(imagen,
fila, columna, color1)
            else:
                img.reemplazarPixel(imagen,
fila, columna, color2)
    img.mostrarRGBTalCual(imagen)
```

Soluciones para el problema en blanco y negro (izquierda) y en color (derecha). Observamos la similitud de la estrategia y el uso de los métodos para color, análogos a los de blanco y negro.

Como primera aproximación, podemos sugerirles a los grupos que lo necesiten que armen un tablero de un color y negro (recordando que los píxeles a los que no les cambiamos el color, quedan en negro). Para esto, alcanza con hacer las modificaciones marcadas en naranja. Luego, podemos recordar que la estructura `if elif` permite extenderse a `if elif else` lo que nos permite agregar código para que se ejecute cuando no se cumple ni la condición de `if` ni la de `elif`. Observamos que, justamente, esto es lo que queremos en este caso: cuando no se cumplen ninguna de las condiciones de pintar del primer color, hay que pintar del segundo.

Continuación posible: definir `pintarBarrasPAL()` que produce una imagen de 800 x 800 con las barras de ajuste de la señal PAL:



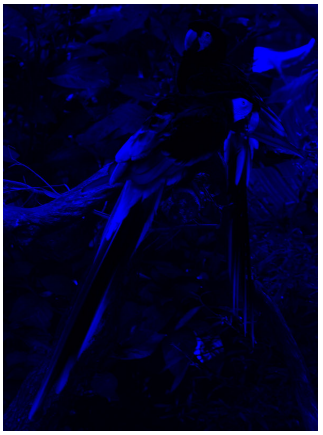
La particularidad de esta señal de ajuste es que está compuesta por los colores primarios, sus complementarios y el blanco y negro puros. En definitiva, estos colores codificados en RGB resultan en: (255, 255, 255), (255, 255, 0), (0, 255, 255), (0, 255, 0), (255, 0, 255), (255, 0, 0), (0, 0, 255), (0, 0, 0). Podemos encontrar diversos patrones, como que el azul alterna entre una barra y la otra, el verde solo está presente en las primeras cuatro y que el rojo aparece alternando de a dos.

Segunda parte: modificando los colores de nuestras imágenes

En esta segunda parte propondremos hacer programas para que alteren los colores de las imágenes, imitando algunos filtros clásicos.

Primera consigna: definir el programa `soloAzul()` *que carga una imagen y deja solo su componente azul.*

Imagen original sin modificaciones.



Ejemplo de resultado del programa *soloAzul()*.

Como siempre, proponemos comenzar planteando el algoritmo. Una posibilidad aparece a continuación:

SoloAzul:
Cargar imagen RGB
Para cada fila de la imagen entre 0 y la cantidad de filas de la imagen:
Para cada columna de la fila actual entre 0 y la cantidad de columnas de la imagen:
Colorear el pixel de la fila y columna con rojo y verde = 0 y azul el valor que tenía.
Al terminar de recorrer todas las filas y columnas mostrar la imagen obtenida.

Damos tiempo para que, comenzando a partir de los programas confeccionados en ítems anteriores, surjan las siguientes dificultades y sus posibles soluciones:

- En vez de crear una imagen en blanco, utilizamos el método `img.cargarRGB(ruta)` que, dada la ubicación de un archivo de imagen (escrita entre comillas, como texto), carga dicha imagen para poder utilizarla dentro del

programa. Este método es similar a `img.crearRGB`, solo que el resultado contiene a nuestra imagen y no un rectángulo negro.

- Dado que la cantidad de filas y columnas ya no está fija a priori, sino que depende de la imagen que carguemos, necesitamos los métodos `img.cantidadDeFilas(foto)` e `img.cantidadDeColumnas(foto)`, que nos permiten obtener la cantidad de filas y de columnas (resp.) a partir de una imagen cargada.

- Necesitamos conocer el valor de azul de un pixel para establecer su valor final. Para esto contamos con los siguientes métodos:
  - `img.azulDelPixel(imagen, fila, columna)`, que retorna el valor númerico entre 0 y 255 que posee el azul en el píxel ubicado en la imagen, fila y columna indicadas como parámetros. Comentamos que también existen métodos análogos para los otros colores primarios: `img.verdeDelPixel(imagen, fila, columna)` y `img.azulDelPixel(imagen, fila, columna)`
  - `img.valorDelPixel(imagen, fila, columna)`. Este método obtiene el valor del color del pixel, representado, al igual que antes, como una lista con tres valores (el del rojo, el del verde y el del azul).

Una solución posible es la siguiente:

```
import img
def soloAzul():
	foto = img.cargarRGB('mi_foto_color.png')
	filas = img.cantidadDeFilas(foto)
	columnas = img.cantidadDeColumnas(foto)
	for filaActual in range(0, filas):
		for columnaActual in range(0, columnas):
			valorDeAzul = img.azulDelPixel(foto,filaActual,columnaActual)
			img.colorearPixel(foto,filaActual,columnaActual,0,0,
			valorDeAzul)
	img.mostrarRGB(foto)
```

Para continuar: definir los programas `soloVerde()` *y* `soloRojo()`*, análogos a* `soloAzul()`*, pero para los otros dos colores primarios.*

Antes de comenzar, observamos similitudes con el problema anterior. Podemos plantear los algoritmos, que solo variarán en el paso en el que computan el color nuevo del píxel en base al original, descartando la información de los colores que no estamos eligiendo. Los tres programas resultantes serán muy similares, hecho que podemos resaltar escribiéndolos uno al lado del otro:

<table>
<tr><td colspan="3">import img</td></tr>
<tr><td>def soloAzul():</td><td>def soloRojo():</td><td>def soloVerde():</td></tr>
<tr><td colspan="3">foto = img.cargarRGB('mi_foto_color.png')
filas = img.cantidadDeFilas(foto)
columnas = img.cantidadDeColumnas(foto)</td></tr>
</table>

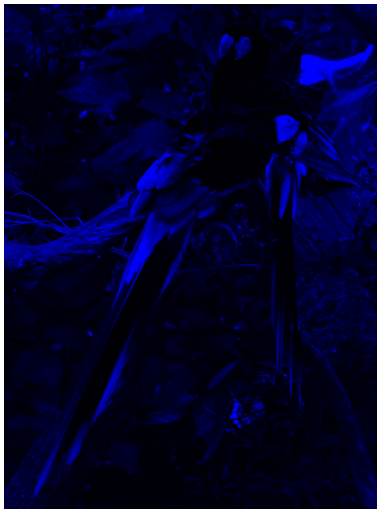
```
        for filaActual in range(0, filas):
                for columnaActual in range(0, columnas):
```

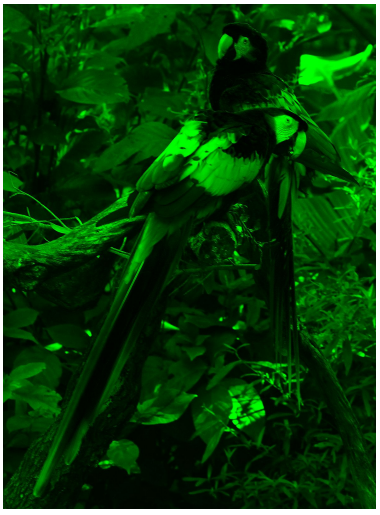
```
valorDeAzul = img.azulDelPixel(foto,filaActual,columnaActual)

img.colorearPixel(foto,filaActual,columnaActual,0,0 valorDeAzul)
```

```
valorDeRojo = img.rojoDelPixel(foto,filaActual,columnaActual)

img.colorearPixel(foto,filaActual,columnaActual,valorDeRojo, 0, 0)
```

```
valorDeVerde = img.verdeDelPixel(foto,filaActual,columnaActual)

img.colorearPixel(foto,filaActual,columnaActual,0, valorDeVerde, 0)
```

```
        img.mostrarRGB(foto)
```





Podemos aprovechar esta comparación para poner en evidencia lo similar de los algoritmos detrás de los tres programas: siempre recorremos todos los píxeles (avanzando por las filas y las columnas) y, para cada uno, obtenemos su color y lo utilizamos para producir, de alguna manera en particular, un nuevo valor de color para reemplazarlo. Por ejemplo, en el caso de `soloAzul`, a partir del color del píxel, obtenemos su parte azul y generamos un nuevo color con este valor y el resto de los primarios en 0.

Segunda consigna: definir el programa `reemplazarColores()` *que intercambia los valores de color de cada pixel: rojo por verde, verde por azul y azul por rojo.*

Motivamos a los grupos a que primero piensen el algoritmo, como una modificación al de la consigna anterior:

ReemplazarColores:
Cargar imagen RGB.
Para cada fila entre 0 y la cantidad de filas de la imagen:
Recorrer cada columna entre 0 y la cantidad de columnas de la imagen:
Colorear el píxel de la fila y columna actual con
rojo = verde, verde = azul y azul = rojo.
Mostrar imagen RGB.

A continuación presentamos dos soluciones posibles, una que utiliza los métodos trabajados en la consigna anterior y otra que retoma `img.varlorDelPixel` e `img.reemplazarPixel`.

```
import img
def reemplazarColores():
```

```
	foto = img.cargarRGB('mi_foto_color.png')
	filas = img.cantidadDeFilas(foto)
	columnas = img.cantidadDeColumnas(foto)
	for filaActual in range(0, filas):
		for columnaActual in range(0, columnas):
```

```
valorDeRojo = img.rojoDelPixel(foto,filaActual,columnaActual)
valorDeVerde = img.verdeDelPixel(foto,filaActual,columnaActual)
valorDeAzul = img.azulDelPixel(foto,filaActual,columnaActual)
```

```
colorOriginal = img.valorDelPixel(foto,filaActual,columnaActual)

colorNuevo = [colorOriginal[1], colorOriginal[2], colorOriginal[0]]

img.reemplazarPixel(foto, filaActual, columnaActual, coloresNuevos)
```

`img.colorearPixel(foto,filaActual,columnaActual, valorDeVerde, valorDeAzul, valorDeRojo)`	
`img.mostrarRGB(foto)`	

Dado que la primera solución es más parecida a las anteriores, es esperable que las soluciones de los grupos sean similares a ésta. Sin embargo, nos interesa la segunda para mostrar explícitamente los valores del color original y el color transformado. Podemos mostrarla o, escribirla entre todos en el pizarrón, como una versión alternativa que usa los métodos `img.valorDelPixel` e `img.reemplazarPixel`.

Cierre

Para terminar la actividad, reforzamos la semejanza entre todos los algoritmos de la segunda parte. Observamos que consisten en recorrer todos los píxeles de la imagen y, a partir del valor de color del pixel original, producir uno nuevo. Enfatizamos que cada uno de los filtros es, en definitiva, una manera particular de producir el pixel definitivo a partir del pixel original.

Entrada / Pixel Original	Filtro	Salida / Pixel modificado
[rojo, verde, azul]	`soloAzul`	[0, 0, azul]
	`soloVerde`	[0, verde, 0]
	`soloRojo`	[rojo, 0, 0]
	`reemplazarColores`	[verde, azul, rojo]

Actividad 3 | SD2

Hacemos nuestros filtros

Objetivos

- Programar filtros de imágenes.
- Presentar el concepto de función y utilizarlas para construir programas.
- Reforzar el manejo de la codificación RGB.

Modalidad de trabajo

Grupal

Materiales y recursos utilizados

- Computadoras con Python y nuestras bibliotecas instaladas

Bajada para el aula

En esta actividad los grupos continuarán trabajando sobre imágenes. Primero desarrollaremos el concepto de función, como un método que produce un resultado, y luego, presentaremos distintas opciones de filtros, cada una con sus variantes, para que los grupos elijan cuáles programarán.

Primera parte: funciones

En esta primera parte presentaremos las funciones como una nueva herramienta para escribir programas más claros. Así como utilizábamos los métodos para separar el programa en fragmentos que tuvieran un comportamiento determinado, utilizaremos las funciones para encapsular el cómputo de valores de salida a partir de valores de entrada. Por ejemplo, retomando el programa `reemplazarColores` de la actividad anterior, podemos definir la función `coloresIntercambiados(color)` que, dado el color de un píxel, produce el color que resulta de intercambiar sus componentes como en la actividad anterior. En este sentido, la función `coloresIntercambiados` se comporta como una caja negra que recibe el color de un pixel y genera otro con características específicas. Al igual que sucede con los procedimientos, ahora podemos usar `coloresIntercambiados` en cualquier lugar de nuestros programas como si fuera una instrucción más, pero también podemos conocer (y modificar) su definición.

```
...
colorOriginal = img.valorDelPixel(im, fila, columna)
colorNuevo = coloresIntercambiados(colorOriginal)
img.reemplazarPixel(im, fila, columna, colorNuevo)
...
```

Un fragmento de una posible versión de `reemplazarColores` utilizando funciones. Observamos que utilizamos la función `coloresIntercambiados` para generar el color definitivo, que luego se le asigna al píxel.

Para definir funciones en Python, se utiliza el mismo esquema que para los métodos agregando la instrucción `return` (devolver, en inglés), que, además de indicar el valor que debe ser devuelto (o producido) por la función, hace que se termine la ejecución de la función y se continúe con el programa donde ésta aparecía usada.

```
def coloresIntercambiados(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    return [valorVerde, valorAzul, valorRojo]
```

Una definición posible de `coloresIntercambiados`. Observamos que en la última línea utilizamos la instrucción `return` para indicar que el valor producido (devuelto) por la función debe ser el color con los componentes en ese orden. Además, esta instrucción indica que debe terminar la ejecución de esta función.

Para trabajar este concepto con los estudiantes, retomamos la observación de la actividad anterior, en la que concluimos que todos los filtros que habíamos programado sobre imágenes a color, consistían en tomar el valor original del color de cada píxel y, en base a éste, producir un valor nuevo según alguna regla determinada (por ejemplo, descartar todas las componentes menos el azul o intercambiar el orden de los colores primarios). Contamos, entonces, que con una motivación similar a la de los métodos, en Python podemos definir funciones para separar del cuerpo de un programa un fragmento que esté destinado a producir un valor en particular. Mostramos el ejemplo de la nueva definición de `coloresIntercambiados` y proponemos que adapten, utilizando funciones, las definiciones de `soloAzul`, `soloVerde` y `soloRojo`.

```
...
colorOriginal = img.colorDelPixel(im, fila, columna)
colorNuevo = dejarSoloAzul(colorOriginal)
img.reemplazarPixel(im, fila, columna, colorNuevo)
...
```

```
def dejarSoloAzul(color):
    valorAzul = color[2]
    return [0, 0, valorAzul]
```

Posible adaptación del programa `soloAzul` utilizando funciones. Observamos que solo tenemos que modificar la parte en la que reemplazamos el color del píxel, mientras que el resto de programa (el recorrido sobre las filas y las columnas) es exactamente igual.

Primera consigna: definir el programa `pasarABlancoYNegro` *que convierte una imagen en color a su equivalente en blanco y negro.*

Para resolver esta consigna, los grupos no necesitan ninguna herramienta nueva de programación. Solamente deberán averiguar qué valor de color producir para generar un gris acorde al color original. Para esto, podemos preguntarles qué tienen de particular los grises representados en RGB, para recordar que el valor de los tres componentes es igual (y, justamente por eso, como ningún color predomina sobre otro, se produce un tono neutro). El próximo paso, entonces, es preguntarse por qué valor deberán colocar en los tres colores, a lo que responderemos que una primera solución puede ser considerar el promedio de los tres canales. Dejamos que los grupos trabajen, estando atentos a su avance e insistimos con que definan una función que produzca el pixel gris a partir del color del pixel original.

```
...
colorOriginal = img.colorDelPixel(im, fila, columna)
colorNuevo = calcularGris(colorOriginal)
img.reemplazarPixel(im, fila, columna, colorNuevo)
...
```

```
def calcularGris(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    promedio[13] = (valorRojo/3 + valorVerde/3 + valorAzul/3)
    return [promedio, promedio, promedio]
```

Observamos que si cambiamos la función `dejarSoloAzul` por `calcularGris` en el programa anterior (y definimos esta última como corresponde), ya tenemos un programa para convertir a blanco y negro, sin necesidad de ninguna otra modificación.

Variaciones: Proponemos explorar otras maneras posibles de calcular el valor de gris a partir de los colores primarios. Para esto, los grupos deberán identificar que solo

[13] A diferencia de como solemos calcular un promedio (primero sumando los números y después dividiendo por la cantidad), en este caso, primero dividimos cada valor y luego lo sumamos. Esto se debe a que el formato de representación utilizado no permite números mayores a 255. Luego, si cuando estamos calculando la suma se excede este límite (lo cual es muy probable) se produce un error o se fija el resultado en 255. En cualquier caso, no es el valor esperado.

necesitan cambiar la función `calcularGris`.

- `calcularGrisPerceptual`: dado que hay colores “más oscuros” que otros, para producir una imagen lo más parecida posible a cómo nosotros percibimos la luminosidad, el valor de gris se obtiene combinando 21,26% del valor de rojo, 71,52% del verde y 7,22% del azul.
- `calcularGrisCanalRojo`: otra opción es considerar solo el valor de rojo y replicarlo en los otros canales, ignorando el resto. Esto produce un efecto visualmente notable, similar al de la fotografía infrarroja.
- `calcularGrisMax`: también podemos elegir el mayor valor de los tres para replicarlo. Para esta variación, consideramos que vale la pena definir otra función que calcule el máximo de tres valores o adaptar el ejercicio en el que calculamos el máximo de una secuencia.

Más variaciones: además de las tres variaciones mencionadas, alentamos a los grupos a que experimenten con otras opciones que se les ocurran. Por ejemplo, pueden variar la fórmula de la primera alternativa con otros coeficientes (teniendo en cuenta que siempre sumen exactamente 100% para permanecer en rango), pueden probar eligiendo solo el canal azul o el verde, pueden tomar el mínimo, etc...

A continuación vemos los distintos efectos de las distintas fórmulas y programas que las computan.

			
`calcularGris`	`calcularGrisPerceptual`	`calcularGrisCanalRojo`	`calcularGrisMax`

```
def calcularGrisPerceptual(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorGrisDecimal = 0.2126 * valorRojo + 0.7152 * valorVerde + 0.0722 * valorAzul
    valorGris = int(valorGrisDecimal)
    return [valorGris, valorGris, valorGris]
```

Lo único diferente en esta función con respecto a la anterior es cómo calculamos el valor de gris. Primero aplicamos la fórmula expresada en la consigna y luego calculamos `int()` de ese valor para redondearlo a un número entero, dado que nuestra representación espera números enteros para cada uno de los colores.	

`def calcularGrisCanalRojo(color):`
`  valorRojo = color[0]`
`  return [valorRojo, valorRojo, valorRojo]`
	`def calcularGrisMax(color):`
	`  valorMax = maximo(color)`
	`  return [valorMax, valorMax, valorMax]`

Podemos proponer parametrizar el color elegido definiendo `calcularGrisCanal(color, canal)`, donde `canal` = 0 indica rojo, 1 verde y 2 azul.	La función `maximo` es una adaptación de la función definida en la actividad de secuencias para buscar el máximo, pues el color está representado como una lista.

Segunda parte: galería de filtros

Para esta parte de la actividad, propondremos diferentes filtros para que los grupos elijan los que más les interesen para programarlos. Además, debemos motivarlos (y ayudarlos) para que investiguen cómo deben manipular los colores para conseguir el efecto deseado, en vez de presentarles la fórmula como hicimos en el caso de blanco y negro. Para esto, sugerimos algunas pistas que pueden ayudarlos a descubrir esta fórmula.

El algoritmo para todos estos filtros es muy similar a los que ya veníamos trabajando: debemos recorrer todos los píxeles de la imagen (avanzando por las filas y las columnas), obtener el color original y producir un color nuevo aplicando alguna función, que dependerá del filtro que queramos hacer. Luego, en los ejemplos que siguen solo presentaremos esta función, mientras que las soluciones completas aparecen programadas en el archivo fuente correspondiente.

- `negativo()`: invierte los colores de la imagen.
  Pista: El negativo del blanco es el negro, o sea que el negativo de [255, 255, 255] es [0, 0, 0]. ¿Cuál será el negativo de [192, 192, 192]? ¿Cuál es el color opuesto del rojo? ¿Cómo se representa?
  Solución: tomar el negativo de cada canal, donde el negativo de un canal es simplemente lo que le falta para alcanzar el valor máximo (en este caso, 255). En este caso, podemos sugerir definir la función `invertir` para realizar esta operación.
  Variantes: se pueden utilizar otras funciones para invertir el color, por ejemplo: (255^n - x^n)^(1/n) para algún n (de hecho, la anterior, es esta función con n = 1).

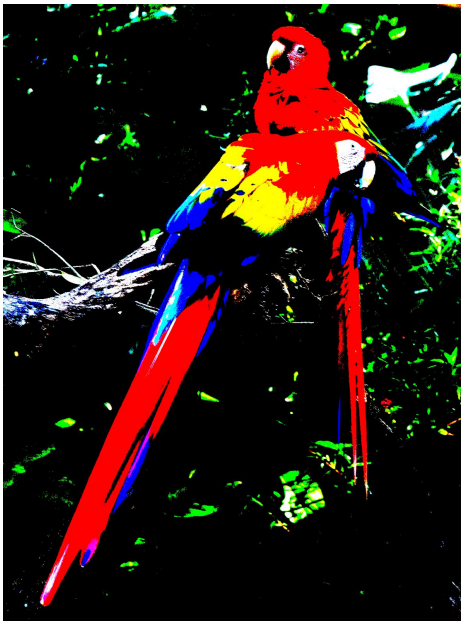
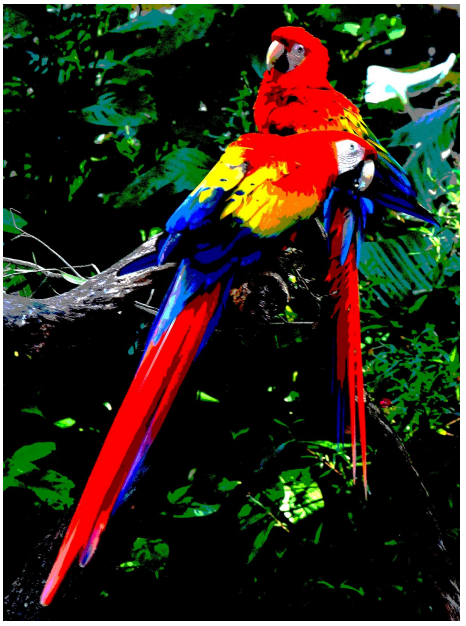
		
Imagen original	Negativo con la fórmula original	Negativo con la fórmula cuadrática

```
def calcularNegativo(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorRojoNeg = 255 - valorRojo
    valorVerdeNeg = 255 - valorVerde
    valorAzulNeg = 255 - valorAzul
    return [valorRojoNeg, valorVerdeNeg,
valorAzulNeg]
```

```
def calcularNegativo(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorRojoNeg = invertir(valorRojo)
    valorVerdeNeg = invertir(valorVerde)
    valorAzulNeg = invertir(valorAzul)
    return [valorRojoNeg, valorVerdeNeg,
valorAzulNeg]

def invertir(valor):
    return 255 - valor

def invertirCuadratico(valor):
    return  int((255**2 - valor**2)**(0.5))
```

- `posterizar()`: este filtro simula un método de representación muy básico en el que no existen valores intermedios para cada color, solamente pueden aparecer con su mayor intensidad o no aparecer.
  Pista: ¿Cuántos colores distintos puede haber en la imagen después de aplicar el filtro? ¿Cuáles son? Observamos que son 8 colores: el blanco, el negro, los primarios y los complementarios, que se representan como [255, 255, 255], [0, 0, 0], [255, 0, 0], [0, 255, 0], [0, 0, 255], [0, 255, 255], [255, 0, 255], [255, 255, 0], es decir, todas las combinaciones posibles de 0 y 255 en cada canal.
  Solución: Si el valor del canal es menor que 128, reemplazarlo por 0 (pues está más cerca de 0 que de 255) y en caso contrario, por 255 (pues está más cerca de éste que de 0).
  Variantes: Extender la cantidad de niveles posibles para cada color, es decir, en vez

de que cada canal pueda valer solo 0 o 255, por ejemplo, permitir que valga 0, 128 o 255. En este caso, utilizaríamos la función `todoONada3`, que es análoga a la anterior, pero considerando tres valores. Los números de los rangos provienen de dividir los números entre 0 y 255 en tres segmentos iguales: a los primeros, aproximarlos al 0, a los segundos al 128 y a los últimos al 255.

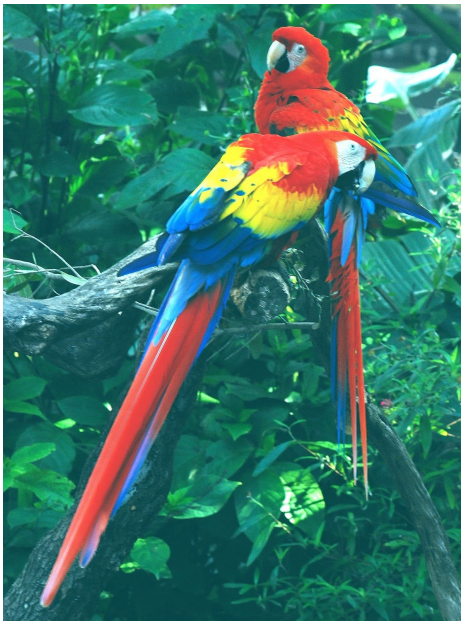
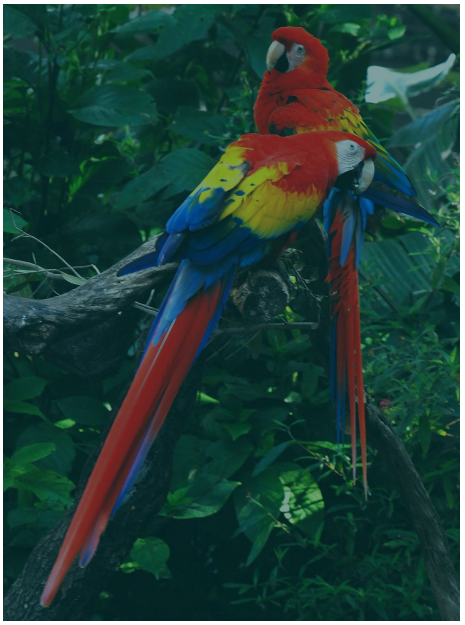
		
Imagen original	Posterizado con dos niveles	Posterizado con tres niveles

```
def posterizarPixel(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorRojoExtremo = todoONada(valorRojo)
    valorVerdeExtremo = todoONada(valorVerde)
    valorAzulExtremo = todoONada(valorAzul)
    return [valorRojoExtremo,
valorVerdeExtremo, valorAzulExtremo]
```

```
def todoONada(valor):
    if valor < 128:
        return 0
    else:
        return 255

def todoONada3(valor):
   if valor < 85:
       return 0
   elif valor < 170:
       return 128
   else:
       return 255
```

- `entintar(color)`: este filtro toma un color y lo agrega a todos los píxeles de la imagen, como si aplicáramos una capa de colorante.
  Pista: ¿Cómo haremos para mezclar dos colores en nuestra representación? Los sumamos canal a canal. ¿Qué precaución hay que tener ahora? Que al sumar, los valores se pueden salir de rango y, por lo tanto, producir valores que no son válidos en nuestra representación.
  Solución: a cada pixel le sumamos el color del que queremos entintar, componente a

componente y no permitimos que exceda el valor 255.
Variantes: pensar en otras maneras de combinar los colores. Por ejemplo, tomando el promedio, la media geométrica[14] o el máximo entre el canal original y el del píxel para entintar. Prestar atención qué pasa con los sectores más claros y más oscuros de la imagen original en cada caso.

		
Imagen original	Entintado sumando	Entintado tomando promedio

```
def entintarPixel(colorOriginal, tinte):
    valorRojo = colorOriginal[0]
    valorVerde = colorOriginal[1]
    valorAzul = colorOriginal[2]
    valorRojoNuevo = saturar(valorRojo +
tinte[0])
    valorVerdeNuevo = saturar(valorVerde +
tinte[1])
    valorAzulNuevo = saturar(valorAzul +
tinte[2])
    return [valorRojoNuevo, valorVerdeNuevo,
valorAzulNuevo]
```

```
def saturar(valor):
    if valor > 255:
        return 255
    else:
        return valor

def promedio(colorOriginal, tinte):
    return (colorOriginal + tinte)/2
```

Además, como sugerencia final, podemos proponer a los grupos que experimenten aplicando la conversión a blanco y negro antes o después de aplicar los filtros que programaron. Para hacer esto, deberán observar que simplemente se agrega un paso en la generación del color del pixel: se aplica la función `calcularGris` (o cualquiera de sus versiones), ya sea sobre el color del pixel original (para aplicar la conversión antes que el filtro) o sobre el color nuevo del pixel (para aplicar la conversión después).

[14] La media geométrica entre dos valores x1 y x2 se calcula como (x1*x2)**(1/2)

```
def posterizarPixelByN(color):
    color = calcularGris(color)
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorRojoExtremo = todoONada(valorRojo)
    valorVerdeExtremo =
todoONada(valorVerde)
    valorAzulExtremo = todoONada(valorAzul)
    colorFinal = [valorRojoExtremo,
valorVerdeExtremo, valorAzulExtremo]
    return colorFinal
```

```
def posterizarPixelByN(color):
    valorRojo = color[0]
    valorVerde = color[1]
    valorAzul = color[2]
    valorRojoExtremo = todoONada(valorRojo)
    valorVerdeExtremo = todoONada(valorVerde)
    valorAzulExtremo = todoONada(valorAzul)
    colorFinal = [valorRojoExtremo,
valorVerdeExtremo, valorAzulExtremo]
    colorFinal = calcularGris(colorFinal)
    return colorFinal
```

```
def posterizarPixelByN(color):
    colorByN = calcularGris(color)
    colorByNPoster =
posterizarPixel(colorByN)
    return colorByNPoster
```

```
def posterizarPixelByN(color):
    colorPoster = posterizarPixel(color)
    colorByNPoster =
calcularGris(colorPoster)
    return colorByNPoster
```

Observamos que las líneas agregadas (marcadas en negrita) aplican la conversión a blanco y negro. En el primer caso, antes de comenzar a trabajar, se reemplaza el valor de `color` por el resultado de su valor de gris. Por lo tanto, el efecto de posterizar se aplica sobre la imagen ya convertida a escala de gris. En el segundo, se hace lo mismo pero con el valor de `colorFinal`, lo que produce que primero se aplique el efecto de posterizar y su resultado se convierta a blanco y negro. Observamos las dos versiones (arriba y abajo): ambas son equivalentes, pero mientras que en la primera modificamos la definición de una función, en la segunda, definimos una nueva combinando las dos anteriores. Esta es otra ventaja de tener funciones individuales definidas para las transformaciones: que las podemos combinar fácilmente y de manera muy clara.

blanco y negro + negativo

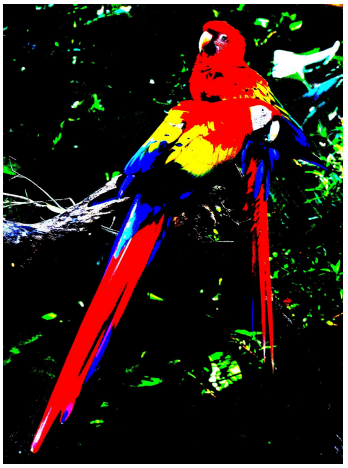
blanco y negro + posterizar

blanco y negro + entintar

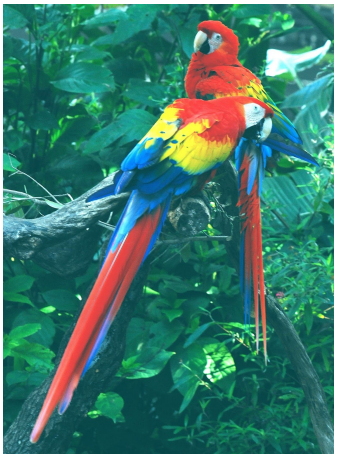
Ficha

En esta actividad trabajaremos modificando los colores de las imágenes. A continuación proponemos tres efectos para que elijan cuáles programar con su grupo.

`negativo`: invierte los colores de la imagen.



`posterizar`: produce una imagen combinando únicamente colores primarios puros (es decir, al máximo de su intensidad).



`entintar`: agrega a toda la imagen un mismo color, por ejemplo, cian.

¿Qué variantes admiten los efectos elegidos? ¿Cómo afectan a la imagen?

Sugerencia: prueben qué sucede si aplicamos el filtro de blanco y negro antes o después de alguno de los filtros que ya programaron.

Parte II: Proyecto de Sintetizador | Programas que suenan

Introducción

Cerraremos el recorrido de programación trabajando sobre un nuevo proyecto. En este caso, los estudiantes construirán un sintetizador de sonido, es decir, un programa capaz de generar archivos de audio a partir de una melodía codificada en un texto.

Para esto, deberán primero investigar sobre la naturaleza del sonido y luego, retomando las herramientas del capítulo anterior, escribir sucesivos programas en Python para ir haciendo crecer las capacidades de su sintetizador. Finalmente, deberán proponer un sistema de representación de melodías, escribir un programa capaz de decodificarlas y utilizar las funciones que definieron previamente para generar el archivo de sonido final.

Actividad 1A | SD1

¿Cómo es el sonido?

Objetivos

- Presentar una aproximación al fenómeno físico que percibimos como sonido.
- Conocer la noción de frecuencia y asociarla con el tono o altura del sonido.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras con micrófono y parlantes / Teléfonos celulares
- Aplicación de osciloscopio
- Instrumentos musicales (opcional)
- Proyector (opcional).

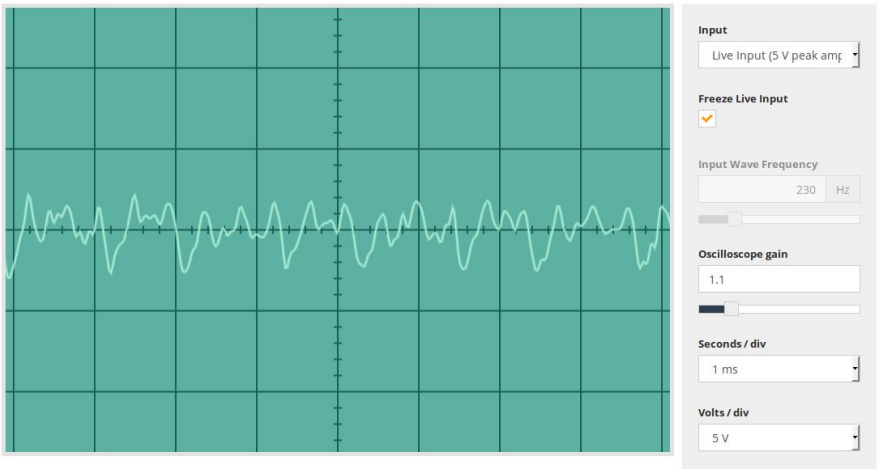
Desarrollo

Comenzamos este proyecto comentándoles a los grupos que programarán su propio sintetizador de sonido. Es decir, escribirán una serie de programas en Python capaces de generar archivos de sonido, que luego podrán ser reproducidos en la computadora.

En esta actividad, comenzaremos a explorar algunas nociones que nos permitirán comprender cómo se almacenan sonidos en una computadora. Este es un primer paso necesario para escribir programas que nos permitan procesar y aplicar efectos de audio y generar nuestros propios sonidos.

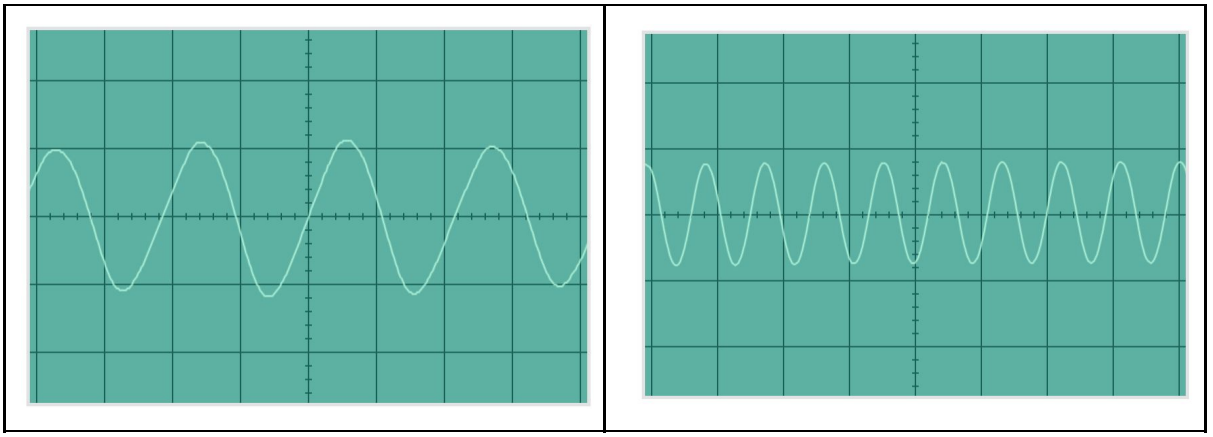
Para comenzar, observaremos una representación gráfica del sonido que, esperamos, sea familiar para los estudiantes[15]. Para esto, utilizaremos un instrumento denominado **osciloscopio**. Pueden usar uno en línea (https://academo.org/demos/virtual-oscilloscope/, en inglés, es importante que elijan la opción Input -> Live input) o, incluso, podemos pedirles que descarguen alguna aplicación para celulares que simule uno (existen muchas gratuitas, basta con buscar en la tienda de aplicaciones). No decimos más, y simplemente les pedimos que experimenten con distintos sonidos y observen el resultado. Si esto no fuera posible, podemos reproducir algún video donde se haga esto, como por ejemplo (https://www.youtube.com/watch?v=XFYLK_b9--4). Esperamos que observen algo similar a la imagen que vemos a continuación:

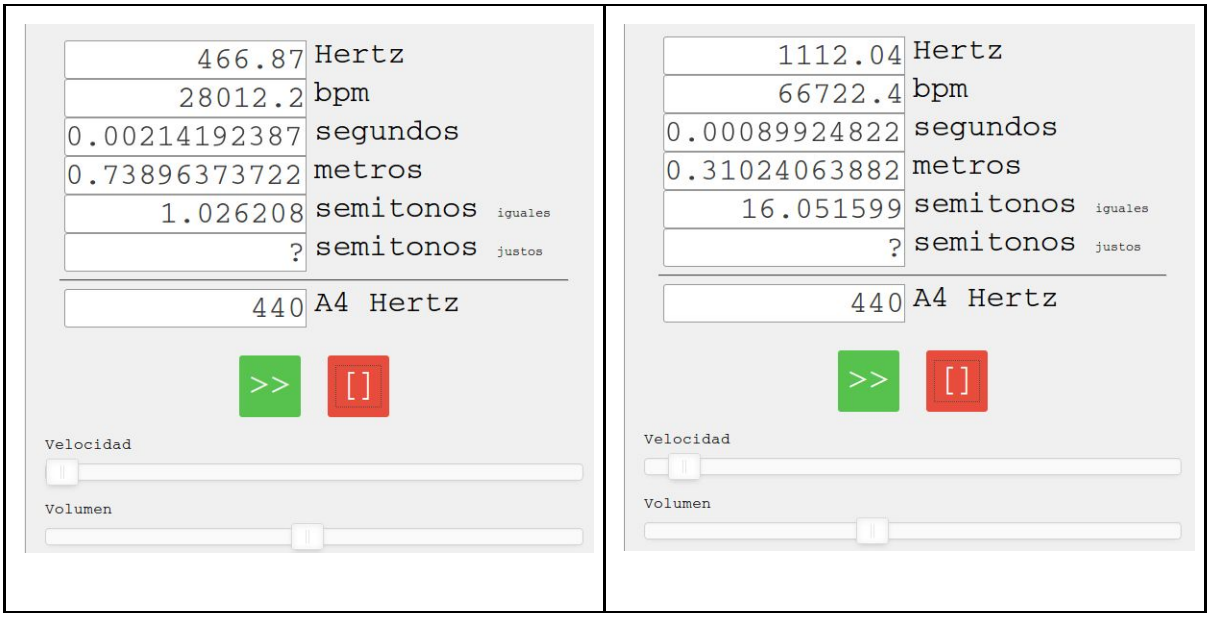
[15] Si ya hubieran realizado una experiencia similar o hayan visto nociones de sonido en Física, podemos omitir esta introducción experimental.



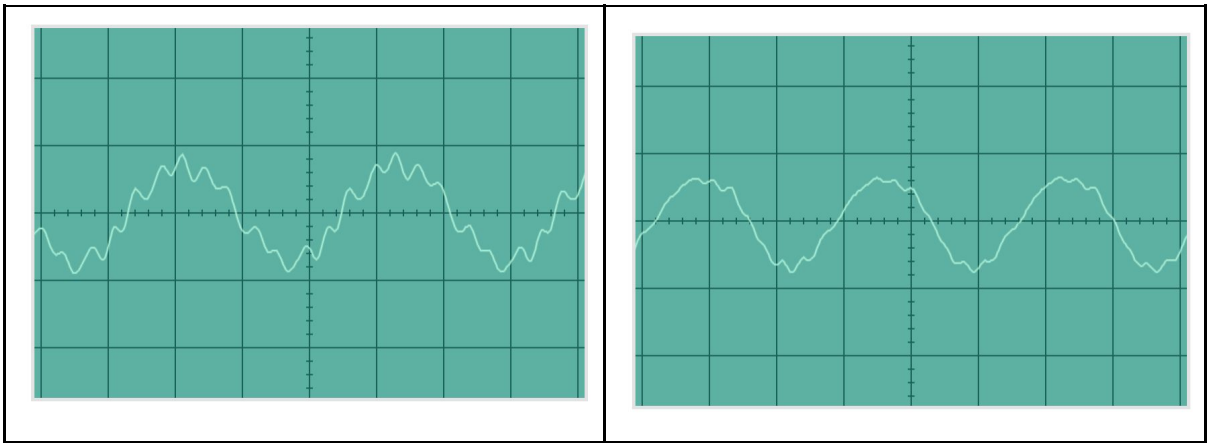
La primera observación es que el dibujo parece periódico, es decir, compuesto por un mismo patrón que se repite. Luego, preguntamos si encuentran alguna relación con cómo se escucha el sonido y con cómo se ve en el osciloscopio. El primer comentario que nos interesa hacer al respecto es que, cuanto más intenso es el sonido, más “alto” es el gráfico.

Para continuar, les pedimos que experimenten cómo se ven los sonidos graves y los agudos. Para esto, pueden producir sonidos con algún instrumento musical o con su propia voz (en ese caso, les sugerimos que pronuncien siempre una misma consonante, en particular, la m o la l, que consisten en sonidos más “simples”, es decir, donde el dibujo es más claro). También, podemos utilizar un generador de sonidos virtual, como https://lengua.la/gtp/?n=1, donde deberemos variar la barra de velocidad. Es importante que no se mezclen los sonidos de los diferentes grupos para poder observar el fenómeno sin interferencias. Recomendamos unir grupos de manera que queden distanciados en el aula, turnarlos, o realizar el experimento nosotros bajo sus directivas y proyectar el resultado.





Un sonido grave (a la izquierda) y otro agudo (a la derecha), generados artificialmente y vistos en el osciloscopio.



Pronunciando la letra *I*, primero más grave y después más aguda. Observamos que, al trabajar con la voz humana, la forma es más compleja (lo que puede hacer que el fenómeno sea difícil de observar).

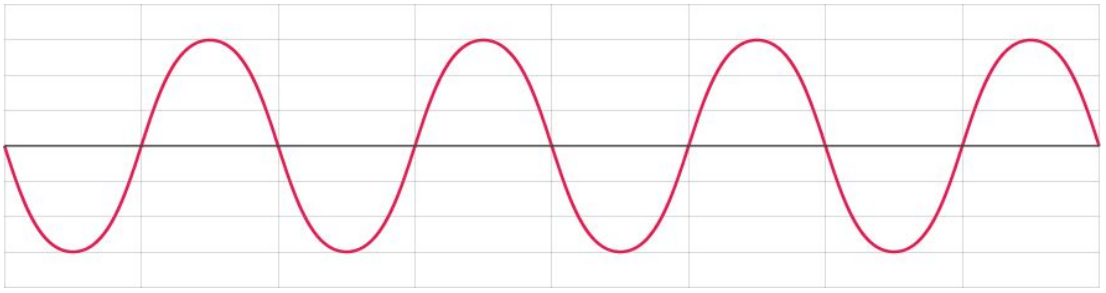
Entonces la pregunta es “¿Qué cambió entre un sonido y otro? ¿Y entre un gráfico y otro? ¿Cómo se relacionan?”. El objetivo de este experimento es observar que los sonidos graves se corresponden con dibujos más “anchos” mientras que los agudos, con dibujos más “angostos” (y, por lo tanto, que se repiten más veces), aunque en la forma son similares.

Ahora que ya observamos algunas relaciones entre los sonidos que oímos y la forma que adopta el osciloscopio, cabe preguntarnos qué es exactamente lo que está graficando. Para esto, podemos explicar que los **sonidos** son vibraciones del aire que somos capaces de percibir con nuestros oídos. Cuando suena la cuerda de una guitarra, la vemos oscilar entre dos posiciones (es decir, moverse repetidas veces hacia un lado y hacia otro, a mucha velocidad). Este movimiento produce que el aire que la rodea se comprima y se expanda. Además, estas compresiones y expansiones también afectan al aire que tienen alrededor y se propagan: si se expanden, “empujan” al aire que está a su alrededor, produciendo que se comprima, y si se comprimen, “dejan lugar” y permiten al aire circundante que se expanda.

Esto produce una nueva perturbación en el aire que rodeaba a la original, que, a su vez producirá nuevas perturbaciones en sus alrededores, que, a su vez, perturbarán el aire que está más allá… A este tipo de fenómenos, donde hay una propagación de una perturbación periódica (es decir, que se repite igual en el tiempo), los llamamos **onda**.

Finalmente, cuando estas perturbaciones llegan a nuestros oídos, hacen vibrar una membrana llamada tímpano. Estas vibraciones producen estímulos eléctricos que son recibidos por el cerebro e interpretados como un sonido. La computadora que utilizamos como osciloscopio no tiene un tímpano, pero sí un micrófono que funciona de una manera muy similar: cuenta con una membrana capaz de vibrar y producir electricidad a partir de su movimiento. Finalmente, esta electricidad es recibida por el osciloscopio y graficada: el eje vertical representa la intensidad, es decir, cuanto más alejada del centro está la curva, significa que la perturbación que recibió el micrófono fue más intensa; el eje horizontal representa el tiempo, dado que a medida que nuevas perturbaciones llegan al micrófono, el gráfico se desplaza para dejar lugar para graficarlas.

Esta misma representación se utiliza normalmente cuando queremos graficar un sonido, como vemos a continuación:



Ejemplo de representación de una onda de sonido

El eje horizontal indica el paso del tiempo. Las vibraciones son representadas por la alternancia entre ascensos y descensos que se ven en el gráfico, siendo más intensas cuanto más alejadas del centro estén.

Ahora que comprendimos la visualización del fenómeno, continuemos al respecto de la relación entre el sonido y cómo se ve la onda que lo produce.

El primer fenómeno que habíamos observado era que, a sonidos más intensos, correspondían dibujos más “altos”: dijimos que, cuanto más intensas eran las perturbaciones en el aire, más alejada del eje horizontal estaba la curva del gráfico. Luego, a mayor intensidad de sonido, observamos gráficos que alcanzan valores mayores en la dimensión vertical. A esta característica de la onda le decimos **amplitud**.

Continuemos ahora reflexionando a propósito del tono: vimos en el osciloscopio que a los sonidos agudos les correspondían dibujos “más angostos”, donde las repeticiones aparecían más juntas en el eje horizontal. Si recordamos que este eje representa el tiempo, esto indica que, para una misma duración, un sonido agudo está formado por más perturbaciones que uno grave. O dicho de otro modo, las perturbaciones se suceden más

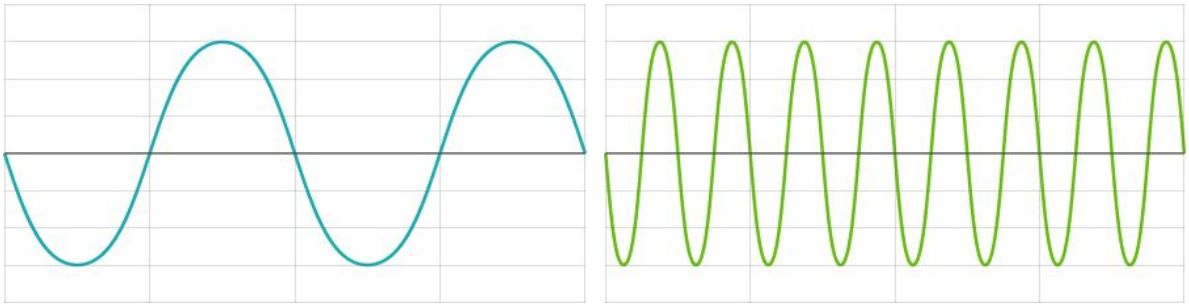
rápidamente en los sonidos agudos que en los graves. A cada una de estas perturbaciones “completas” que se repiten periódicamente les decimos oscilaciones o **ciclos**.

¿Por qué cada cuerda de la guitarra suena diferente de las demás?

Debido a las diferencias en su grosor y su tensión, algunas cuerdas vibran más rápidamente que otras. Si una cuerda vibra más rápidamente, las perturbaciones en el aire se producen a mayor velocidad y, como vimos, en nuestro gráfico las alternancias aparecen más juntas (pues ha ocurrido menos tiempo entre ellas, que es lo que representa la distancia en el eje horizontal). Esto hace que percibamos que unas cuerdas suenen más agudas que otras.

A esta característica de las ondas, de tener sus alteraciones más juntas o más separadas en el tiempo (y, por lo tanto, tener gráficos “más anchos” o “más angostos”), le llamamos **frecuencia**. Recopilando lo que comentamos hasta ahora y observamos en el osciloscopio, podemos concluir que los sonidos más graves tienen frecuencias más bajas que los sonidos más agudos.

Para pasar en limpio este concepto, podemos mostrar dos ondas con diferentes frecuencias. A modo de ejemplo, la siguiente imagen muestra comparativamente una onda correspondiente a un sonido más grave (y, por lo tanto, de menor frecuencia) y otra a un sonido más agudo y, por lo tanto, de mayor frecuencia. De hecho, la segunda onda tiene una frecuencia exactamente cuatro veces mayor que la primera. Además, señalamos los ciclos para dejar en claro a qué nos estamos refiriendo y hacer énfasis en que incluye tanto la perturbación “hacia arriba” como “hacia abajo”.

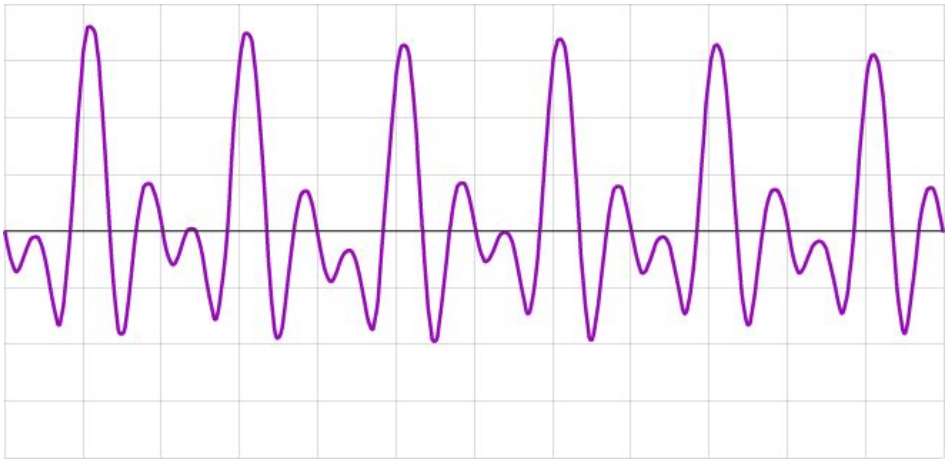


Sonido grave **Sonido agudo**

La forma más común de expresar la frecuencia de una onda es dando la cantidad de ciclos completos que se producen en un segundo. Esto puede calcularse contando la cantidad de crestas o puntos altos (o de valles o puntos bajos) que hay en ese período de tiempo. Por ejemplo, si los gráficos dados como ejemplo representaran un segundo completo de sonido, la onda de la izquierda tendría una frecuencia de dos ciclos por segundo, lo que usualmente

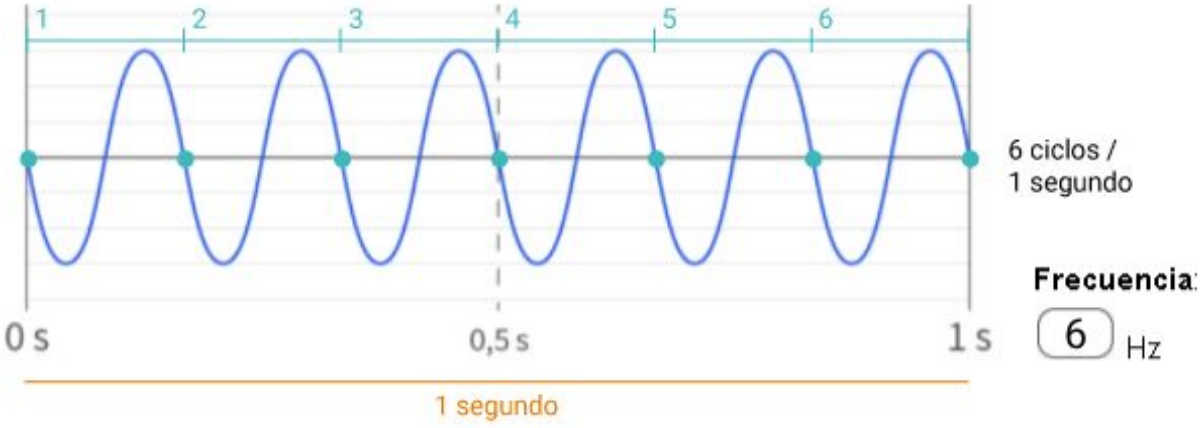
se expresa diciendo que la frecuencia es de dos **hercios** o 2 Hz. En cambio, la onda de la derecha tendría una frecuencia de 8 Hz.

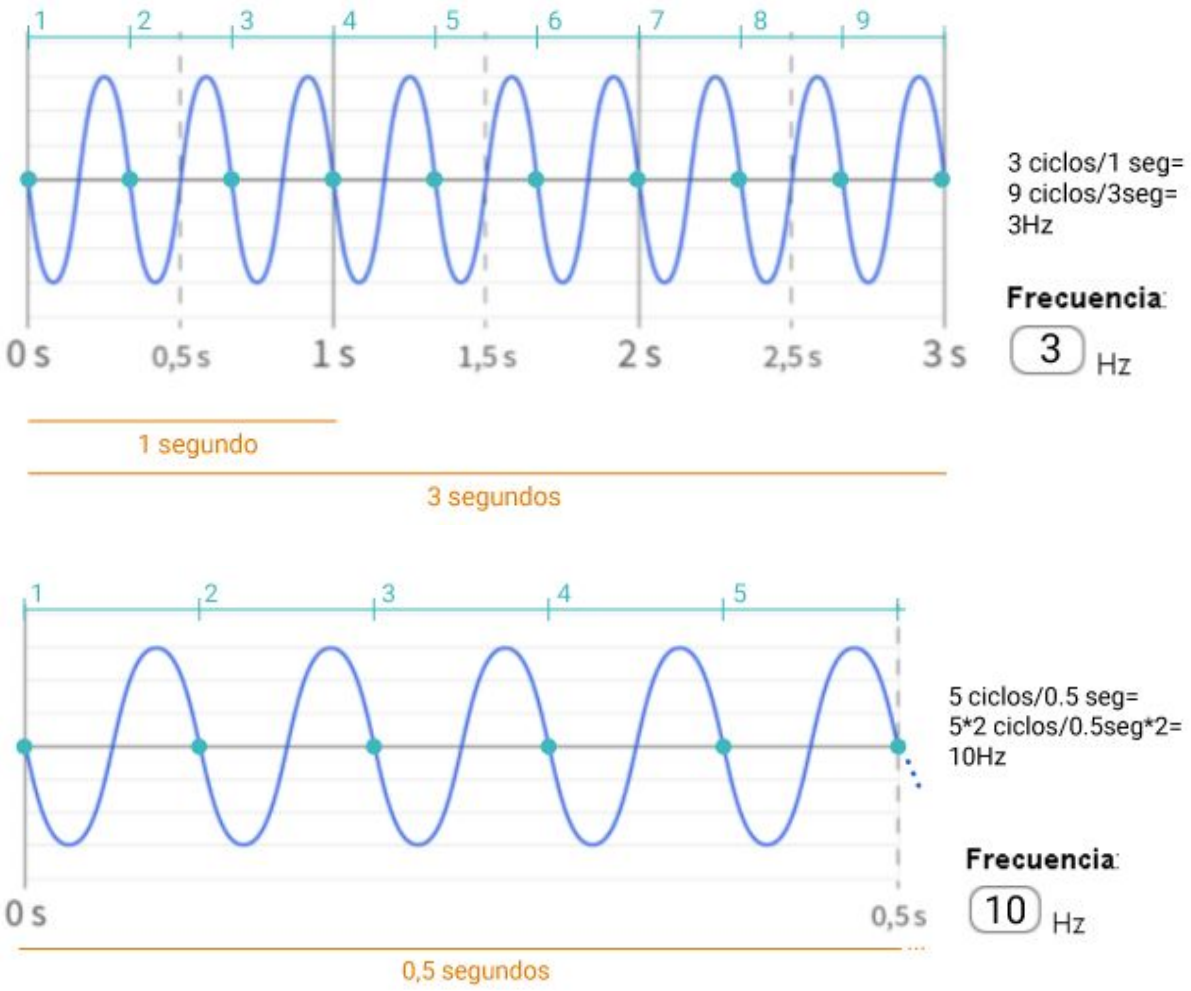
Cabe destacar que, en realidad, las ondas asociadas a los sonidos que escuchamos cotidianamente son más complejas que los ejemplos anteriores, como vimos en el osciloscopio al principio de la actividad. Esto se debe a que, además de tener una frecuencia principal, contienen varias otras frecuencias secundarias sonando simultáneamente. Estas frecuencias secundarias, llamadas **armónicos**, son las que le otorgan la característica propia a cada sonido, de manera que podamos distinguir sonidos de la misma frecuencia tocados en una guitarra, en un piano o por una voz humana.



Ejemplo de onda producida por una persona hablando

Para terminar esta parte, indicamos a los estudiantes que resuelvan la primera actividad de la ficha, donde se muestran tres gráficos de ondas y se pide que determinen su frecuencia en hercios. Es importante que tengan en cuenta qué porción del gráfico representa un segundo. Así, el primero de los gráficos representa una onda de 6 Hz, el segundo una de 3 Hz, y el tercero, de 10 Hz.





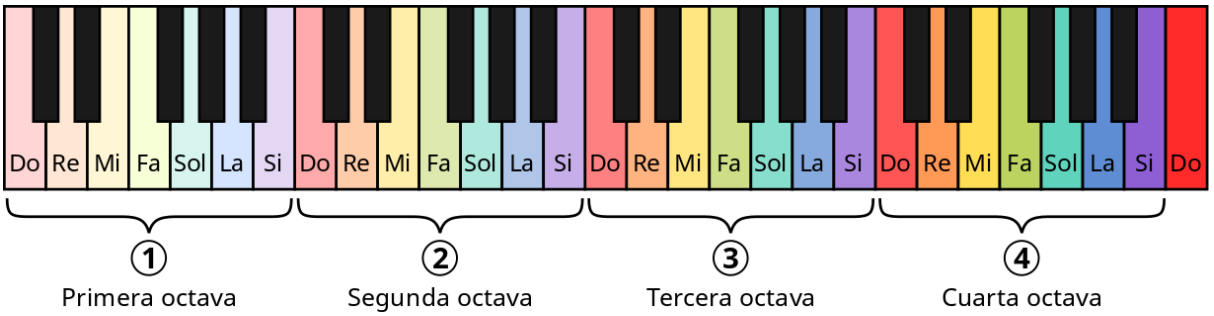
Luego de que resuelvan el ejercicio, ponemos en común los resultados y los invitamos a preguntarse cómo creen que se escucharán esos sonidos. Les contamos que las frecuencias de esas ondas son demasiado bajas para poder ser oídas por el ser humano. La mayoría de las personas podemos escuchar sonidos cuyas frecuencias están entre los 20 Hz y los 20.000 Hz.

Por último, preguntamos si saben de qué manera se distinguen en la música los sonidos de diferentes frecuencias. La idea es recuperar el concepto de **nota musical**. Una nota musical es un sonido de una frecuencia determinada, al que se le asoció un nombre particular que permite referirse a él de manera sencilla. Las notas musicales son centrales a la hora de describir **melodías**: una melodía es una sucesión de sonidos de distintas frecuencias, es decir, de notas musicales, organizados en el tiempo de forma tal que tengan cierto ritmo.

Para presentar lo que sigue, es ideal que tengamos a nuestra disposición un piano o teclado musical. De no contar con uno físico, existen infinidad de sitios web o aplicaciones móviles que simulan pianos y nos permitirán ilustrar las mismas ideas.

Las notas musicales que se utilizan en la música occidental pueden verse en las teclas de un piano, ya que a cada tecla le corresponde una nota distinta. Por simplicidad, en este proyecto trabajaremos solamente con las notas que corresponden a las teclas blancas de un piano. Estas notas reciben siete nombres distintos, que forman un patrón que se repite a

lo largo del teclado: *do*, *re*, *mi*, *fa*, *sol*, *la* y *si*. Es decir, existen varias notas que reciben el nombre *do*, varias que reciben el nombre *re*, y así sucesivamente. Para distinguirlas suelen usarse números, y así en un piano pueden aparecer las notas do_1, do_2, do_3, do_4, etc. A cada una de las secuencias de siete notas que van de *do* a *si* se la conoce como **octava**.



Notas de un teclado musical pequeño, de 49 teclas, que contiene cuatro octavas completas.

Podemos reconocer fácilmente las teclas que corresponden a un *do* en un teclado musical, porque son aquellas que no tienen una tecla negra a la izquierda y tienen dos teclas negras a la derecha. A partir de allí es sencillo encontrar el resto de las notas de la escala musical.

Existen teclados con más y con menos octavas, pero en todos ellos el *do* que se encuentra más cerca del centro se conoce como ***do* central**, y es el que se denota usualmente como do_3. Para terminar esta actividad, los estudiantes deberán investigar y averiguar cuáles son las frecuencias de las notas de la octava correspondiente al *do* central, completando la tabla que se encuentra en la ficha. Los valores correctos son los que se muestran a continuación.

Nota musical	Frecuencia
do_3	261,626 Hz
re_3	293,665 Hz
mi_3	329,628 Hz
fa_3	349,228 Hz
sol_3	391,995 Hz
la_3	440 Hz
si_3	493,883 Hz

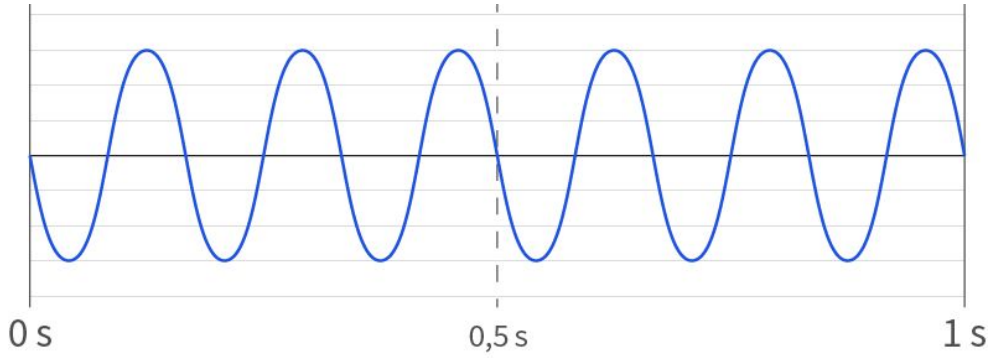
Cierre

En esta actividad nos aproximamos a la naturaleza del sonido tal y como lo escuchamos, pero en adelante, nos interesará trabajar con sonidos representados en una computadora. Para ir preparando el terreno, cerramos esta actividad con una reflexión acerca de maneras de almacenar sonido, pidiendo a los estudiantes que mencionen medios que conozcan para hacerlo: archivos de computadora (posiblemente transmitidos a través de internet), CDs, discos de vinilo, cintas magnéticas o *cassettes*, etc. ¿Qué similitudes y diferencias hay entre estos medios? ¿Cómo será que se representa en ellos una onda de sonido?

Ficha para el estudiante

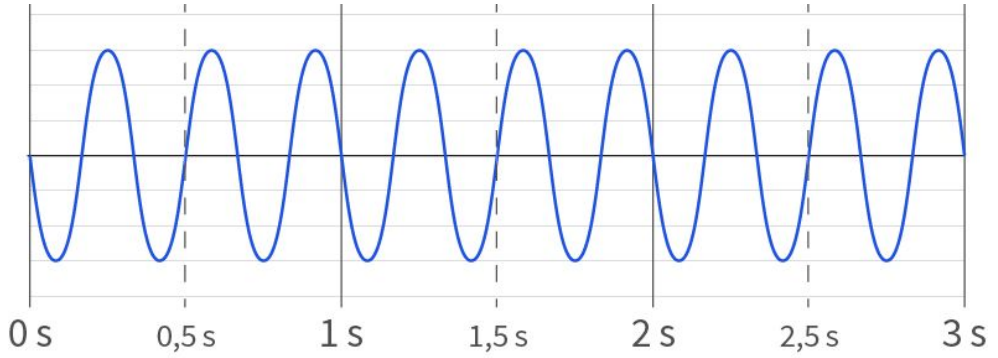
1) ¿Cuál es la frecuencia de las siguientes ondas de sonido?

a)



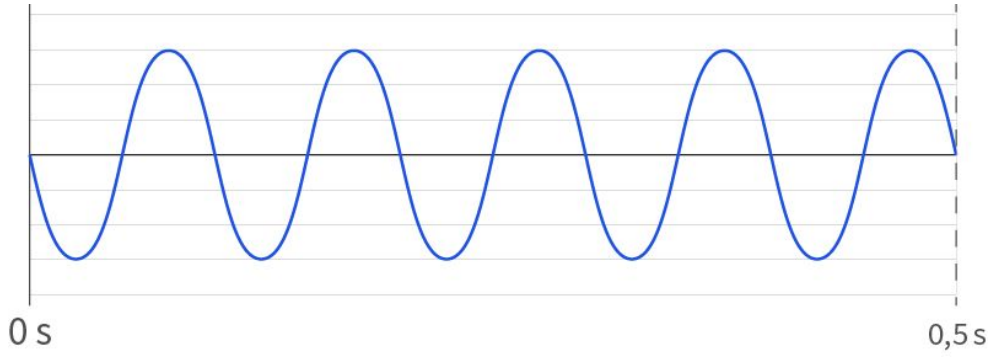
Frecuencia: ☐ Hz

b)



Frecuencia: ☐ Hz

c)



Frecuencia: ☐ Hz

2) Averiguá cuál es la frecuencia de las siguientes notas musicales y completá la tabla a continuación.

Nota musical	Frecuencia
do_3	
re_3	
mi_3	

fa_3	
sol_3	
la_3	440 Hz
si_3	

Actividad 1B | SD1

¿Cómo se representa el sonido?

Objetivos

- Conocer una representación digital del sonido y el concepto de tasa de muestreo.
- Reconocer la importancia de elegir una tasa de muestreo adecuada.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Materiales / Ficha

Desarrollo

Ahora que conocemos las características fundamentales del sonido y podemos representarlo y analizarlo de manera gráfica, nos resta conocer cómo se representa en una computadora. Si conocemos esta representación, podremos generar sonido simplemente generando los números adecuados.

Como ya vimos anteriormente, toda la información que se almacena en una computadora debe ser representada como una secuencia de números; el sonido no es una excepción. Para poder hacer esto con el audio se realiza un procedimiento conocido como **discretización**.

¿Cuánta información hay en un segundo de sonido?

Vimos anteriormente que el sonido es un fenómeno físico que se produce a lo largo del tiempo, y, de hecho, graficamos su intensidad en un eje temporal. Si quisiéramos guardar el sonido (es decir, representar el sonido de alguna manera en algún medio para ser capaces de reproducirlo después), podríamos simplemente anotar el valor de la intensidad de la perturbación que observamos en el gráfico para cada instante de tiempo.
¿Pero qué momentos deberíamos observar? ¿El instante 0 y 1s? En ese caso, no estaríamos observando el fenómeno después de 0,5 segundos. Y aún si anotáramos ese valor, estaríamos descartando la información para 0,25 y 0,75 segundos. Sin embargo, entre estos cuatro puntos, hay muchísimos que no estamos considerando. De hecho, hay infinitos. Es por esto que decimos que el sonido (el tiempo, en realidad) es un fenómeno **continuo**. En cambio, a los fenómenos que ocurren como eventos aislados (y, por lo tanto, entre dos eventos distintos, no hay uno diferente), los llamamos **discretos**.

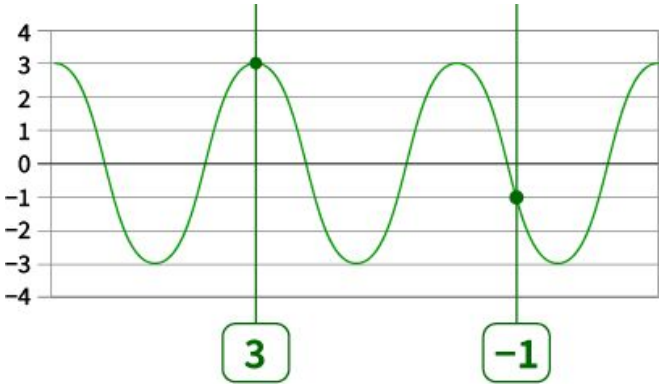
A la hora de almacenar una onda de sonido en la computadora se hace algo similar, dado que no podríamos registrar los infinitos instantes de tiempo en los que ocurre un sonido: nos limitamos a tomar algunos momentos de la misma y guardar la información correspondiente

a ellos, descartando los demás. Cada uno de estos instantes puntuales se representará luego como un número, y diremos que se trata de una **muestra**.

En esta actividad buscaremos que los estudiantes se aproximen por indagación al proceso de discretización del sonido, descubriendo en el camino algunos de los conceptos y de las dificultades centrales que subyacen al mismo. Para comenzar, repartiremos entre ellos copias de las ondas de sonido proporcionadas en los materiales.

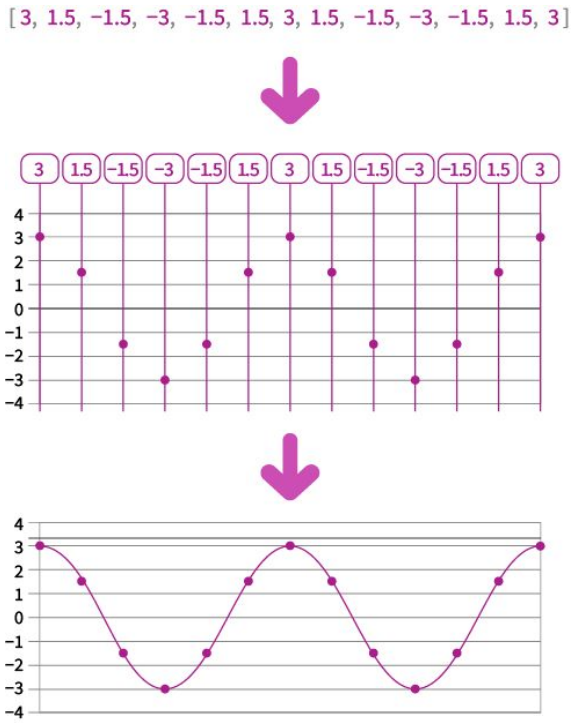
A continuación pedimos que se agrupen en parejas. Es importante que ninguno de los estudiantes pueda ver qué onda de sonido le tocó a su compañero, ya que el objetivo es que cada uno de ellos logren comunicar su onda al otro. Para hacerlo podrán usar, como único recurso, una secuencia de números.

Antes de comenzar, a modo de ayuda, les indicaremos cómo pueden hacer para representar mediante números ciertos instantes de la onda de sonido. La idea es que habiendo elegido un momento, es decir, un punto en el eje horizontal, tracen una línea vertical y se fijen a qué altura se cruza con la onda, mirando los números ubicados en el eje vertical. No es relevante si expresan la altura usando números decimales o la aproximan con el número entero más cercano. A continuación puede observarse un ejemplo.



Ejemplo de dos puntos discretizados en una onda de sonido

También les mostramos cómo se puede realizar el proceso inverso: si tenemos una secuencia de números, podemos reconstruir la onda a la cual pertenecen.

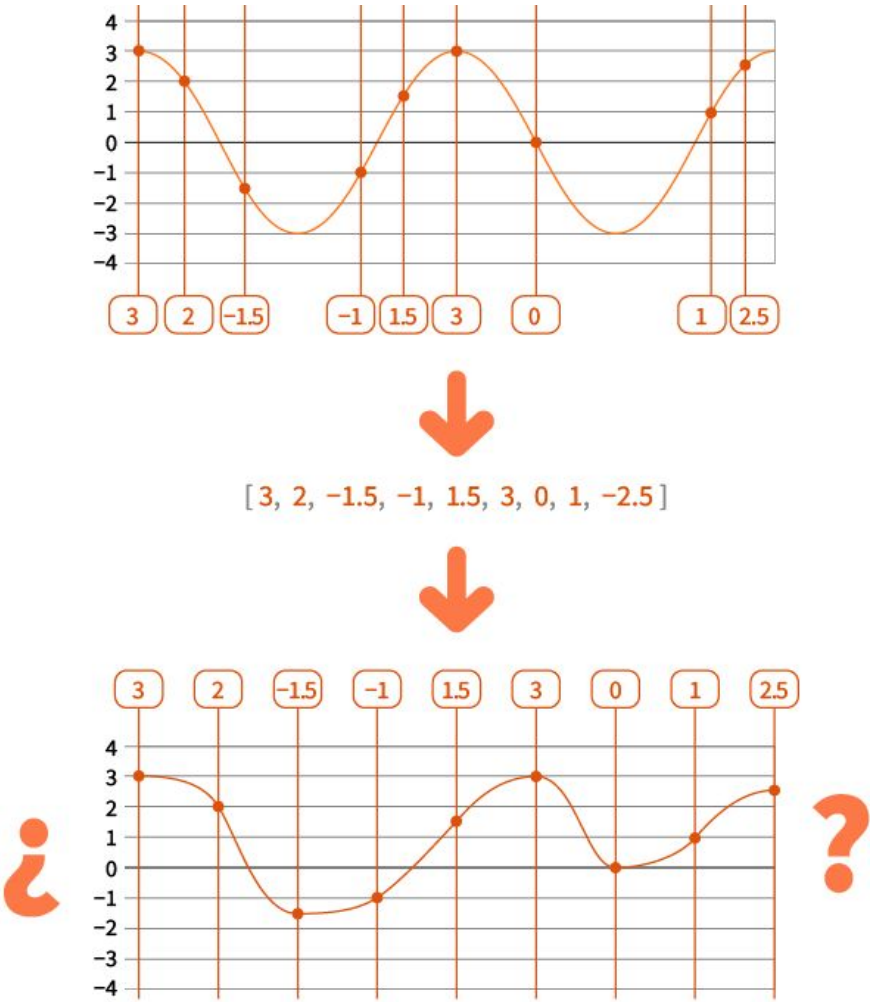


Reconstrucción de una onda de sonido a partir de su discretización en una secuencia de números

La idea es que se trate de una actividad de exploración y experimentación; lo más importante no es que logren completar el objetivo, sino que se encuentren con varios de los inconvenientes que surgen al querer comunicar una onda tomando solamente algunas muestras de la misma. Si vemos que los estudiantes están teniendo dificultades, podemos brindarles más ayuda, ya sea a todo el curso o puntualmente a algunas de las parejas. Después de dar a los estudiantes un tiempo prudencial para que se enfrenten al problema, haremos una puesta en común donde podremos discutir juntos estas dificultades.

La solución a la que queremos llegar es que la onda sea discretizada y a partir de la secuencia de números obtenida, puede ser reconstruida. A continuación se muestran algunos intentos fallidos que pueden surgir al tratar de llegar a una solución.

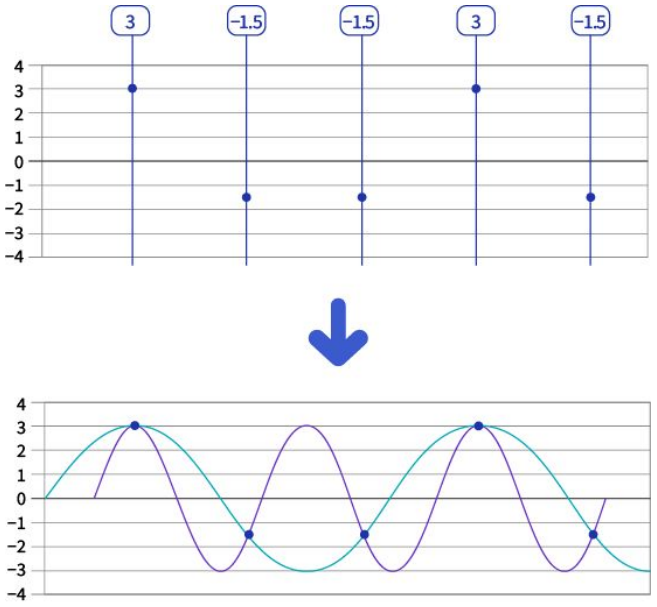
- Un aspecto fundamental es que las muestras sean tomadas a intervalos regulares, es decir, siempre con la misma separación. Esto es fundamental para que la onda pueda ser reconstruida. Si la separación entre muestras varía a lo largo de la onda, la persona que reciba la secuencia de muestras no sabrá cómo tiene que disponer los puntos para reconstruir la onda. Si intenta, por ejemplo, suponer que las muestras están separadas por intervalos regulares, obtendrá un resultado incorrecto al reconstruir la onda.



Onda discretizada incorrectamente debido a que las muestras se tomaron a intervalos irregulares

- Una de las decisiones que deben tomarse a la hora de discretizar una onda de sonido es cuántas muestras deben tomarse en un determinado período de tiempo, lo cual se conoce como **tasa de muestreo**[16]. Usualmente la tasa de muestreo se expresa a través de la cantidad de muestras que se toman por segundo. Cuando la tasa de muestreo es menor, tenemos cada vez menos información acerca del sonido. Si es demasiado baja, podemos tener problemas para reconstruir el sonido original.

[16] A la tasa de muestreo también se le suele decir *frecuencia de muestreo*. Elegimos no utilizar esta terminología para evitar confusiones con la frecuencia del sonido.



Discretización que puede ser reconstruida de dos formas distintas debido a su baja tasa de muestreo. Con la información disponible, es imposible saber cuál de las dos era la onda original.

Es importante que los estudiantes comprendan la importancia del concepto de tasa de muestreo. Si las dificultades surgidas durante la actividad anterior no fueran suficientes para exponer este hecho, podemos profundizar utilizando el ejercicio que se presenta en la ficha. Allí se encontrarán con dos ondas de sonido, y con cuatro secuencias de números distintas. Cada una de las secuencias tendrá una cantidad de muestras menor que la anterior. Les diremos que cada secuencia de muestras corresponde a una de las dos ondas de sonido, y les pediremos que intenten decidir de cuál de ellas se trata, para lo cual deberán reconstruir la onda de sonido a partir de las muestras. En caso de que se encuentren con algún problema para hacerlo, tendrán que explicar por qué.

Al realizar la tarea comprobarán no solo que la dificultad aumenta a medida que la tasa de muestreo disminuye, sino que en el último caso, resulta imposible decidir a cuál de las dos ondas corresponde la versión discretizada. De las tres versiones discretizadas brindadas, la primera corresponde a la onda 2 y la segunda a la onda 1, pero en el caso de la tercera, un ejemplo muy similar al recién presentado, es imposible determinar si se trata de la onda 1 o de la onda 2.

Una pregunta que puede surgir es cómo se escucha un sonido grabado con una tasa de muestreo demasiado baja; esto lo experimentaremos en breve, cuando comencemos a trabajar con las actividades de programación del proyecto. Por último, discutimos acerca de qué tasa de muestreo será necesaria para poder representar correctamente los sonidos que podemos escuchar los seres humanos.

Existe un resultado matemático[17] que dice que la cantidad de muestras a tomar por segundo debe ser, por lo menos, el doble de la frecuencia de la onda. Recordemos que las personas podemos escuchar sonidos de hasta aproximadamente 20.000 Hz; la tasa de muestreo que se utiliza, por ejemplo, en los CDs de audio y muchos estándares profesionales es de

[17] Conocido como el teorema de muestreo de Nyquist-Shannon.

44.100 Hz, ligeramente más que el doble. Esto asegura que se puedan capturar correctamente todas las frecuencias audibles. En definitiva, si bien en el proceso de discretización estamos descartando información (pues solo conservamos una cantidad finita de muestras), si elegimos una tasa de muestreo apropiada, es posible reconstruir el sonido sin diferencias perceptibles (recordemos, que, en todo caso, estaremos perdiendo frecuencias que no son audibles por el ser humano).

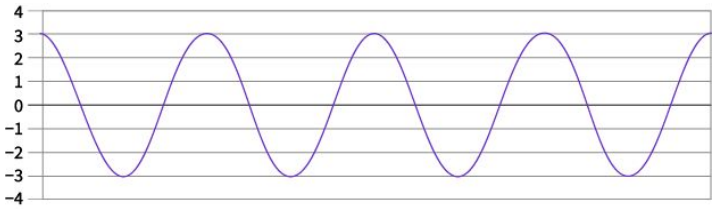
Cierre

Vimos en la actividad que podemos representar una onda de sonido simplemente a partir de una registrar algunos valores de la onda en instantes determinados. Sin emabargo, dado que hay que almacenar las muestras que se registraron (que suelen ser más de 40.000 por segundo), el espacio de almacenamiento requerido puede ser muy grande. Como ya vimos en el capítulo de representación de la información, es posible ahorrar espacio de almacenamiento usando formatos de audio **comprimidos**. De la misma manera que con las imágenes, existen formatos con compresión sin pérdida[18] (como FLAC) a partir de los cuales se puede reconstruir el sonido original exactamente como fue comprimido. Por otro lado, también existen formatos con compresión con pérdida (MP3, OGG), muy popular para distribuir audio por Internet. Estos últimos producen archivos mucho más pequeños que los primeros, pero el sonido obtenido ya no es exactamente el mismo que el original sino que será más o menos parecido, dependiendo de cuánto espacio de almacenamiento estemos dispuestos a dedicarle. Idealmente, de todas maneras, se puede ahorrar mucho espacio y (casi) no percibir la diferencia con el original.

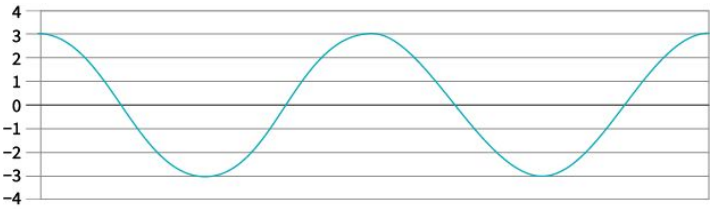
[18] Podemos preguntarles a los estudiantes si se les ocurre alguna manera de comprimir sonido sin pérdida. Para esto, recordamos que lo que tenemos que comprimir es una secuencia de números y tratamos de relacionarlo con el método RLE que vieron para comprimir imágenes. Podemos pensar, por ejemplo, cómo se representará un segundo de silencio: probablemente sean 44100 muestras de valor 0.

Ficha para el estudiante

Mirá atentamente las siguientes dos ondas de sonido.



Onda 1



Onda 2

¿A cuál de las dos ondas anteriores corresponde cada una de estas versiones discretizadas? Si tenés problemas para identificar alguna de ellas, explicá brevemente por qué.

a) [3, 1.5, −1.5, −3, −1.5, 1.5, 3, 1.5, −1.5, −3, −1.5, 1.5, 3]

b) [3, 0 , −3, 0, 3, 0, −3, 0, 3, 0 , −3, 0, 3, 0, −3, 0, 3]

c) [3, −1.5, −1.5, 3, −1.5, −1.5, 3]

Actividad 2 | SD1

Manipulamos sonido

Objetivos

- Conocer la representación en Python de archivos de sonido.
- Trabajar en Python con archivos de sonido.
- Aplicar transformaciones sencillas sobre archivos de sonido.
- Afianzar el concepto de *tasa de muestreo*.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras con Python (preferentemente PyCharm Edu) y nuestras bibliotecas de sonido previamente instalados.
- Auriculares.
- Proyector (opcional).

Desarrollo

Primera Parte:

Con esta actividad, damos comienzo a la parte proyectual del capítulo. A partir de aquí, los estudiantes comenzarán a programar un sintetizador de sonido. Para esto, primero deberán extender los conceptos trabajados anteriormente sobre el sonido y su representación a las herramientas de programación que utilizarán en el resto del proyecto. A esto dedicaremos la primera parte de la actividad, en la que los grupos trabajarán con la consola para explorar el formato *.wav*. Luego, les propondremos que programen algunas transformaciones sencillas, para observar los efectos de alterar la frecuencia o la tasa de muestreo.

Abrimos la actividad comentando que, ahora que ya conocemos algunos aspectos fundamentales del sonido y su representación digital, comenzaremos con la tarea de programación del sintetizador. Para empezar esta tarea y continuando lo trabajado en la actividad anterior, decimos que comenzaremos a explorar la manera que utilizaremos para representar sonido en Python.

La primera parte de la actividad consiste en explorar el formato y las funciones disponibles a partir de trabajar directamente sobre la consola interactiva. Para esto, en el archivo .py correspondiente, encontrarán algunos comandos ya escritos y otros para completar. La idea es que vayan ejecutando los comandos en la consola para observar los efectos y experimentar con sus resultados. Indicamos que lo abran en el entorno de programación y observamos los dos primeros:

```
import audioBasico as ab
```

```
help(ab)
```

Como ya vimos en el capítulo anterior, la instrucción **import** nos permite incorporar al programa (o a la consola en la que estemos ejecutando instrucciones) el contenido de una biblioteca. Además, **as** ab nos permite referirnos a esta biblioteca con el nombre abreviado ab cada vez que queramos usar sus funciones. La novedad es que, en la próxima línea, utilizamos el comando help (ayuda). Seleccionamos estas líneas para ejecutarlas en la consola (como veníamos haciendo en las actividades anteriores para probar partes de los programas) y observamos la salida producida por este último comando:

```
>>> help(ab)
Help on module audioBasico:
NAME
      audioBasico
FILE
      /home/jdabbah/PycharmProjects/cap4/audioBasico.py
FUNCTIONS
      abrirArchivoWav(nombreDelArchivo)

      agregarAlFinal(secuencia, elemento)

      crearSecuenciaVacia()

      fragmento(secuencia, desde, hasta)

      graficar(muestras)

      guardarComoWav(lista_de_intensidades, frecuencia_de_muestreo,
nombre_del_archivo)
```

Vemos que muestra el nombre del módulo (recordemos que en Python se dice *módulo* para referirse a las bibliotecas), el archivo donde está implementado y, lo más importante, las funciones que tiene definidas. Si observamos los nombres, vemos que tenemos funciones para abrir y guardar archivos *wav*, para graficar y para manipular secuencias (crear una secuencia vacía, agregar al final de una secuencia o tomar un fragmento). No esperamos que sea claro para qué vamos a utilizar todo esto en este momento, pero es importante que los grupos sepan que cuentan con estas herramientas para comenzar la exploración. También es un buen momento para dejar en claro que *wav* es un formato de archivos que contienen información de audio. No entraremos en más detalles, pero podemos comentar que es probable que nunca lo hayan visto (y, en cambio, sí se hayan encontrado con archivos con formato *mp3*) debido a que como este formato no tiene compresión y, por ende, los archivos que lo utilizan ocupan mucho espacio, solo se utiliza en contextos con muchas exigencias de

calidad. Sí podemos comentar que, internamente, el formato *wav* no es muy distinto de una secuencia de muestras.

Antes de continuar con la próxima instrucción del archivo Python, preguntamos a los grupos qué se imaginan que sucederá.

```
datosDelEjemplo = ab.abrirArchivoWav('ejemplo_corto_mono.wav')
```

Si es necesario, podemos aprovechar para repasar la forma en la que accedemos a las funciones de las librerías (observamos que hay que anteponer el nombre de la librería, separado por un punto). También podemos observar que `datosDelEjemplo` es una variable en la que se está almacenando el resultado de `ab.abrirArchivoWav`, que, si hace lo que indica su nombre, será el contenido del archivo, representado de alguna manera. Ahora sí, pedimos a los grupos que ejecuten la línea en la consola. Si no se produjo ningún error, no deberían observar nada más, simplemente que la línea se ejecutó. Luego, el próximo paso, es averiguar cómo está representada la información del archivo de sonido en nuestro programa, para poder trabajar con ella o modificarla. Para esto, les sugerimos a los grupos que ejecuten la siguiente instrucción en la consola y observen el resultado:

```
>>> print datosDelEjemplo
(44100, array([ 428, 4056, 9584, ...,   26,   45,    4], dtype=int16))
```

Mono y Estéreo

Muchos de los archivos de audio que utilizamos son *estéreo*, esto quiere decir que en realidad contienen dos pistas o canales de sonido: una para el parlante (o auricular) izquierdo y otra para el derecho. Si cargamos estos archivos con nuestras funciones, observaremos algo similar a lo siguiente:
`(44100, array([[ -47,  -32], [-211, -181], [-311, -111], ..., [ 581, 431], [1183, 1253], [1869, 1779]], dtype=int16))`
En este caso, vemos que la secuencia de muestras, en vez de estar compuesta por números, está compuesta por otras secuencias de dos números cada una, ya que el primer número corresponde a una muestra de un canal y el segundo a una del otro. Para evitar esta complicación, en este proyecto trabajaremos con archivos mono. De todas maneras, las soluciones que proveemos funcionan con ambos tipos de archivo.

Damos un tiempo y después escuchamos las propuestas de interpretación de cada grupo. Apuntamos a hacer los siguientes comentarios:

- Estos datos están compuestos por dos elementos bien diferentes: por un lado, un número (44100) y por otro, un array. No nos detenemos a explicar qué significa esto

último, pero observamos que se ve muy similar a una secuencia[19]. En este caso, podemos agregar que, en Python, cuando tenemos datos compuestos, accedemos a cada uno de sus componentes como si fueran los elementos de una secuencia: `datosDelEjemplo[0]` hace referencia al primer valor (es decir 44100) y `datosDelEjemplo[1]` al segundo (el *array*).

- Nos preguntamos qué significará el número 44100, dado que estamos hablando de representación de sonido. Si hiciera falta, recordamos que lo mencionamos en la actividad anterior, para observar que debe ser la tasa de muestreo pues se corresponde con un estándar muy utilizado (el de grabación de CD).
- Finalmente, observamos que el resto de la información está presentado como si fuera una secuencia de números. Otra vez, recordamos de la actividad anterior que una manera de representar un sonido es como una secuencia de muestras. Luego, es de esperar que este *array* contenga estos valores. De hecho, podemos agregar que podemos operar con él de la misma manera que operábamos con las secuencias.

Pedimos, entonces, que los grupos agreguen a sus archivos las definiciones de las siguientes funciones:

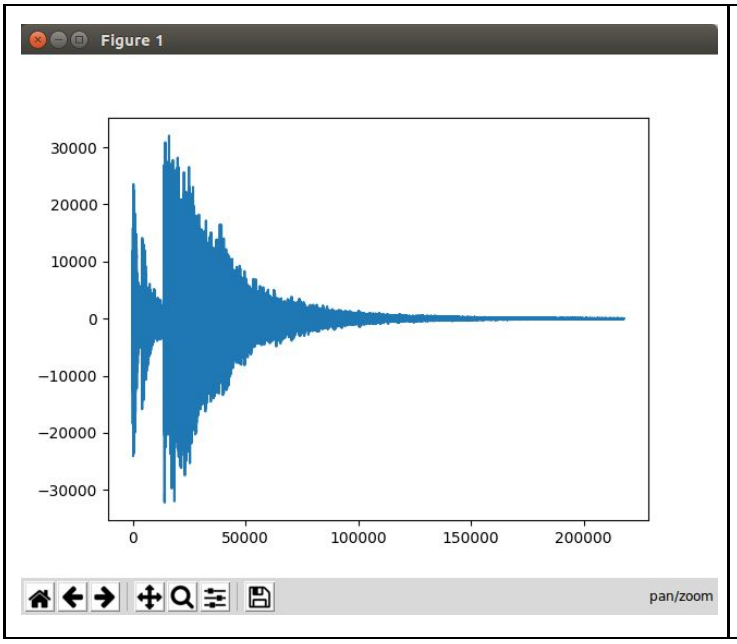
```
def obtenerTasaDeMuestreo(datosDeWav):
    return datosDeWav[0]
```

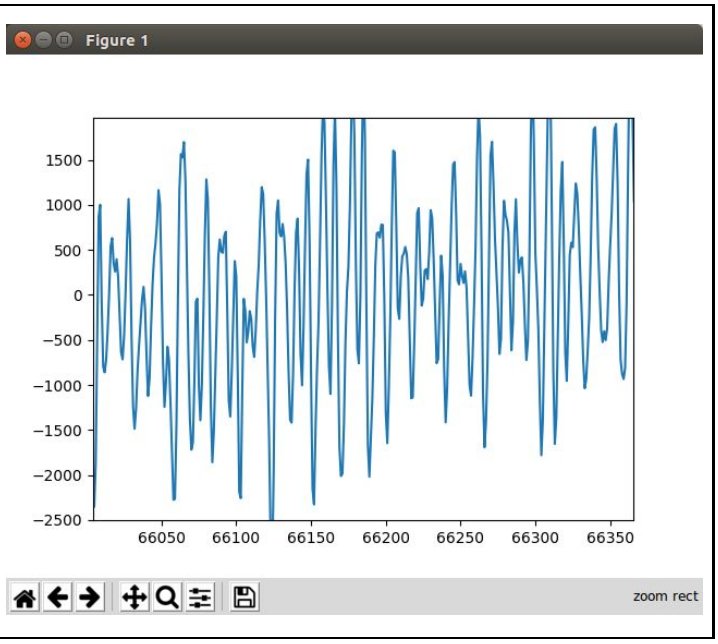
```
def obtenerMuestras(datosDeWav):
    return datosDeWav[1]
```

Ahora, la siguiente pregunta es cómo podemos observar mejor la secuencia de muestras. Esperamos que algún grupo recuerde que, entre las funciones provistas por la biblioteca `audioBasico`, existe una llamada `graficar`. Damos tiempo a los grupos para que la utilicen y estamos atentos a las dificultades. Sobre todo, es probable que la apliquen directamente sobre los datos leídos del archivo, lo que producirá un error y ningún gráfico, pues la función espera una secuencia de valores para graficar, mientras que los datos tal cual fueron leídos están compuestos, por un lado, por la tasa de muestreo pero, además, por la secuencia de muestras (que es lo que nos interesa en este punto). Otra dificultad posible, que dependerá del largo del archivo elegido y de la potencia de cada computadora, es que si la secuencia contiene demasiadas muestras, confeccionar y manipular el gráfico puede volverse tedioso o, incluso, inviable. Para eso, proveemos la función `fragmento`, que nos permite tomar únicamente un fragmento para después graficar.

```
muestrasDelEjemplo = obtenerMuestras(datosDelEjemplo)
ab.graficar(muestrasDelEjemplo)
```

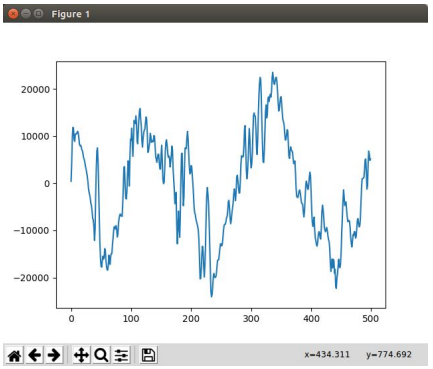
[19] En esencia, es una secuencia, pero, internamente, está construida de manera diferente que las listas de Python. Por eso, además, aclara dtype=int16 que indica que está preparada para contener únicamente números enteros de 16 bits. Estas diferencias se deben a motivos de rendimiento.





Observamos que los gráficos de las muestras son similares a los que observamos en las primeras actividades, cuando investigábamos la representación del sonido. Esta herramienta de visualización, al hacer clic sobre la lupa permite trazar un rectángulo y visualizar los datos únicamente de esa región (lo que, en general, llamamos *hacer zoom*). Sin embargo, si esto no fuera posible por limitaciones técnicas, podemos ejecutar el siguiente código para tomar, por ejemplo, solo las primeras 500 muestras.

```
muestrasDelEjemplo =
obtenerMuestras(datosDelEjemplo)
primerasMuestras =
ab.fragmento(muestrasDelEjemplo, 0, 500)
ab.graficar(primerasMuestras)
```



Segunda parte:

En esta segunda parte, trabajaremos manipulando la secuencia de muestras y la tasa de muestreo. Esta parte está orientada mediante preguntas / consignas del tipo "¿Qué sucederá sí…?", para las que los grupos deberán confeccionar programas que efectivamente realicen lo que se especula en la pregunta y luego los apliquen sobre archivos de audio para observar sus efectos.

Primera pregunta: ¿Qué pasa si duplicamos cada una de las muestras y dejamos la misma tasa de muestreo?

En este momento, planteamos la pregunta, damos a los grupos unos minutos para que piensen y escuchamos sus hipótesis. Es fundamental que no digamos si son acertadas o no, dado que deberán hacer un un programa para averiguarlo. Les decimos, entonces, que comiencen con esta tarea.

Damos un tiempo para que los grupos piensen la estrategia y qué funciones de la biblioteca o que ya hayan definido podrían usar para resolver cada paso. Además, pueden suponer que cuentan con la función `duplicarElementos(secuencia)` que, dada una secuencia, produce otra con los elementos duplicados. Por ejemplo, `duplicarElementos([1, 2, 3])` debe producir la secuencia `[1, 1, 2, 2, 3, 3]`. A continuación presentamos una estrategia posible:

Paso de la estrategia	Función que podemos utilizar para realizarlo
1. Abrir el archivo wav y almacenar los datos en una variable.	`ab.abrirArchivoWav`
2. Extraer de estos datos la secuencia de muestras y la tasa de muestreo.	`obtenerMuestras` y `obtenerTasaDeMuestreo`
3. Duplicar la secuencia de muestras obtenidas.	`duplicarElementos`
4. Generar un archivo wav con las muestras duplicadas y la tasa de muestreo original	`ab.guardarComoWav`

`ab.abrirArchivoWav` nos permite acceder en Python al contenido de un archivo wav. Sin embargo, no es suficiente, pues necesitamos trabajar con la secuencia de muestras y con la tasa de muestreo por separado. Para eso, utilizaremos las funciones `obtenerMuestras` y `obtenerTasaDeMuestreo`. Luego, aplicando `duplicarElementos` sobre las muestras que conseguimos en el paso anterior, obtenemos la secuencia con los elementos duplicados. Finalmente, para poder escuchar el sonido en un reproductor externo, generamos un nuevo archivo wav. Para esto, utilizamos la función `ab.guardarComoWav` que necesita, por un lado, la secuencia de muestras y por otro, la tasa de muestreo a utilizar.

Ahora sí, dado que todas las funciones que necesitan ya están resueltas excepto `duplicarElementos`, les indicamos que la implementen. En este momento, podemos recordar que teníamos las funciones `ab.crearSecuenciaVacia()` y `ab.agregarAlFinal(secuencia, elemento)` para crear secuencias y agregarle elementos. Dejamos que los grupos trabajen solos, mientras observamos sus avances y estamos atentos a sus necesidades.

La estrategia para esta función es crear una lista nueva (y, por lo tanto, vacía) e ir construyéndola a partir de la anterior. Esperamos que esto sea claro al observar la implementación incompleta que proveemos, pero de no serlo, debemos dejarlo claro con los grupos. Para construir la lista con los duplicados, deberán observar que necesitan un recorrido `for` sobre la secuencia original para ir accediendo una a una a las muestras. Luego, lo que hay que hacer *para cada* muestra, si queremos obtener una secuencia en la que todos los elementos aparezcan dos veces, es agregarla dos veces al resultado. Para

esto, podemos usar dos veces la función `ab.agregarAlFinal`, como vemos en la solución que aparece a continuación:

```
def duplicarElementos(secuencia):
    #Creamos una secuencia vacía para construir el resultado
    nuevaSecuencia = ab.crearSecuenciaVacia()

    #Recorremos todos los elementos de la secuencia
    for elemento in secuencia:
        #Para cada un de ellos, lo agregamos dos veces al resultado
        ab.agregarAlFinal(nuevaSecuencia, elemento)
        ab.agregarAlFinal(nuevaSecuencia, elemento)

    #Devolvemos la secuencia que construimos
    return nuevaSecuencia
```

Comenzamos creando una secuencia vacía sobre la que construiremos el resultado. Luego, recorremos todos los elementos de la secuencia original y, en este caso, agregamos cada uno a la secuencia creada. Finalmente, devolvemos esta secuencia.

Como metodología de trabajo, al igual que en actividades anteriores, sugerimos que vayan probando las implementaciones a medida que las van completando. Para esto, pueden seleccionar el texto completo de la definición y, haciendo clic derecho, elegir la opción "*Execute selection in console*" (Ejecutar lo seleccionado en la consola). De esta manera, se incorpora (o actualiza) la definición de la función en el intérprete de la consola y pueden probarla directamente en el intérprete. También los invitamos a que prueben con ejemplos sencillos para los que puedan observar rápidamente si la respuesta del programa es correcta o no.

```
>>> def duplicarElementos(secuencia):
...     nuevaSecuencia = ab.crearSecuenciaVacia()
...     for elemento in secuencia:
...         ab.agregarAlFinal(nuevaSecuencia,
elemento)
...         ab.agregarAlFinal(nuevaSecuencia,
elemento)
...     return nuevaSecuencia
...
>>>
```

```
>>> secuenciaDePrueba = [1, 2, 3]
>>> duplicarElementos(
secuenciaDePrueba)
[1, 1, 2, 2, 3, 3]

>>>
```

A la izquierda, vemos la consola luego de seleccionar las definiciones en el código y ejecutarlas. A partir de este momento, y si no se produjeron errores, contamos con las funciones definidas para utilizar en el intérprete. En la segunda columna, observamos una posible prueba: primero creamos la variable `secuenciaDePrueba`, donde almacenamos una secuencia de números muy simple, y luego le aplicamos la función `duplicarElementos`, cuyo resultado se imprime en la línea de abajo. En este caso, vemos que funciona correctamente.

Con esta función definida ya podemos implementar la estrategia que habíamos especificado antes de comenzar a programar. De esta manera, tendremos un programa que procese un archivo de audio duplicando sus muestras y podremos observar qué efecto produce y si se condice con las hipótesis planteadas al comienzo.

```
datosDelEjemplo = ab.abrirArchivoWav('ejemplo_corto_mono.wav')
tasaDeMuestreoDelEjemplo = obtenerTasaDeMuestreo(datosDelEjemplo)
muestrasDelEjemplo = obtenerMuestras(datosDelEjemplo)
muestrasDuplicadas = duplicarElementos(muestrasDelEjemplo)
ab.guardarComoWav(muestrasDuplicadas, tasaDeMuestreoDelEjemplo,
'ejemplo_duplicadas.wav')
```

Traducimos la estrategia a Python utilizando las funciones que habíamos identificado al comienzo. Observamos que, cuando queremos utilizar los datos producidos por una instrucción en otra más adelante (como los datos del archivo, la secuencia de muestras o la tasa de muestreo), los guardamos en una variable con un nombre apropiado para explicitar su contenido.
Observamos que, dado que ya venimos trabajando en el archivo, es probable que ya tengamos definidas algunas de estas variables. En ese caso, no hace falta volverlas a crear, simplemente podemos utilizarlas.

A medida que vayan consiguiendo el nuevo archivo, les indicamos a los grupos que lo escuchen (si es posible, con auriculares, para no adelantarle el resultado a los otros grupos), que observen la diferencia con el original, verifiquen si la hipótesis que habían planteado al comienzo era correcta o no y que propongan una explicación. Cuando todos estén listos, hacemos una puesta en común.

En este caso, observaremos que el nuevo archivo es más grave y más largo. Esto último es, tal vez, lo más evidente: conservamos la tasa de muestreo (es decir, la cantidad de muestras que se deben reproducir por segundo) pero descartamos la mitad de las muestras (es decir, contamos con la mitad del material para reproducir), por lo tanto, la reproducción durará el doble. Para poner esto en evidencia, si hiciera falta, podemos preguntar cómo será la longitud de la nueva secuencia de muestras comparada con la longitud de la secuencia de muestras original.

Para explicar el cambio de tono, deberán recordar que éste estaba determinado por la frecuencia, y que, entonces, debe haber bajado la frecuencia del sonido. Damos unos minutos para que lo piensen y podemos sugerirles que se ayuden por un esquema, en el que se muestrea una onda simple cualquiera.

Observamos que, al duplicarse las muestras, las ondas “se estiran” pues todos los puntos aparecen duplicados. Si recordamos que las ondas “más anchas” eran las de menor frecuencia, podemos ver que, al hacer esta modificación, se reduce la frecuencia. De hecho, en el esquema vemos que la onda a disminuido su frecuencia a la mitad: en el mismo tiempo en que el sonido original completaba dos ciclos, el sonido modificado completa uno solo.

Segunda pregunta: ¿Qué pasa si descartamos una de cada dos muestras y dejamos la misma frecuencia de muestreo?

Al igual que antes, escuchamos hipótesis pero no decimos nada. Simplemente les proponemos a los grupos que escriban un programa que tome un sonido y genere un archivo nuevo descartando las muestras.

La estrategia general para abrir el archivo y modificarlo es idéntica a la que trabajamos en la pregunta anterior, ahora utilizando `descartarPosicionesImpares` (que, dada una secuencia, genera una nueva que contiene únicamente sus posiciones pares) en vez de `duplicarElementos`. Nos concentraremos, entonces, en esta.

Como pista, les podemos indicar que observen el esquema de la función que definieron en la pregunta anterior (crear una secuencia vacía e ir construyendo en ella el resultado, a partir de recorrer la lista), para que identifiquen que la dificultad que aparece en este caso es que no hay que hacer lo mismo con todos los elementos de la secuencia original. Ésta debería ser la motivación para que incluyan una alternativa condicional dentro del recorrido para hacer una u otra cosa (agregar o no agregar el elemento). Para responder la pregunta "¿Cómo sabemos si tenemos que agregar el elemento que estamos mirando?" podemos sugerirles que observen que existe una función `esPar` y que queremos descartar las posiciones impares. A continuación vemos dos soluciones posibles. La segunda no utiliza la función `esPar`, pero puede motivarse a partir de pensar que, en el momento en que se agrega un elemento, ya se sabe que no debe agregarse el próximo y, por lo tanto, podemos guardar esa información en una variable booleana.

```
def descartarPosicionesImpares(secuencia):
    nuevaSecuencia = ab.crearSecuenciaVacia()
    posicion = 0
    for elemento in secuencia:
        if esPar(posicion):
            ab.agregarAlFinal(nuevaSecuencia,
                              elemento)
        posicion = posicion + 1
    return nuevaSecuencia

def esPar(n):
    return n%2 == 0
```

```
def descartarPosicionesImparesBool(secuencia):
    nuevaSecuencia = ab.crearSecuenciaVacia()
    hayQueAgregarElProximo = True
    for elemento in secuencia:
        if hayQueAgregarElProximo:
            ab.agregarAlFinal(nuevaSecuencia,
                              elemento)
            hayQueAgregarElProximo = False
        else:
            hayQueAgregarElProximo = True

    return nuevaSecuencia
```

Dos soluciones distintas para la función pedida. En ambas comenzamos creando una secuencia vacía y utilizamos un recorrido sobre los elementos de la secuencia original para ir agregándolos a la nueva. Sin embargo, la nueva dificultad es que no siempre debemos agregar los elementos, solo uno de cada dos. Para eso, el primer enfoque define la variable `posicion` que representa la posición en la secuencia del elemento que se está considerando en cada ejecución del cuerpo del recorrido (por eso comienza en 0 y en cada paso se incrementa en uno), y, mediante una alternativa condicional, si la posición es par, lo agrega y si no, no. La segunda, utiliza una variable booleana que indica si hay que agregar el próximo elemento. Luego, al ejecutarse

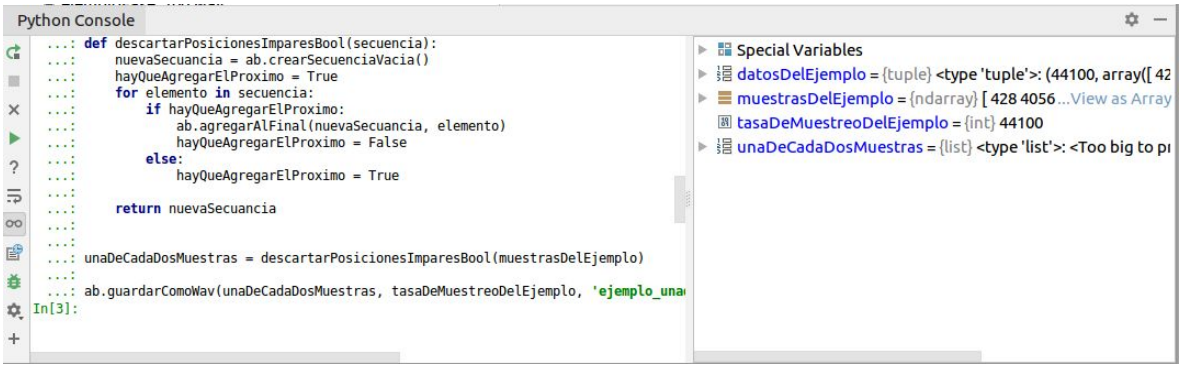
el cuerpo del recorrido, si esta variable es verdadera, agrega el elemento (y establece su valor a falso, pues no hay que agregar el próximo elemento pues se acaba de agregar uno). Si esto no sucede, no se agrega el elemento pero se cambia el valor de la variable a verdadero, pues si este elemento no fue agregado, deberá agregarse el próximo.

De manera muy similar a cómo lo hicimos anteriormente, podemos utilizar las funciones provistas para abrir un archivo, extraer las muestras, descartar las posiciones impares y generar un nuevo archivo con ellas, como vemos a continuación.

```
datosDelEjemplo = ab.abrirArchivoWav('ejemplo_corto_mono.wav')
tasaDeMuestreoDelEjemplo = obtenerTasaDeMuestreo(datosDelEjemplo)
muestrasDelEjemplo = obtenerMuestras(datosDelEjemplo)
unaDeCadaDosMuestras = descartarPosicionesImpares(muestrasDelEjemplo)
ab.guardarComoWav(unaDeCadaDosMuestras, tasaDeMuestreoDelEjemplo,
'ejemplo_unadedos.wav')
```

El esquema general es idéntico a la pregunt anterior. Observamos que la única diferencia es la función `descartarPosicionesImpares` en vez de `duplicarElementos`.

IDE TIP:
A medida que ejecutamos líneas de código en la consola (ya sea seleccionándolas individualmente y ejecutándolas o ejecutando un archivo completo) que definen variables, vemos que éstas aparecen en un recuadro especial a la derecha de la consola.



[Captura de pantalla después de ejecutar lo que hicimos hasta ahora en la Act2]
Podemos observar que, además de sus nombres, podemos ver el valor que tienen o inspeccionar su contenido haciendo clic sobre las flechas.

Al igual que antes, les indicamos a los grupos que escuchen el archivo que obtuvieron, observen las diferencias con el original, contrasten estas diferencias observadas con sus hipótesis y propongan una explicación. No nos interesa que consigan una respuesta cerrada, sino que puedan identificar qué conceptos vistos en este capítulo pueden ser útiles para explicarlos. De todas maneras, es muy probable que debamos orientarlos con preguntas o sugerencias.

La diferencia observable es que el sonido es más agudo y más corto. El argumento para explicar la reducción en la duración es el mismo que para la pregunta anterior, pero al

revés: descartar muestras reduce la cantidad de puntos para reproducir y acabamos de deshacernos de la mitad de las muestras que traía el archivo.

Para explicar el cambio de tono, al igual que antes, deberían comenzar por identificar que lo que cambió del sonido fue su frecuencia. Damos unos minutos para que piensen una explicación, sobre todo, teniendo en cuenta lo que dijimos para la modificación anterior. No nos interesa ir a fondo en la explicación, pero basándonos en lo que ya vimos, podemos convencernos de que, si duplicar las muestras "ensanchaba" la onda, descartar la mitad de las muestras debería "juntarla", lo que significa un aumento de la frecuencia.

Tercera consigna: ¿Qué sucede si, sin modificar las muestras, duplicamos la tasa de muestreo? ¿Y si la reducimos a la mitad?

Al igual que antes, dejamos que los grupos hagan sus especulaciones y las fundamenten, y dejamos la pregunta abierta, al mismo tiempo que les indicamos que hagan programas para experimentar con las dos situaciones.

En este caso, dado que no hay que modificar la secuencia de muestras, sino simplemente el valor de la tasa de muestreo, esperamos que los grupos puedan resolverlo más rápidamente. Como siempre, cuando vayan terminando, les pedimos que escuchen y corroboren o corrijan lo planteado antes de comenzar.

```
#Ya tenemos definidas todas las variables con los datos que
necesitamos.
tasaDeMuestreoDoble = 2*tasaDeMuestreoDelEjemplo
ab.guardarComoWav(muestrasDelEjemplo, tasaDeMuestreoDoble,
'ejemplo_tasadoble.wav')

tasaDeMuestreoMitad = tasaDeMuestreoDelEjemplo / 2
ab.guardarComoWav(muestrasDelEjemplo, tasaDeMuestreoMitad,
'ejemplo_tasamitad.wav')
```

Si ya ejecutamos las soluciones a las consignas anteriores, tendremos definidas las variables `tasaDeMuestreoDelEjemplo` y `muestrasDelEjemplo` que contienen los datos del archivo original. Luego, para generar los archivos modificados, basta con computar las nuevas tasas de muestreo, que se obtienen fácilmente a partir de las anteriores multiplicando y dividiendo, respectivamente, por dos.

Hacemos una puesta en común y observamos que produjimos un efecto similar al de las funciones de las consignas anteriores:

- al duplicar la tasa de muestreo se reduce a la mitad la duración del sonido y se percibe más agudo, y
- al reducir a la mitad la tasa de muestreo, se duplica la duración del sonido y se percibe más grave.

Observamos que, al variar la tasa de muestreo, varía la cantidad de muestras que se reproducen en un segundo. En el primer caso, al reducir la tasa a la mitad, se reproducen cuatro muestras en un segundo, mientras que en el segundo, se reproducen 16. Si comparamos los gráficos todos a la misma escala, vemos que mientras en el primero se producen dos ciclos completos en un segundo, en el segundo se produce uno solo y en el último, cuatro en este mismo intervalo de tiempo. Mirando los gráficos resultantes, podemos ver que en el primer caso disminuye la frecuencia y en el segundo, aumenta.

Para terminar esta actividad, proponemos la última consigna:

> *Cuarta consigna:* ¿Qué sucederá si eliminamos la mitad de las muestras y además, reducimos a la mitad la tasa de muestreo?

Les preguntamos a los grupos qué creen que sucederá. Si hiciera falta, les recordamos que ya experimentamos con cada uno de esos factores por separado, para que observen que estas dos alteraciones tienen efectos contrarios. Luego, ¿no sucederá nada, ya que los efectos de una serán compensados por la otra? Damos tiempo para que experimenten, sugiriéndoles enfáticamente que reutilicen las funciones definidas y las variables ya cargadas en la consola.

```
ab.guardarComoWav(unaDeCadaDosMuestras, tasaDeMuestreoMitad,
'ejemplo_unadedosmitad.wav')
```

Después de ejecutar los programas de las consignas anteriores, deberíamos tener definidas las variables `unaDeCadaDosMuestras`, que contiene la mitad de las muestras y `tasaDeMuestreoMitad` que contiene la mitad del valor de la tasa de muestreo original

Al escuchar el audio resultante, es muy probable que no se perciban diferencias. Luego, planteamos el siguiente interrogante para debatir entre todos. *"Si aplicáramos esta misma transformación sobre el archivo que obtuvimos, obtendríamos un archivo con la cuarta parte de las muestras del original, un cuarto de su tasa de muestreo y que suena exactamente igual. Y si lo hiciéramos una y otra vez, obtendríamos el mismo audio, pero con muchísimas menos muestras. ¿Es esto posible? ¿Por qué?"*

Nos interesa señalar en el debate que esto representaría una ventaja enorme, pues podríamos almacenar audio en formato digital utilizando mucha menos espacio de almacenamiento (pues, recordemos, para almacenar cada muestra necesitamos espacio de almacenamiento para un número), pero que, sin embargo, vimos que los estándares que conocemos (como el CD), no lo hacen. A partir de esta observación, entonces, enlazamos con lo visto en las primeras actividades del capítulo, en las que señalamos que utilizar tasas de muestreo inadecuadas puede producir una representación errónea. Luego, el desafío

consistirá en repetir esta transformación hasta que la diferencia con el original sea perceptible.

La cantidad de veces que haya que reducir la tasa de muestreo para notar una diferencia dependerá del archivo original tomado como ejemplo, de la calidad de los instrumentos de reproducción utilizados y de la agudeza de la percepción de los estudiantes, pero al reducirlo a la cuarta u octava parte ya debería ser evidente.

```
unaDeCadaCuatroMuestras =
descartarPosicionesImpares(unaDeCadaDosMuestras)
tasaDeMuestreoCuarto = tasaDeMuestreoMitad / 2
ab.guardarComoWav(unaDeCadaCuatroMuestras, tasaDeMuestreoCuarto,
'ejemplo_bajarmuestreo4.wav')

unaDeCadaOchoMuestras =
descartarPosicionesImpares(unaDeCadaCuatroMuestras)
tasaDeMuestreoOctavo = tasaDeMuestreoCuarto / 2
ab.guardarComoWav(unaDeCadaOchoMuestras, tasaDeMuestreoOctavo,
'ejemplo_bajarmuestreo8.wav')
```

Dos aplicaciones sucesivas de la misma transformación. En la primera, descartamos la mitad de las muestras de la secuencia que contiene solo la mitad de las muestras del original (`unaDeCadaDosMuestras`) y generamos un archivo con un cuarto de la tasa de muestro que el original. En la segunda, repetimos el proceso sobre las muestras que obtuvimos en la primera parte parte (`unaDeCadaCuatroMuestras`), es decir, descartamos la mitad de la secuencia que contiene un cuarto de las muestras, lo que deja únicamente un octavo de las muestras originales. Generamos un archivo con éstas y una tasa de muestreo equivalente a un octavo del original.

Ahora sí, observamos que en cada transformación obtenemos un sonido que podemos identificar como el mismo, pero que se oye diferente. Podemos decirles a los grupos que escuchen atentamente el audio degradado y que observen que los sonidos agudos del original ya no se escuchan. Si consiguen observar esto, les decimos que pueden explicar lo que ha sucedido retomando lo visto cuando expusimos los valores adecuados para la tasa de muestreo. En ese momento mencionamos el teorema de Nyquist, que establecía que la tasa de muestreo debía ser, por lo menos, dos veces la frecuencia más aguda que quisiéramos representar. Luego, al bajar la tasa de muestreo, estamos perdiendo la capacidad de representar sonidos agudos, que fue lo que observamos con nuestro programa.

Cierre

A modo de conclusión, podemos pasar en limpio la manera en la que representaremos el audio en nuestros programas y cómo debemos trabajar con él. Nuestros sonidos consisten, básicamente, en una secuencia de muestras y una tasa de muestreo. Vimos cómo podemos traba

Actividad 3 | SD1

Generamos sonidos

Objetivos

- Generar archivos de sonidos en Python.
- Conocer el concepto y la utilidad de los casos de test.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras con Python (preferentemente PyCharm Edu) y nuestras bibliotecas de sonido previamente instalados.
- Auriculares.
- Proyector (opcional).

Desarrollo

Al comienzo del capítulo trabajamos sobre los fundamentos de la representación digital del sonido y en la actividad anterior los pusimos en práctica, observando una representación posible en Python. Más aún, hicimos programas que modificaban estos sonidos trabajando sobre su representación interna y observamos sus efectos. En esta actividad, comenzaremos a generar o *sintetizar* sonidos, es decir, a producir archivos de sonido sin ningún archivo de entrada, simplemente escribiendo programas para que generen las muestras que luego se codificarán dentro del archivo *wav*.

Comenzamos la actividad contando que, por primera vez, trabajaremos generando sonidos y no modificando alguno ya existente. Les decimos a los grupos que el plan para esto es escribir programas que generen secuencias de muestras para luego escribir archivos *wav* que se puedan reproducir en la computadora, al igual que hicieron en la actividad anterior.

Dado que hasta ahora nunca escribieron programas que generen secuencias sin otra como referencia, a modo de ejercicio preliminar, podemos proponer que completen la función `generarRepetidos(elemento, repeticiones)`, que recibe un elemento y produce la secuencia que consiste en este elemento repetido tantas veces como se le indica en el segundo parámetro. Por ejemplo, `generarRepetidos(1, 4)` debe producir la secuencia `[1, 1, 1, 1]`. La solución consiste en crear una secuencia vacía y, utilizando una repetición simple, agregar el elemento todas las veces que sean necesarias, como vemos a continuación:

```
def generarRepetidos(elemento, repeticiones):
    repetidos = ab.crearSecuenciaVacia()
    for i in range(repeticiones):
        ab.agregarAlFinal(repetidos, elemento)
```

```
    return repetidos
```

Para continuar, les contamos a los grupos que, antes de empezar a programar, deberán completar lo que en el desarrollo de software se conoce como **casos de test**, es decir, ejemplos de resultados correctos que deberían obtenerse con las funciones pedidas. Luego, al comparar estos ejemplos con los resultados obtenidos por los programas que efectivamente se escribieron, podemos observar si funcionan correctamente o producen resultados incorrectos.

¿Podemos estar seguros de que los programas no tienen errores?

Aún si los *tests* no detectan errores, por muchos que sean, no podemos asegurar que el programa sea correcto; para eso, deberíamos probarlo sobre los infinitos valores posibles de todos sus parámetros, lo que evidentemente es irrealizable. De hecho, existe un resultado teórico de las Ciencias de la Computación que dice que no puede existir un programa capaz de decidir si otro está libre de errores.

Enfatizamos que es una buena práctica elegir y diseñar los casos de test antes de comenzar a programar, para pensar únicamente en cómo debe comportarse el programa (sin pensar en cómo está implementado) y, además, para explorar y comprender mejor el problema antes de avanzar a la etapa de programación.

Para guiarlos en la elaboración de los tests proveemos uno completo a modo de ejemplo, otro a medio completar, para hacer foco en algunos aspectos importantes del problema (cuánto dura un ciclo y cuántas muestras hay que generar en consecuencia) y otro en blanco para que ejerciten la tarea por completo. Observamos que no hace falta que estén planteados sobre valores reales (veremos que utilizaremos frecuencias inaudibles y tasas de muestreo demasiado bajas para un archivo de audio concreto); al contrario, elegiremos valores para los cuales pueda ser fácil conocer el resultado esperado “a mano” sin tener que utilizar el programa. En este mismo sentido, elegimos 1 como valor arbitrario para la amplitud, solo por simplicidad.

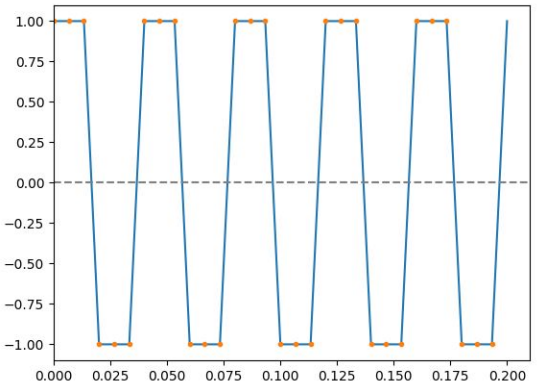
Es importante que aclaremos en este momento que generaremos lo que se conoce como ondas cuadradas, como la que se ve a continuación:

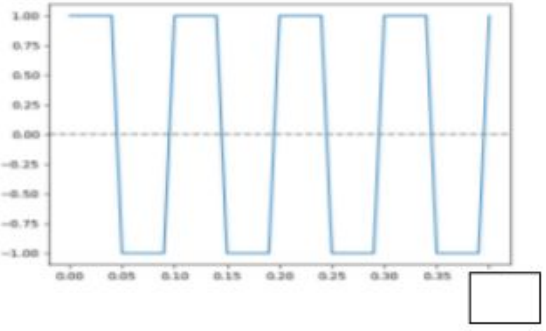
Onda cuadrada de amplitud *a*: los únicos valores que toman las muestras son *a* y *-a*.

Este tipo de ondas están representadas por muestras de únicamente dos valores: las muestras de la parte alta (que adoptan un valor positivo) y las muestras de la parte baja (que adoptan el mismo valor que las altas, pero negativo). Si bien esta forma les otorga un sonido muy particular (y tal vez no del todo agradable), es un punto de partida para facilitar

el desarrollo de esta actividad con el que luego podrán experimentar los grupos a los que les interese.

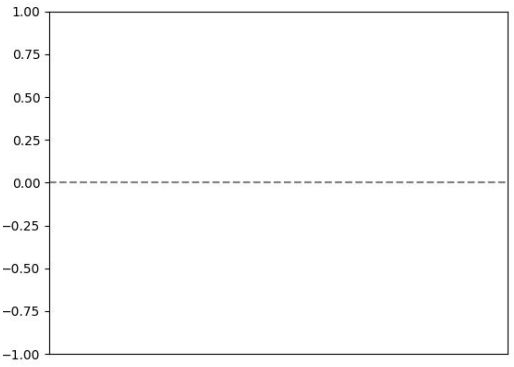
A continuación, vemos los casos de test para la función `generarSonido`, que recibe como parámetros la frecuencia, la amplitud, la duración y la tasa de muestreo y produce un sonido con estas características.

Caso de test nro 0			
generarSonido			
frecuencia = 25	amplitud = 1	duracion = 0.2s	muestreo = 150
			
duración total = 0.2 s		duración de un ciclo = 1/25 s = 0.04 s	
cantidad de ciclos = 5		muestras por ciclo = 6	

Caso de test nro 1			
generarSonido			
frecuencia = 10	amplitud = 1	duracion = 0.4s	muestreo = 100
			
duración total = **0.4 s**		duración de un ciclo = **0.4 s/4 = 0.1 s**	

cantidad de ciclos = ***4***	muestras por ciclo = ***10***

Para completar los datos faltantes, observamos que la duración total del sonido, según lo especificado en los parámetros es de 0.4 segundos[20]. Luego, observamos en el dibujo que la onda representada está formada por 4 ciclos. A partir de estos datos podemos concluir que un ciclo tiene que durar 0.1 segundos (pues 4 duran 0.4 s). Finalmente, debemos averiguar cuántas muestras corresponden a un ciclo. Para esto, recordamos que la tasa de muestreo expresa cuántas muestras hay por segundo (en este caso, 100 muestras) y ya sabemos cuánto dura un ciclo en segundos (0.1 s, o un décimo de segundo). Luego, la cantidad de muestras que corresponden a un ciclo se obtiene multiplicando estos dos valores (en este caso, 100 * 0.1 = 10). Sin embargo, estos datos pueden obtenerse intuitivamente a partir del gráfico sin necesidad de formular explícitamente la regla general. No adelantamos esta respuesta, pues deberán hacer este razonamiento para programar la función siguiente.

Caso de test nro 2			
generarSonido			
frecuencia =	amplitud = 1	duracion =	muestreo =
			
duración total =		duración de un ciclo =	
cantidad de ciclos =		muestras por ciclo =	

Una vez que hayan completado los casos de test y resuelto las preguntas necesarias para hacerlo, deberán conservarlos para poner a prueba sus programas cuando los hayan terminado. Ahora sí, pueden comenzar a programar. Para eso, les indicamos que avancen a la sección “Funciones principales” del archivo Python, donde está plasmada una parte de la estrategia que utilizaremos para trabajar en esta actividad: para generar un sonido, separaremos el problema de generar un ciclo de la onda, que resolveremos definiendo una función `generarCiclo`, y luego la usaremos en `generarSonido`, la función principal que acabamos de presentar.

[20] Utilizamos el punto (en vez de la coma) para separar la parte decimal de la parte entera pues así es como lo hace Python, dado que es lo usual en inglés. Como ya vimos, la coma se utiliza para separar valores, por ejemplo, dentro de una secuencia.

```
def generarSonido(frecuencia, amplitud, duracion, tasaDeMuestreo):
   # generamos un ciclo
   unCiclo = generarCiclo(frecuencia, amplitud, tasaDeMuestreo)

   sonido = ab.crearSecuenciaVacia()
   #   ...
   #COMPLETAR
   #   ...
   return sonido
```

Vemos que la estrategia para `generarSonido` implica, primero, generar un ciclo (con la función `generarCiclo`) y luego, construir una secuencia con el sonido completo.

Continuamos con `generarCiclo`: si miramos la definición incompleta que proveemos, veremos que utiliza algunas funciones auxiliares que también hay que completar.

```
def generarCiclo(frecuencia, amplitud, tasaDeMuestreo):
   cantidadDeMuestras =
cantidadDeMuestrasDelCiclo(frecuencia, tasaDeMuestreo)

   ciclo = ab.crearSecuenciaVacia()
   #   ...
   #COMPLETAR
   #   ...
   return ciclo
```

```
def cantidadDeMuestrasDelCiclo(frecuencia, tasaDeMuestreo):
   duracionDelCiclo = duracionDeUnCiclo(frecuencia)

   #CAMBIAR
   muestras = 0

   #La cantidad que devolvemos debe ser entera
   return int(muestras)
```

```
def duracionDeUnCiclo(frecuencia):
   #Atención que este número tiene que poder tener decimales.
   duracion = 0.0
   return duracion
```

La función principal (`generarCiclo`) y las auxiliares (`cantidadDeMuestrasDelCiclo` y `duracionDeUnCiclo`) para completar, tal cual aparecen en Act3.py.

Sugeriemos que comiencen por estas funciones, que son clave en la estrategia elegida para resolver `generarCiclo`. Para completarlas, deberán proponer e implementar una solución general para la deducción de los datos que tuvieron que completar en los casos de test. Si no lo notaran, se lo podemos señalar para que recuerden cómo hicieron para encontrar los valores pedidos en los ejemplos del principio de la actividad y que piensen, a partir de ahí, cómo escribir un programa para obtenerlos para cualesquiera otros valores. Básicamente, deben responderse las siguientes preguntas:

¿Cuántas muestras hay que generar?

Esto dependerá de la tasa de muestreo elegida y de la duración del ciclo. Suponiendo que las conocemos, dado que la tasa de muestreo determina la cantidad de muestras por

unidad de tiempo, para conocer cuántas muestras son necesarias para cubrir una cierta duración, simplemente hay que multiplicar estos valores.

```
def cantidadDeMuestrasDelCiclo(frecuencia, tasaDeMuestreo):
    duracionDelCiclo = duracionDeUnCiclo(frecuencia)
    muestras = duracionDelCiclo * tasaDeMuestreo
    return int(muestras)
```

Función `cantidadDeMuestrasDelCiclo` completa. Observamos que el valor que se produce (es decir, la cantidad de muestras), se obtiene multiplicando la duración del ciclo por la tasa de muestreo. Como ya lo vimos en algún ejemplo anterior, antes de devolver el valor lo convertimos a un número entero, pues no podemos generar una cantidad fraccionaria de muestras.

¿Cuánto dura un ciclo?

Para esto, deberán recordar lo visto en la actividad a propósito de la naturaleza del sonido. Si la frecuencia indica cuántas veces por segundo se repite un ciclo, la duración del ciclo será la fracción indicada por este valor. Por ejemplo, si una onda tiene frecuencia 500Hz (es decir, se repiten 500 ciclos en un segundo), cada ciclo debe durar 1/500 = 0.02 segundos. A este valor se lo conoce como período de la onda.

En este punto debemos estar atentos a cómo los grupos escriben esta expresión en Python. Si escriben `duracion = 1/frecuencia`, observarán que el valor que se le asigna a `duracion` es 0. Esto es así porque, al ser enteros los dos números involucrados en la división, Python interpreta que se está calculando la división entera y el resultado debe ser entero. Entonces, en el caso del ejemplo, estaríamos calculando la división entera de 1 por 500, cuyo resultado es 0 con resto 1. Para evitar esto, hay que forzar a que alguno de los números involucrados tenga decimales, es decir, sea un `float`. Esto se puede conseguir convirtiéndolo explícitamente (por ejemplo, `float(1)`), o escribiéndolo con decimales aunque no haga falta (`1.0` en vez de `1`). Observemos que hacer `float(1/500)` no tiene el efecto deseado, pues primero se calcula la división (entera) y luego se convierte el resultado (que fue 0) a `float`.

```
def duracionDeUnCiclo(frecuencia):
    return 1.0/frecuencia
```

La función `duracionDeUnCiclo` completa. Insistimos con observar que escribimos `1.0` en vez de `1` para forzar a que la división no sea entera.

Con estas dos funciones resueltas, ya pueden escribir una solución para `generarCiclo` sin mayores dificultades: si ya sabemos cuántas muestras tiene el ciclo que queremos generar, tendrá la mitad de esa cantidad de muestras altas al principio y luego, otra mitad de muestras bajas. A continuación vemos una solución posible:

```
def generarCiclo(frecuencia, amplitud, tasaDeMuestreo):
```

```
    cantidadDeMuestras = cantidadDeMuestrasDelCiclo(frecuencia, tasaDeMuestreo)

    ciclo = ab.crearSecuenciaVacia()
    mitadDeLasMuestras = cantidadDeMuestras / 2

    for i in range(0, mitadDeLasMuestras):
        ab.agregarAlFinal(ciclo, amplitud)

    for i in range(0, mitadDeLasMuestras):
        ab.agregarAlFinal(ciclo, -amplitud)

    return ciclo
```

Una manera de completar `generarCiclo` (observamos en itálica la parte completada): utilizando dos repeticiones simples, agregamos a la secuencia que será el resultado primero la mitad de las muestras con el valor alto y luego, la otra mitad con el valor bajo.

Esta solución, si bien creemos que es muy clara y por eso la presentamos, tiene un pequeño problema, que se vuelve imperceptible cuando se trabaja con valores de muestreo elevados con respecto a la frecuencia (como los que se utilizan en los sonidos estándar). Si la cantidad de muestras por ciclo es un valor impar, cuando se calcula la mitad en *`mitadDeLasMuestras = cantidadDeMuestras / 2`*, el resultado se redondea hacia abajo, lo que produce que, al final de cuentas, el ciclo cuente con una muestra menos que la que debería tener.
Por ejemplo, si la cantidad de muestras por ciclo (según lo calculado en la primera línea) es 5, `mitadDeLasMuestras` valdrá 2 y por lo tanto, el ciclo producido consistirá en dos muestras altas y dos muestras bajas, es decir, cuatro muestras en total. Esto produce que el ciclo dure menos y, por lo tanto, la frecuencia sea mayor que la pedida.
Si queremos ser precisos, podemos corregirlo utilizando una alternativa condicional y la función `esPar` que presentamos en la actividad anterior o utilizando directamente el operador `%` (que calcula el resto de la división entera), como vemos a continuación:

```
from Act2 import esPar
def generarCiclo(...):
 ...
for i in range(0, mitadDeLasMuestras):
   ab.agregarAlFinal(ciclo, -amplitud)
if not esPar(cantidadDeMuestras):
   ab.agregarAlFinal(ciclo, -amplitud)
...
```

```
def generarCiclo(...):
 ...
segundaMitad = mitadDeLasMuestras +
cantidadDeMuestras % 2
for i in range(0, segundaMitad):
   ab.agregarAlFinal(ciclo, -amplitud)
...
```

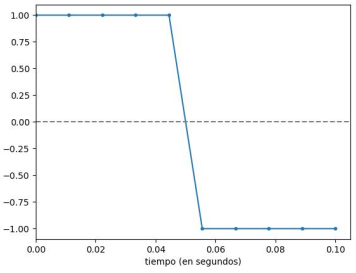
En el primer caso, si la cantidad de muestras que hay que generar no es par (o sea, es impar) agregamos una muestra baja más al final del ciclo, mientras que en el segundo, calculamos cuántas muestras bajas agregar como la mitad de las muestras (que en realidad es la división entera) más el resto de dividir por 2 (que será 1 si el número original era impar).

Es importante tener esto en cuenta, no para el proyecto final, pues la diferencia puede ser muy fina, pero sí para los casos de test, en los que hacemos énfasis en la cantidad de muestras por ciclo. También podemos desentendernos del problema si elegimos valores de prueba que requieran una cantidad par de muestras por ciclo.

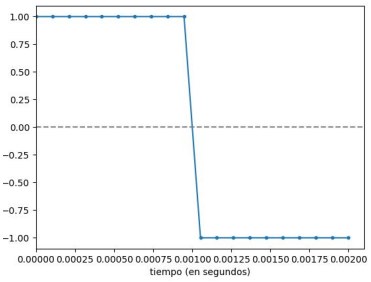
Al igual que con las actividades anteriores, insistimos para que los grupos vayan probando sus funciones a medida que van avanzando. Si bien todavía no pueden obtener sonidos completos, pueden utilizar la función `ab.graficar` (que ya la conocen) y, además, importar el archivo `audioGraficos` y experimentar con las funciones disponibles. En particular, contiene `audioGraficos.graficarCiclo` que además dibuja las muestras y agrega información de tiempo para observar los ciclos obtenidos. Dado que no escucharemos los sonidos, no nos preocuparemos por el valor de la amplitud, por lo que utilizaremos siempre el valor 1 como ejemplo.

A continuación proponemos dos casos de test completos para la función `generarCiclo`, para que los grupos puedan comparar los resultados obtenidos por su implementación de la función cuando crean que es correcta. Además, en la sección *Tests* del archivo, proveemos los métodos `test_1_generarCiclo()` y `test_2_generarCiclo()` que ejecutan `generarCiclo` con los valores correspondientes a cada caso. Nuevamente, los valores elegidos no corresponden a sonidos reales, sino a casos simples que puedan ser fácilmente comparables a ojo.

Caso de test nro 1		
`frecuencia = 10`	`amplitud = 1`	`muestreo = 100`
duración del ciclo = 1/10 s = 0.01 s	cantidad de muestras = 10	



Caso de test nro 2		
`frecuencia = 500`	`amplitud = 1`	`muestreo = 10000`
duración del ciclo = 1/500 s = 0.002 s	cantidad de muestras = 20	




```python
def test_1_generarCiclo():
    muestras = generarCiclo(10, 1, 100)
    audioGraficos.graficarCiclo(muestras, 100)
```

```python
def test_2_generarCiclo():
    muestras = generarCiclo(500, 1, 10000)
    audioGraficos.graficarCiclo(muestras, 10000)
```

Los dos casos de tests y las dos funciones que ejecutan `generarCiclo` con los valores correspondientes.

Una vez completadas estas funciones, podemos continuar con `generarSonido`. Para esto,

observamos que la estrategia elegida comienza generando un ciclo utilizando `generarCiclo` y luego crea una secuencia vacía para construir el resultado.

```
def generarSonido(frecuencia, amplitud, duracion, tasaDeMuestreo):
    # generamos un ciclo
    unCiclo = generarCiclo(frecuencia, amplitud, tasaDeMuestreo)

    sonido = ab.crearSecuenciaVacia()
    #   ...
    #COMPLETAR
    #   ...
    return sonido
```

Función `generarSonido` para completar en el archivo `Act3.py`. Observamos que lo primero que se hace es generar un ciclo utilizando la función definida anteriormente para luego construir una secuencia con el sonido completo.

En este punto, los grupos deben darse cuenta que para generar un sonido de una duración razonable, deberán repetir muchas veces el mismo ciclo. Para conocer cuántos ciclos deben repetirse para completar el sonido y cómo pueden obtener esta respuesta, les recordamos los casos de test completados al principio, donde tuvieron que completar este dato. Para expresar la regla general, pueden seguir alguno de los siguientes razonamientos:

- Si la frecuencia del sonido indica cuántos ciclos se repiten en un segundo y conocemos la cantidad de segundos de sonido que queremos generar, basta con multiplicar estos valores.
- Si recordamos que para completar la función anterior definimos la función `duracionDeUnCiclo`, podemos utilizarla para conocer la duración del ciclo generado al comienzo. Luego, si conocemos la duración total del sonido, la cantidad de veces que hay que repetir este ciclo resulta de dividir la duración total por la duración de un único ciclo.

```
duracionDelCiclo = duracionDeUnCiclo(frecuencia)
cantidadDeCiclos = int(duracion / duracionDelCiclo)
```

```
cantidadDeCiclos = int(frecuencia * duracion)
```

Las dos alternativas comentadas anteriormente, incorporadas en Python a la definición de `generarSonido`. Observamos que el resultado está forzado a ser un número entero. Debemos indicarles a los grupos que así lo hagan, para que lo que resta del programa sea más sencillo.

Una nueva dificultad que encontrarán, que ahora ya no tiene que ver con el sonido en sí, sino meramente con la manipulación de secuencias en Python, tiene que ver con cómo unir varias secuencias (es decir, cómo resolver el problema de generar una secuencia que consista en sucesivas repeticiones de una secuencia original). Sugerimos definir una función auxiliar para dejar en clara la división en subtareas (y poder reutilizarla más adelante si es necesario)

Una solución simple consiste en agregar sucesivamente los elementos de la secuencia a unir al final de la secuencia original, como vemos a continuación:

```
def agregarSecuenciaAlFinal(secuenciaBase, agregada):
   for elemento in agregada:
       ab.agregarAlFinal(secuenciaBase, elemento)
   return secuenciaBase
```

Al igual que antes, recorremos uno por uno los elementos de la secuencia a agregar y lo agregamos al final de la secuencia original. Además, devolvemos la secuencia base, aunque podríamos omitir este paso, pues `agregarAlFinal` lo agrega modificando la misma secuencia que recibió por parámetro. Para motivar esta solución, podemos recordarles que ya saben cómo hacer crecer una secuencia, con el fin de que reutilicen la función `agregarAlFinal`. Comentamos que a la operación de unir dos secuencias se la conoce como **concatenar**.

Existe una solución más breve, que en esencia es equivalente a la anterior, pero utiliza directamente el operador **+** que, al aplicarse sobre secuencias, produce la unión de ambas. Es decir `[1, 2, 3] + [4, 5, 6]` produce `[1, 2, 3, 4, 5, 6]`. Esta solución es más difícil de motivar, pero si algún grupo ya la conoce es muy probable que quiera utilizar este operador para resolver el problema o, incluso, considere innecesario definir una función que únicamente aplique **+**.

```
def agregarSecuenciaAlFinal(secuenciaBase, agregada):
   return secuenciaBase + agregada
```

Otro aspecto negativo de esta solución es que, a diferencia de la anterior, produce una nueva secuencia en vez de modificar la de base. Por lo tanto, debe devolverse el valor que produce y asignarse a una variable para conservar la modificación.

A tener en cuenta:
Hasta ahora no hicimos diferencias (ni nos interesa hacerlas) pero estuvimos trabajando con dos tipos diistintos de secuencias. Por un lado, utilizamos las listas de Python, que son las que generamos explícitamente cuando escribimos los corchetes o las que producimos con la función `crearSecuenciaVacia`. Por otro, la biblioteca numérica que utilizamos internamente tanto para levantar sonidos como para graficar (numpy), utiliza su propio tipo de secuencias por motivos de rendimiento (el *array* que descubrimos en la actividad anterior). Si bien suelen ser compatibles, no lo son para estas funciones:

- El operador + no funciona como es esperado cuando alguna de las dos secuencias es una de las secuencias de la biblioteca.
  Al querer utilizarse con la secuencia equivocada producirá un error de ejecución con un mensaje crítptico y, por lo tanto, difícil de detectar (“ValueError: operands could not be broadcast together with shapes...”).

- La función `agregarAlFinal` funciona solo para secuencias de Python y, por lo tanto, la `secuenciaBase` de `agregarSecuenciaAlFinal` debe ser de este tipo. En caso de utilizar una secuencia incorrecta el error producido es solo un poco más claro: AttributeError: 'numpy.ndarray' object has no attribute 'append'

De todas maneras, siempre evitamos devolver secuencias que no sean nativas de Python, pero por alguna modificación podría ser que esto no fuera así y se produjera alguno de los errores citados.

Ahora sí, pueden escribir la parte restante de la función `generarSonido` que, como ya identificaron luego del ejercicio en papel, consiste en generar una secuencia con una determinada cantidad de repeticiones de la secuencia `ciclo`.

```
for i in range(0, cantidadDeCiclos):
```

`agregarSecuenciaAlFinal(sonido, ciclo)`	`sonido = sonido + ciclo`

Junto con una repetición simple (acá vemos por qué queríamos que el valor de `cantidadDeCiclos` fuera un número entero: para poder utilizarlo como la cantidad de repeticiones) y una concatenación, construimos `sonido` como una secuencia que contiene repetidas veces la secuencia `ciclo`. Presentamos las dos alternativas para la concatenación: en la primera agregamos los elementos al final de la secuencia `sonido`, mientras que en la segunda producimos una nueva secuencia con + y la asignamos a la variable `sonido`.

Finalmente, la definición de la función `generarSonido` puede completarse, como vemos a continuación (contemplando las alternativas propuestas en cada punto).

```
def generarSonido(frecuencia, amplitud, duracion, tasaDeMuestreo):
    # generamos un ciclo
    ciclo = generarCiclo(frecuencia, amplitud, tasaDeMuestreo)

    sonido = ab.crearSecuenciaVacia()

#Calculamos la cantidad de ciclos que hay que generar para cubrir la duración
```

`duracionDelCiclo = duracionDeUnCiclo(frecuencia)` `cantidadDeCiclos = int(duracion / duracionDelCiclo)`	`cantidadDeCiclos = int(frecuencia * duracion)`

```
#Generamos una secuencia que contenga las repeticiones necesarias del ciclo
for i in range(0, cantidadDeCiclos):
```

`agregarSecuenciaAlFinal(sonido, ciclo)`	`sonido = sonido + ciclo`

```
return sonido
```

Otra solución:
Otra solución que puede surgir es utilizar la función `generarRepetidos` (que

programamos como ejercicio preliminar) para generar, justamente, la repetición de los ciclos. Sin embargo, con esto no es suficiente, pues recordemos que esta función producía una secuencia con el elemento modelo repetido. Luego, si el elemento es un ciclo (que es una secuencia), el resultado será una secuencia de secuencias, en vez de una secuencia con todas las muestras. Por ejemplo, `generarRepetidos([1, -1], 3)` produce `[[1, -1], [1, -1], [1, -1]]` en vez de `[1, -1, 1, -1, 1, -1]`.
Para eso, deberán prorgamar una función que, dada una secuencia de secuencias, produzca una única secuencia con los elementos en el mismo orden. Es decir, que, a partir de `[[1, 2], [3, 4], [5, 6]]` (es decir, la secuencia que contiene a las secuencias `[1, 2]`, `[3, 4]` y `[5, 6]`) produzca la secuencia `[1, 2, 3, 4, 5, 6]`. A esta operación se la conoce como “aplanar” y, en algún punto, es similar a lo que hicieron en el capítulo anterior al trabajar con imágenes, pues hay dos recorridos anidados: por un lado, acceder, una por una, a las secuencias, y, una vez elegida una de éstas, acceder uno por uno a sus elementos.

```
def generarSonido(frecuencia, amplitud, duracion, tasaDeMuestreo):
   ciclo = generarCiclo(frecuencia, amplitud, tasaDeMuestreo)
   cantidadDeCiclos = int(frecuencia * duracion)
   ciclosRepetidos = generarRepetidos(ciclo, cantidadDeCiclos)
   sonido = aplanar(ciclosRepetidos)
   return sonido


def aplanar(secuenciaDeSecuencias):
   todosLosElementos = ab.crearSecuenciaVacia()
   for secuencia in secuenciaDeSecuencias:
       for elemento in secuencia:
           ab.agregarAlFinal(todosLosElementos, elemento)

   return todosLosElementos
```

De la misma manera que antes, sugerimos que, una vez que hayan completado la función y sospechen que funciona correctamente, comparen los resultados obtenidos con ella contra los casos de test que definieron al principio. Para esto, deberán generar sonidos con las características planteadas en los tests y utilizar la función `audioGraficos.graficarSonido` para comparar los gráficos de los resultados obtenidos con los esperados. Si la comparación es satisfactoria, ya podemos generar archivos wav y finalmente escuchar los sonidos que sintetizamos. Pueden escribir métodos análogos a `test_1_generarCiclo()` para encapsular estas dos acciones y asegurar que generarSonido se utiliza con los parámetros que especifica cada caso de test.

Una última consideración para generar archivos audibles: deberemos trabajar con los valores de amplitud que establece el estándar *wav*. Recordamos que la amplitud determina el volumen del sonido, y, por lo tanto, si utilizamos valores muy pequeños, el sonido no se escuchará. También, el formato de representación impone límites que no se pueden exceder. Para averiguar estos límites, les podemos sugerir que carguen en la consola sus programas para calcular máximos y mínimos de secuencias y que los utilicen para calcular el máximo y el mínimo de la secuencia de muestras de algún archivo de ejemplo.

Dependerá del archivo, pero es muy probable que se encuentren con valores alrededor de 32000 y -32000 respectivamente. Si no quisiéramos dedicar tiempo a esta parte, simplemente podemos indicarles que estos son los límites y que pueden experimentar con valores intermedios de amplitud, enfatizando que cuanto menores sean (en valor absoluto) más débil se escuchará el sonido.

Por ejemplo, con el siguiente programa generamos un sonido La_3 de 2 segundos de duración y muestreo de 44100, y lo guardamos en el archivo `la_generado.wav`:

```
muestrasLa = generarSonido(440, 32000, 2, 44100)
ab.guardarComoWav(muestrasLa, 44100, 'la_generado.wav')
```

De hecho, podemos abstraer este proceso en una única función que tome como parámetros la frecuencia, la duración y el nombre del archivo (y utilice un valor de amplitud y de frecuencia de muestreo estándar, dado que como vimos, no solemos modificarlos) y que genere directamente el archivo de sonido.

```
def generarArchivoDeSonido(frecuencia, duracion, nombreDeArchivo):
    tasaDeMuestreo = 44100
    muestras = generarSonido(frecuencia, 16000, duracion, tasaDeMuestreo)
    ab.guardarComoWav(muestras, tasaDeMuestreo, nombreDeArchivo)
```

Definimos una función que contenga las operaciones necesarias para generar un sonido y guardarlo como wav, cuyos únicos parámetros sean la frecuencia, la duración y el nombre del archivo. La tasa de muestreo (que debe ser la misma en la generación de las muestras que en el guardado del archivo, y por eso la guardamos en una variable) se establece en 44100 para respetar el estándar de CD y la amplitud en 16000 para utilizar un valor intermedio.

Consigna extra: *Experimentar con otras formas de onda. En el archivo Act3_ciclos.py proveemos otras funciones, análogas a `generarCiclo` pero con otras formas. Aquellos grupos que hayan terminado y quieran explorar distintas alternativas, pueden importarlas a sus programas e incorporarlas a la función `generarSonido` para observar los resultados que producen. También pueden graficar los sonidos obtenidos para observar la diferencia de formas y asociarlas con las características audibles.*

Consigna extra: *Cuando generamos los ciclos, utilizamos la mitad de muestras altas y la otra mitad de muestras bajas. ¿Cómo se escuchará un sonido si hacemos esto de manera diferente? (Por ejemplo, ponemos un tercio de muestras altas y el resto muestras bajas, o todas bajas menos una alta.)*

Para completar estas consignas simplemente hay que modificar la instrucción de `generarSonido` donde se utiliza `generarCiclo` para generar un ciclo, como vemos a continuación:

```
def generarSonido(frecuencia, amplitud, duracion,
tasaDeMuestreo):
   # generamos un ciclo
   ciclo = generarCiclo(
           frecuencia, amplitud, tasaDeMuestreo)

   sonido = ab.crearSecuenciaVacia()

   cantidadDeCiclos = int(frecuencia * duracion)

   for i in range(0, cantidadDeCiclos):
       sonido = agregarSecuenciaAlFinal(sonido,
ciclo)

   return sonido
```

```
def generarSonidoSinusoidal(frecuencia, amplitud,
duracion, tasaDeMuestreo):
   # generamos un ciclo sinusoidal
   ciclo = a3Ciclos.generarCicloSinusoidal(
           frecuencia, amplitud, tasaDeMuestreo)

   sonido = ab.crearSecuenciaVacia()

   cantidadDeCiclos = int(frecuencia * duracion)

   for i in range(0, cantidadDeCiclos):
       sonido = agregarSecuenciaAlFinal(sonido,
ciclo)

   return sonido
```

Con este esquema, para cada generador de ciclos distinto, deberíamos repetir la definición de `generarSonido`, solo para cambiar una línea. Sin embargo, en Python también podemos utilizar parámetros para funciones. De esta manera, el generador de ciclos puede ser un dato maś de entrada, y la función se limita a utilizarla.

```
def generarSonidoGenerico(frecuencia, amplitud,
duracion, tasaDeMuestreo, generadorDeCiclo):
   ciclo = generadorDeCiclo(frecuencia, amplitud,
tasaDeMuestreo)

   sonido = ab.crearSecuenciaVacia()

   cantidadDeCiclos = int(frecuencia * duracion)

   for i in range(0, cantidadDeCiclos):
       sonido = agregarSecuenciaAlFinal(sonido,
ciclo)

   return sonido
```

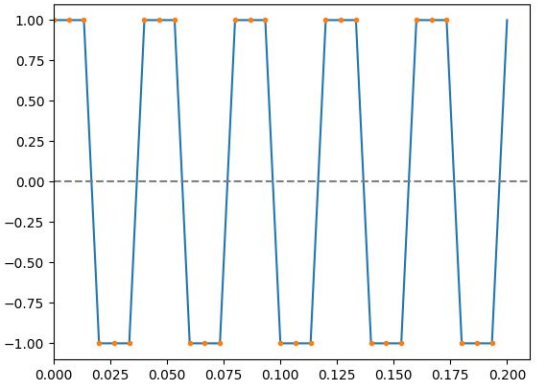
```
sonidoCuadrado = generarSonidoGenerico(440, 16000, 1,
44100, generarCiclo)

sonidoSinusoidal = generarSonidoGenerico(440, 16000,
1, 44100, a3Ciclos.generarCicloSinusoidal)

def generarSonidoCuadrado(frecuencia, amplitud,
duracion, tasaDeMuestreo):
   return generarSonidoGenerico(frecuencia, amplitud,
duracion, tasaDeMuestreo, generarCiclo)

def generarSonidoSinusoidal(frecuencia, amplitud,
duracion, tasaDeMuestreo):
   return generarSonidoGenerico(frecuencia, amplitud,
duracion, tasaDeMuestreo,
a3Ciclos.generarCicloSinusoidal)
```

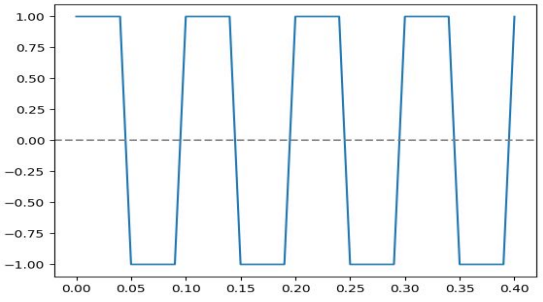
Parametrizamos el generador de ciclos: simplemente se agrega el parámetro a la definición y luego se utiliza como si fuera una función que ya está definida (al igual que hacíamos con los valores: le asignábamos un nombre en la definición de la función y luego lo utilizábamos como si existiera). Luego, usamos la función genérica con los mismos argumentos que antes (frecuencia, amplitud, duración y tasa de muestreo) y, además, un generador de ciclos (ya sea el orginal, que producía ondas cuadradas o el sinusoidal). Observamos, también, que podemos definir funciones que encapsulen esta tarea, simplemente llamando a la función genérica con los argumentos originales y el generador de ciclo correspondiente.

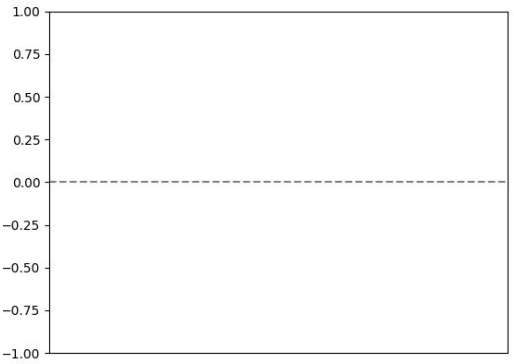
Ficha

En esta parte, generaremos nuestros propios sonidos. Trabajaremos con ondas cuadradas, como la que se ve a continuación:

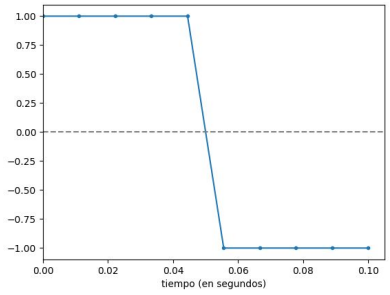
Antes de comenzar, definan los siguientes casos de test para la función `generarSonido`, que será la encargada de producir el sonido según las características especificadas.

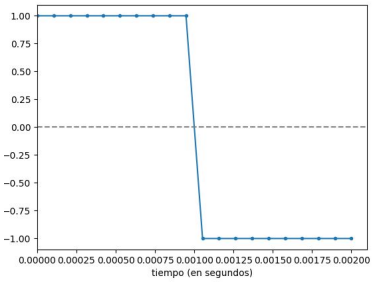
Caso de test nro 0			
generarSonido			
frecuencia = 25	amplitud = 1	duracion = 0.2s	muestreo = 150
			
duración total = 0.2 s		duración de un ciclo = 1/25 s = 0.04 s	
cantidad de ciclos = 5		muestras por ciclo = 6	

Caso de test nro 1			
generarSonido			
frecuencia = 10	amplitud = 1	duracion = 0.4s	muestreo = 100
			
duración total =		duración de un ciclo =	
cantidad de ciclos =		muestras por ciclo =	

Caso de test nro 2			
generarSonido			
frecuencia =	amplitud = 1	duracion =	muestreo =
			
duración total =		duración de un ciclo =	
cantidad de ciclos =		muestras por ciclo =	

Cuando los tengan listos, pueden empezar a programar. En algún momento van a tener que completar la función generarCiclo, que pueden probarla contra los ejemplos que aparecen a continuación:

Caso de test nro 1		
frecuencia = 10	amplitud = 1	muestreo = 100
duración del ciclo = 1/10 s = 0.01 s	cantidad de muestras = 10	
		

Caso de test nro 2		
frecuencia = 500	amplitud = 1	muestreo = 10000
duración del ciclo = 1/500 s = 0.002 s	cantidad de muestras = 20	
		

Cuando estén seguros de que su función `generarSonido` se comporta como esperamos, pueden definir la función `generarArchivoDeSonido(frecuencia, duracion, nombreDeArchivo)` para generar un sonido y guardarlo, en una misma instrucción. Así, pueden generar archivos más rápido y probar cómo se escuchan.

Para experimentar: ¿Cómo afecta al sonido la forma de la onda?

Para esto, pueden probar, primero, generando un ciclo asimétrico (es decir, no poner mitad de muestras bajas y mitad de altas, sino, por ejemplo, un tercio de bajas y dos tercios de altas, o una alta y el resto bajas, etc.). Luego, en el archivo ciclos.py, hay funciones para generar ciclos con otras formas. Pueden incorporarlos a sus programas, escuchar los resultados y graficar sus formas.

Actividad 4 | SD1

Generamos melodías

Objetivos

- Proponer un sistema de representación de melodías y programar un intérprete para el mismo.
- Utilizar las funciones definidas previamente para generar melodías.

Modalidad de trabajo

En parejas

Materiales y recursos utilizados

- Computadoras con Python (preferentemente PyCharm Edu) y nuestras bibliotecas de sonido previamente instaladas.
- Auriculares.
- Proyector (opcional).

Desarrollo

Hasta ahora, conseguimos generar sonidos puros utilizando programas en Python. Sin embargo, estamos lejos de poder generar una canción. En este momento del proyecto daremos un fuerte paso en esta dirección: escribiremos un programa que, valiéndos de las funciones que escribimos anteriormente, pueda leer una melodía codificada en un texto y producir un archivo de sonido con ella.

> Proveemos el archivo Act3_min.py con las mínimas funciones requeridas para poder completar esta actividad, para aquellos grupos que, por algún motivo u otro, no tengan completa la actividad anterior o no la tengan disponible en el momento de la clase.

Comenzamos la actividad contando que utilizaremos los programas que construimos hasta ahora para hacer un programa capaz de generar melodías. Mejor aún: nuestro programa leerá un texto en el que esté codificada la melodía, la interpretará y, a partir de generar sonidos puros con las funciones que ya tenemos, generará un sonido con la melodía completa.

Lo primero que debemos preguntarles a los grupos es si conocen cómo se representa (se “escribe”) la música. Es muy probable que en la clase de música conozcan las notas, las figuras y los pentagramas. Básicamente, este sistema parte de un pentagrama (es decir, cinco líneas paralelas) y asocia cada posición entre ellas con una nota en particular. Además, para definir la duración del sonido, utiliza distintos símbolos (negras, corcheas, semicorcheas, etc.), donde cada uno representa la mitad de la duración que el anterior. Naturalmente, el sistema completo es mucho más complejo, pero nos alcanza con estos elementos para lo que queremos hacer.

A continuación, observamos que si queremos que la codificación propuesta sea procesada por nuestro programa, no puede ser gráficamente tan compleja. Por el contrario, deberá estar limitada a texto. Veremos, sin embargo, que esta no es una limitación demasiado grande.

Ahora sí, podemos pasar a la parte de programación:

> ***Consigna final:*** *Escribir un programa que, a partir de una secuencia de notas codificada en un texto, prodzca el sonido compuesto por la sucesión de esas notas, todas de la misma drarción. Es decir, si la melodía codificada es "do, re, mi, do, mi, do, mi", en el archivo de sonido resultante deben escucharse estas notas (la duración de cada nota queda a definir por los grupos).*

A partir de esta consigna, esperamos que los grupos identifiquen dos problemas para resolver:

- cómo codificar la melodía, y
- cómo generar los sonidos.

Para el segundo, deberán pensar en cómo reutilizar la función que programaron en la actividad anterior. Para lo primero, pueden ser tan creativos como gusten, sin embargo, la codificación propuesta debe ser:

- unívocamente decodificable, es decir, no deben poder decodificarse dos melodías distintas a partir del mismo texto;
- extensible, es decir, deben poder agregársele otros atributos además del nombre de la nota (por ahora, solo sintetizamos siete notas distintas, pero después nos gustaría poder codificar sonidos de distintas duraciones o de distintas octavas, por ejemplo);
- fácilmente decodificable, es decir, no debería producir dificultades excesivas de programación la decodificación. Para ser más restrictivos (pero también más precisos), sugerimos enfáticamente que exploren el comportamiento de la función `partir` para utilizarla en sus decodificaciones. Para esto, tienen definida en el archivo Act4.py `secuenciaDePrueba`: el texto de prueba *Hola Manola como te va*. Les pedimos que observen el resultado de la ejecución de `partir(secuenciaDePrueba)` y `partir(secuenciaDePrueba, 'o')`.

`>>> partir(secuenciaDePrueba)`
`['Hola', 'Manola,', 'como',`
`'te', 'va.']`
	`>>> partir(secuenciaDePrueba,`
	`'o')`
	`['H', 'la Man', 'la, c', 'm', '`
	`te va.']`

Observamos que la función produce una secuencia, resultado de separar la secuencia original (en este caso, de caracteres) en algunos lugares específicos: si no la llamamos con otros argumentos más que con la secuencia original, la separa en los espacios (o cualquier otro caracter en blanco, como una marca de tabulación o un salto de línea), mientras que si proveemos un argumento más,	

utiliza este elemento para hacer los cortes (en nuestro caso, observamos que la secuencia está cortada en todos los lugares donde había una o).

A partir de estas restricciones y de la presentación de la función `partir`, damos un tiempo a los grupos para que piensen una codificación. Idealmente, debemos orientarlos para que se concentren más en la simplicidad (y puedan avanzar con el resto de la actividad) que en la originalidad de la representación propuesta. De hecho, escribir los nombres de las notas separados por espacios (o un número del 1 al 7), cumple las condiciones pedidas, como vemos en los ejemplos a continuación:

La codificación que representa cada nota con su inicial, no es adecuada. ¿Por qué?
Pista: ¿Cómo se codifica una escala, es decir: Do, Re, Mi, Fa, Sol, La, Sí?
(Respuesta: Porque no es unívocamente decodificable, pues sin información extra no podemos saber si la S representa a Sol o a Sí. En el ejemplo propuesto, la codificación resultante sería D R M F S L S, que puede decodificarse a la escala original, pero también a Do, Re, Mi, Fa, Si, La, Sol, por ejemplo.)

Melodía a codificar: Do, Re, Mi, Do, Mi, Do, Mi.		
Codificación A	Codificación B	Codificación C
`m = “DO RE MI DO MI DO MI”`	`m = “1 2 3 1 3 1 3”`	`m = “do-re-mi-do-mi-do-mi”`
`>>> partir(m)` `['DO', 'RE', 'MI', 'DO', 'MI', 'DO', 'MI']`	`>>> partir(m)` `['1', '2', '3', '1', '3', '1', '3']`	`>>> partir(m, '-')` `['do', 're', 'mi', 'do', 'mi', 'do', 'mi']`
Tres codificaciones simples que pueden resolverse con la función `partir`. Observamos que en el último caso, las notas, en vez de estar separadas por un espacio, están separadas por el carácter `-`, por lo cual éste debe ser el segundo argumento de la función.		

Una sugerencia para evitar errores innecesarios es que utilicen siempre mayúsculas o siempre minúsculas. Dado que en Python “Do” es distinto de “do” y de “DO”, si la codificación propuesta es “Do” y luego, a la hora de utilizarlo, se escribe “do” o “DO” en el programa, éste no lo reconocerá y no se comportará como se espera, lo que muy probablemente desoriente o desmotive a los grupos, ya que deberán dedicar tiempo a detectar y corregir este error.

Una vez decidida la codificación, resta generar sonidos a partir de la secuencia de notas producida a partir del texto. Hacemos énfasis en que ya tenemos resuelta una parte difícil, que es la de efectivamente generar el sonido, en la función `generarSonido` que completamos anteriormente.

Como siempre, antes de comenzar a programar nos concentramos en la estrategia. De la misma manera que hicimos antes, les indicamos que piensen qué funciones usarían para resolver cada paso, pero con la salvedad de que ahora también deberán identificar las funciones que falten y ponerles un nombre apropiado; la única que sugerimos nosotros es `interpretarFrecuencia(nota)` que, dado un código de nota, produce la frecuencia correspondiente al sonido de esa nota.

Damos un tiempo para que cada grupo arme una estrategia y hacemos una puesta en común. Una posible es la siguiente:

Paso de la estrategia	Función que podemos utilizar para realizarlo
1. Obtener la secuencia con los códigos de las notas a partir del texto que codifica la melodía.	`separarEnNotas`
2. Para cada código de esta secuencia: *a. obtener la frecuencia correspondiente a cada nota* *b. generar un sonido con esta frecuencia y una duración, una amplitud y una tasa de muestreo predeterminadas.*	`interpretarFrecuencia` y `a3.generarSonido`
3. Unir estos sonidos.	`a3.agregarSecuenciaAlFinal`
4. Devolver el resultado	
Una estrategia posible y las funciones que utiliza, identificando las que deben definir (naranja), las que proponemos para completar (verde) y las que se pueden reutilizar de la actividad anterior (azules). Deben definir la función `separarEnNotas` en base a la codificación decidida y completar la definición de `interpretarSecuencia` que proponemos. Además, para poder utilizar las funciones de la actividad anterior, importamos el archivo que las contiene bajo el nombre de `a3`, por eso lo anteponemos a `generarSonido` y `agregarSecuenciaAlFinal`.	

En este momento, les indicamos que comiencen a programar teniendo en cuenta la estrategia que habían propuesto.

Para resolver el primer paso, simplemente deben programar en una función la separación de los códigos en base al sistema de representación que propusieron al principio. Además, deben completar la variable `partitura` con una melodía de prueba para ir probando sus programas con un ejemplo sencillo.

`partitura = "do re mi do mi do mi"`
`def separarEnNotas(partitura):`
`  return partir(partitura)`
	Observamos una melodía de ejemplo, codificada simplemente como los nombres de las notas en minúsculas y separadas por espacios. De esta manera, para dividirlas basta con usar la función `partir` sin especificar un separador.

Para el segundo paso, lo primero que podemos observar es que lo que se está haciendo es procesar una secuencia de elementos (en este caso, de códigos de notas). De esta manera, debería surgir la motivación de usar un recorrido `for`.

Para averiguar qué frecuencia corresponde a cada símbolo, es decir, completar la función `interpretarFrecuencia`, podemos preguntarles cómo lo harían si solo tuvieran dos símbolos posibles, para que identifiquen que necesiten una alternativa condicional. Por ejemplo, si los dos símbolos fueran "+" o "-", esto podría resolverse con una construcción similar a `if simbolo == '+': ... else ...`. Sin embargo, ahora, como tenemos más símbolos, si el símbolo en cuestión no es igual al primero, no sabemos qué debemos hacer. Por el contrario, debemos seguir preguntando si es alguno de los seis restantes. Para esto, recordamos la construcción `if elif else`, y aclaramos que se le pueden agregar más ramas "`elif`", como vemos a continuación.

```
if condición1:
   bloque1
elif condición2:
   bloque2
elif condición3:
   bloque3
else:
   bloque4
```

```
if condición1:
   bloque1
else:
   if condición2:
       bloque2
   else:
       if condición3:
           bloque3
       else:
           bloque4
```

```
if condición1:
   bloque1
else:
   if condición2:
       bloque2
   elif condición3:
       bloque3
   else:
       bloque4
```

Tres programas equivalentes escritos con distintas combinaciones de alternativas condicionales. Observamos que la construcción `if elif elif ... else` resulta mucho más sintética y fácil de leer.

```
def obtenerFrecuencia(nota):
   if nota == 'do':
       return 261
   elif nota == 're':
       return 294
   elif nota == 'mi':
       return 330
   elif nota == 'fa':
       return 349
   elif nota == 'sol':
       return 400
   elif nota == 'la':
       return 440
   elif nota == 'si':
       return 494
   else:
       print "El código es incorrecto"
```

Observamos cómo, para cada valor posible del símbolo devolvemos el valor de frecuencia correspondiente. Estos valores (que aquí aparecen aproximados) ya los averiguaron en las primeras actividades del capítulo cuando trabajaron con ondas y frecuencias. Además, sugerimos agregar el último caso para identificar cuándo se está produciendo un error en la decodificación.

Una vez que hayan resuelto este problema no necesitan definir nada nuevo, dado que pueden reutilizar las funciones definidas en la actividad anterior. La única dificultad adicional es que para poder utilizarlas, como no están definidas en el mismo archivo .py, deben importar el archivo (con el mismo objetivo aparece importado el archivo `audioBasico`).

```
import audioBasico as ab
#Agregamos esta linea para poder acceder a las funciones definidas en
Act3_completo:
import Act3_completo as a3
def generarMelodia(partitura):
   secuenciaDeNotas = separarEnNotas(partitura)
   melodia = ab.crearSecuenciaVacia()

   #Utilizamos el esquema for in para acceder, una por una, a las notas de
la secuencia resultado
   for nota in secuenciaDeNotas:
       #Utilizamos la función interpretarFrecuencia para obtener el valor
de frecuencia necesario
       frecuenciaDeLaNota = interpretarFrecuencia(nota)
       #Generamos un sonido con esta frecuencia y valores preestablecidos
de amplitud, duración y muestreo.
       #Obervamos que para referirnos a la función generar sonido debemos
anteponer a3., pues está definida en uno de los archivos importados
       sonidoDeLaNota = a3.generarSonido(frecuenciaDeLaNota, 16000, 1,
44100)
       #Utilizamos la misma función (y de la misma manera) que cuando
necesitamos unir los ciclos repetidos para generar un sonido de duración
mayor.
       a3.agregarSecuenciaAlFinal(melodia, sonidoDeLaNota)

   return melodia
```

Como siempre, motivamos que vayan probando el programa a medida que lo escriban. Cuando terminen, ya tendrán funcionando una versión rudimentaria del sintetizador con la que pueden experimentar. Al igual que con la actividad anterior, para escuchar el sonido generado, deberán guardarlo en un archivo *wav* y luego reproducirlo:

```
>>> muestrasDoReMi = generarMelodia(partitura)
>>> ab.guardarComoWav(muestrasDoReMi, 44100, 'doremi.wav')
```

Utilizamos la función `generarMelodia` con la melodía de ejemplo para generar las muestras, que luego guardamos utilizando la función `guardarComoWav` como antes.

Cuando lleguen hasta aquí, los grupos que lo deseen pueden agregarle alguna(s) de las siguientes características que proponemos, haciendo algunas modificaciones pequeñas al programa.

- Permitir generar silencios
  En la música, los silencios son importantísimos y, por eso, el sistema tradicional para escribir música que vimos al comienzo de la actividad tiene una representación especial para ellos. Para completar nuestro sintetizador, queremos agregarle la posibilidad de generar silencios.
  Para esto, los grupos deberán:
  - definir un símbolo que represente el silencio (por ejemplo “x”),
  - definir una función `generarSilencio` que genere muestras correspondientes al silencio. Para esto, preguntamos cómo serán las muestras de un silencio, si el sonido eran variaciones en la presión del aire (o en el valor de las muestras), para identificar que será una secuencia de números sin variaciones: en particular, podemos elegir el 0. La única dificultad es calcular la cantidad de muestras, pero puede resolverse observando la definición de `generarSonido` y `generarCiclo` de la actividad anterior.
  - Además, deberán incorporarla a la función principal.

  Proponemos la siguiente solución:

```
def generarMelodiaConSilencios(partitura):
   secuenciaDeNotas = separarEnNotas(partitura)
   melodia = ab.crearSecuenciaVacia()
   tasaDeMuestreo = 44100

   for nota in secuenciaDeNotas:
       duracionDeLaNota = 1
       #Antes de generar el sonido, debemos agregar una alternativa condicional para
identificar el silencio.
       #En este caso, el código 'x' identifica al silencio.
       if nota == 'x':
           sonidoDeLaNota = generarSilencio(duracionDeLaNota, tasaDeMuestreo)
       else:
           frecuenciaDeLaNota = interpretarFrecuencia(nota)
           sonidoDeLaNota = a3.generarSonido(frecuenciaDeLaNota, 16000, duracionDeLaNota,
tasaDeMuestreo)

       a3.agregarSecuenciaAlFinal(melodia, sonidoDeLaNota)

   return melodia

def generarSilencio(duracion, tasaDeMuestreo):
   cantidadDeMuestras = int(duracion * tasaDeMuestreo)
   #Utilizamos la función generarRepetidos de los preliminares de la actividad anterior.
   muestrasDelSilencio = a3.generarRepetidos(0, cantidadDeMuestras)
   return muestrasDelSilencio
```

- Permitir que las notas tengan duraciones diferentes.
  Al igual que con la representación tradicional de la música (donde la ubicación en el

pentagrama identifica la nota y la forma de la figura, la duración), nos gustaría poder indicar la duración de cada nota. Es decir, queremos extender la codificación propuesta de manera que cada código identifique una nota y una duración. Para esto, hay que resolver las siguientes dificultades:

- Codificar las duraciones. En música, se identifica la duración de una figura considerada la unidad (la "negra") y se define el resto de las figuras como múltiplos o fracciones de ésta (por ejemplo, la corchea dura la mitad de una negra y la blanca, el doble). Luego, si la negra dura un segundo, la corchea durará medio y la blanca, dos. Por lo tanto, pueden codificarse estas figuras con símbolos elegidos (por ejemplo, las iniciales, si no se repiten) o puede recuperarse la numeración que se utiliza para la base de los compases. Su uso en el sistema de notación musical no es relevante, pero se identifica a la negra con el 4, a la corchea con el 8 y a la blanca con el 2, es decir, comienza con 1 para la redonda y se duplica en cada paso, como vemos a continuación.

Figura	Duración	Código
Redonda	4 Tiempos	1
Blanca	2 Tiempos	2
Negra	1 Tiempo	4
Corchea	1/2 Tiempo	8
Semicorchea	1/4 Tiempo	16
Fusa	1/8 Tiempo	32
Semifusa	1/16 Tiempo	64

- Combinar la codificación para la nota y la duración de manera que sea fácil identificar cada una. Una solución sencilla es elegir un nuevo carácter para separar estos dos valores, de manera de poder obtenerlos utilizando la función `partir`. Por ejemplo, podemos utilizar el símbolo '*' para producir códigos como "re*4" (de esta manera, si hacemos `partir("re*4", "*")`, obtendremos `["re", "4"]`).

En ambos casos, no esperamos que descubran estas soluciones, sino cómo incorporarlas a sus programas.

```
def generarMelodiaConDuracion(partitura):
   secuenciaDeCodigos = separarEnNotas(partitura)
   duracionDeUnTiempo = 1
   tasaDeMuestreo = 44100
   melodia = ab.crearSecuenciaVacia()

   for codigo in secuenciaDeCodigos:
       #Utilizamos la funcion partir para separar la duración del nombre
de la nota
       #Accedemos a la secuencia producida para extraer cada uno.
       notaYDuracion = ab.partir(codigo, '*')
       codigoDeLaNota = notaYDuracion[0]
       #Observamos que lo que nos viene es texto, queremos convertirlo a
número para hacer cuentas
       duracionDeLaNota = int(notaYDuracion[1])

       duracionEnSegundos = calcularDuracion(duracionDeLaNota,
duracionDeUnTiempo)

       if codigoDeLaNota == 'x':
           sonidoDeLaNota = generarSilencio(duracionEnSegundos,
tasaDeMuestreo)
       else:
           frecuenciaDeLaNota = interpretarFrecuencia(codigoDeLaNota)
           sonidoDeLaNota = a3.generarSonido(frecuenciaDeLaNota, 16000,
duracionEnSegundos, tasaDeMuestreo)

       a3.agregarSecuenciaAlFinal(melodia, sonidoDeLaNota)

   return melodia


def calcularDuracion(duracionDeLaNota, duracionDeUnTiempo):
   #La siguiente fórmula se puede observar de la tabla:
   #   A medida que aumenta el valor del código, baja la duración, luego
podemos pensar que identifica una fracción.
   #   Vemos que no es exactamente 1 / el código, pues la negra sería 1/4
y es 1. Luego, multiplicamos el resultado por 4 para corregir este
defasaje.
   cantidadDeTiempos = 1.0/duracionDeLaNota * 4
   return  cantidadDeTiempos * duracionDeUnTiempo
```

Una solución que utiliza los números de la base de los compases (que, además, permite experimentar con valores intermedios). Otra solución podría ser identificar cada figura con una letra y utilizar un esquema `if elif elif else` como para las notas. Observamos los agregados en la función principal en negrita.

- Permitir identificar diferentes octavas
  Como vimos en las primeras actividades del capítulo, las notas se dividen en octavas, y así, existen “does” más agudos y “does” más graves. Proponemos

extender la codificación para poder representar la información de octavas. Lo interesante es que, si conocemos la frecuencia correspondiente a una nota para una octava, la frecuencia de la nota en la octava siguiente (más aguda) es el doble, mientras que en la anterior (más grave) es la mitad. Por ejemplo, si la frecuencia de la_3 es 440 Hz, la frecuencia de la_2 es 220 Hz y la de la_4, 880Hz. Luego, de una manera similar a cómo agregamos la información de duración, podemos agregar el número de octava junto al nombre de la nota con un nuevo carácter, por ejemplo "do^3". De la misma manera en la que obtuvimos la información de duración a partir del código completo, podemos obtener la información de octava utilizando la función `partir` con el separador correcto como argumento.

```
def generarMelodiaConOctavas(partitura):
    secuenciaDeCodigos = separarEnNotas(partitura)
    duracionDeUnTiempo = 1
    tasaDeMuestreo = 44100
    melodia = ab.crearSecuenciaVacia()

    #La codificación es, por ejemplo, "do^3*4" para identificar una negra
de do central.
    for codigo in secuenciaDeCodigos:
        #Utilizamos la funcion partir para separar la duración del nombre
de la nota y la octava
        #Accedemos a la secuencia producida para extraer cada uno.
        notaYDuracion = ab.partir(codigo, '*')
        codigoNotaYOctava = notaYDuracion[0]
        duracionDeLaNota = int(notaYDuracion[1])

        duracionEnSegundos = calcularDuracion(duracionDeLaNota,
duracionDeUnTiempo)

        if codigoNotaYOctava == 'x':
            sonidoDeLaNota = generarSilencio(duracionEnSegundos,
tasaDeMuestreo)
        else:
            notaYOctava = ab.partir(codigoNotaYOctava, '^')
            nombreDeLaNota = notaYOctava[0]
            octava = int(notaYOctava[1])
            frecuenciaDeLaNota = interpretarFrecuencia(nombreDeLaNota)
            frecuenciaDeLaNota =
aplicarModificacionDeOctava(frecuenciaDeLaNota, octava)
            sonidoDeLaNota = a3.generarSonido(frecuenciaDeLaNota, 16000,
duracionEnSegundos, tasaDeMuestreo)

        a3.agregarSecuenciaAlFinal(melodia, sonidoDeLaNota)

    return melodia



def aplicarModificacionDeOctava(frecuenciaBase, octava):
    #Hay que restar 3 porque estamos utilizando la octava 3 como cero.
    cantidadDeOctavasACorrer = octava - 3
    #Avanzar (o retroceder) varias octavas es multiplicar (o dividir)
```

```
muchas veces por 2.
   #Por eso hacemos 2 elevado a la cantidad (** es el exponente)
   factorParaMultiplicar = 2.0**cantidadDeOctavasACorrer
   return frecuenciaBase * factorParaMultiplicar
```

Una solución para permitir codificar escalas. Observamos las modificaciones en la función principal en negrita.

- Permitir que las notas tengan alteraciones (sostenidos)
  Vimos al principio del capítulo que, en realidad, en una octava completa existen 12 sonidos distintos (es decir, 12 teclas de piano), pero que las notas nombran solo a 7 de ellas (las teclas blancas). Para referirse a las teclas negras se agregan "alteraciones" a las notas. En este caso, proponemos agregar sostenidos al código, es decir, la posibilidad de indicar que no nos referimos a la tecla que nombra la nota, sino a la siguiente en el piano. Por ejemplo "do#" se refiere a la tecla negra entre do y re, y "mi#" simplemente a fa.
  Conociendo el valor de frecuencia correspondiente a la nota sin alterar, se puede calcular la frecuencia del sostenido multiplicando por $\sqrt[12]{2}$, es decir, aproximadamente 1.0594630943592953. El significado de este número lo podemos explicar si tenemos en cuenta que entre una nota y la misma de la octava siguiente hay 12 pasos. Luego, aplicar 12 veces la acción de subir un sostenido, debe producir la frecuencia de la nota en la octava siguiente, es decir, debe multiplicar por dos.
  Para esta parte, dado que la información que hay que codificar es si la nota es sostenido o no (es decir, la existencia o no de la alteración, que es, en definitiva, un valor booleano), podemos sugerir que experimenten con el operador `in`.
  Esperamos que observen que, dado un elemento y una secuencia, devuelve `True` si el elemento se encuentra en la secuencia y `False` en caso contrario. En castellano lo podemos traducir como "en" o "está en". Por ejemplo `2 in [1, 2, 3]` produce `True`, mientras que `2 in [1, 3, 4]`, `False`. Lo interesante es que también se puede usar para secuencias de caracteres, de manera que `'u' in "secuencia"` produce `True` y `'f' in "texto"` produce `False`. Luego, detectar si hay que aplicar la alteración de sostenido, se puede resolver preguntando si el código elegido para representarlo (en este caso, utilizamos el símbolo tradicional: #), aparece en la codificación. Otra dificultad que se agrega es la necesidad de eliminarlo del código de la nota antes de decodificar la frecuencia. Para esto, simplemente podemos dar la solución que consiste en utilizar el operador `[:-1]`.

```
def generarMelodiaConSostenidos(partitura):
   secuenciaDeCodigos = separarEnNotas(partitura)
   duracionDeUnTiempo = 1
   tasaDeMuestreo = 44100
   melodia = ab.crearSecuenciaVacia()

   #La codificación es, por ejemplo, "do#^3*4" para identificar una negra
de do sostenido central.
   for codigo in secuenciaDeCodigos:
```

```
        #Utilizamos la funcion partir para separar la duración del nombre
de la nota y la octava
        #Accedemos a la secuencia producida para extraer cada uno.
        notaYDuracion = ab.partir(codigo, '*')
        codigoNotaYOctava = notaYDuracion[0]
        duracionDeLaNota = int(notaYDuracion[1])

        duracionEnSegundos = calcularDuracion(duracionDeLaNota,
duracionDeUnTiempo)

        if codigoNotaYOctava == 'x':
            sonidoDeLaNota = generarSilencio(duracionEnSegundos,
tasaDeMuestreo)
        else:
            notaYOctava = ab.partir(codigoNotaYOctava, '^')
            nombreDeLaNota = notaYOctava[0]
            octava = int(notaYOctava[1])
            haySostenido = '#' in nombreDeLaNota
            if haySostenido:
                nombreDeLaNota = sacarUltimoCaracter(nombreDeLaNota)
            frecuenciaDeLaNota = interpretarFrecuencia(nombreDeLaNota)
            frecuenciaDeLaNota =
aplicarModificacionDeOctava(frecuenciaDeLaNota, octava)
            frecuenciaDeLaNota = aplicarSostenido(frecuenciaDeLaNota,
haySostenido)
            sonidoDeLaNota = a3.generarSonido(frecuenciaDeLaNota, 16000,
duracionEnSegundos, tasaDeMuestreo)

        a3.agregarSecuenciaAlFinal(melodia, sonidoDeLaNota)

    return melodia


def sacarUltimoCaracter(secuencia):
    return secuencia[:-1]

def aplicarSostenido(frecuenciaDeLaNota, haySostenido):
    if haySostenido:
        factor = 2.0**(1.0/12)
        return frecuenciaDeLaNota * factor
    else:
        return frecuenciaDeLaNota
```

Cierre

Cerraremos esta actividad y, por lo tanto, este capítulo, socializando las producciones de los grupos. Les proponemos que busquen algunas melodías de fragmentos de canciones y las traduzcan al sistema de representación que definieron para generarlas con el sintetizador. De la misma manera, pueden explicarles este sistema a otros grupos para intercambiar “partituras” y generar sonidos con los programas de sus compañeros.

Como reflexión final, y para cumplir la promesa que hicimos cuando comenzamos en el capítulo anterior, observamos que, si bien el recorrido por Python fue arduo, valió la pena trabajar con este lenguaje para poder hacer cosas mucho más poderosas que las que

podíamos hacer con Alice. De hecho, resaltamos que con lo que hicieron hasta ahora ya están en buenas condiciones para continuar aprendiendo Python (o incluso, otros lenguajes similares) y resolver una enorme variedad de problemas.

ANEXO III

REDES DE DATOS E INTERNET

Una red de computadoras es un conjunto de dispositivos interconectados con el objetivo de compartir información, recursos y servicios. El conjunto de computadoras, el software de red, los medios y los dispositivos de interconexión forman un sistema de comunicación donde existen emisores, receptores y medios de transmisión.

En este anexo se trabaja en torno a los conceptos básicos de redes, a través de actividades que permiten experimentar y comprender el funcionamiento de una red en general y de Internet en particular. Asimismo, se incluyen algunas actividades que permiten reflexionar sobre los buscadores en Internet.

Anexo 3. Las redes de datos e Internet

Introducción

Una **red de computadoras** es un conjunto de dispositivos interconectados con el objetivo de compartir información, recursos y servicios. El conjunto de computadoras, el software de red, los medios y los dispositivos de interconexión, forman un sistema de comunicación donde existen **emisores**, **receptores** y **medios de transmisión**.

En este anexo se trabaja principalmente en torno a los conceptos básicos de redes, a través de actividades que permiten experimentar y comprender el funcionamiento de una red en general y de Internet en particular. Asimismo se incluyen algunas actividades que permiten reflexionar sobre los buscadores en Internet.

Índice

- Secuencia didáctica 1: Sistema de comunicación
  - Actividad: El teléfono descompuesto, recargado
- Secuencia didáctica 2: Protocolos de comunicación
  - Actividad: ¿Debemos ser políglotas?
  - Actividad: Nos comunicamos con emoticones
  - Actividad: Mensajes sin contexto
- Secuencia didáctica 3: Servicios en Internet
  - Actividad: Venta de boletos
  - Actividad: Servidor web humano
- Secuencia didáctica 4: Las tecnologías de acceso para redes hogareñas
  - Actividad: Dominó de redes
- Secuencia didáctica 5: Identificación de los recursos en Internet
  - Actividad: Enviando mensajes internos
  - Actividad: Enviando mensajes intergrupo
  - Actividad: Inspeccionando la identidad de los dispositivos
- Secuencia didáctica 6: Ruteo y transporte de datos en Internet
  - Actividad: Viajes y tiempos
  - Actividad: Paquetes y tabletas
- Secuencia didáctica 7: Detección de errores
  - Actividad: Verificando la integridad del mensaje
- Secuencia didáctica 8: Búsqueda de información en la Web
  - Actividad: ¿Por qué necesitamos un buscador?
  - Actividad: El orden de los resultados, ¿altera el producto?
- Modelo de evaluación
  - Actividad Integradora 1: Las redes hogareñas y su acceso a Internet
  - Actividad Integradora 2: Programando protocolos
- Glosario

Secuencia didáctica 1: Sistema de comunicación

Bajada

En un **sistema de comunicaciones** el dispositivo que envía el mensaje se conoce como **transmisor** o **emisor** mientras que el dispositivo que lo recibe se conoce como **receptor**.
Para que el mensaje, que físicamente se convierte en una señal, pueda llegar desde el emisor u origen al receptor o destino, es necesario contar con un **medio de transmisión**, el cual puede ser un medio guiado como un cable o no guiado como el aire.
En esta secuencia didáctica se retoma la actividad de la secuencia didáctica 1 del anexo 7, “El teléfono descompuesto”, que propone el intercambio de mensajes entre estudiantes, pero en esta instancia se incluyen distintos **medios de transmisión**.

Objetivos

Que los estudiantes logren:

- Comprender cómo funciona la transmisión de mensajes entre dispositivos conectados a través de una red.
- Entender el concepto de medio de transmisión, sus características y restricciones.

Actividad

Titulo

El teléfono descompuesto, recargado

Objetivos

Que los estudiantes puedan:

- Identificar los elementos que forman parte de un sistema de comunicación.
- Comprender el concepto de **ruido** en la comunicación.

Modalidad de trabajo

Se trabajará en parejas.

Materiales y recursos utilizados

Vasos de plástico unidos de a pares por un hilo, un tubo de cartón largo y los teléfonos celulares de los estudiantes.

Bajada para el aula

Desarrollo

Al comenzar la actividad, organizá a los estudiantes de a parejas y pediles que se dispersen

en el aula asignándoles a cada pareja un medio de transmisión para comunicarse. Los medios disponibles serán: los vasos de plástico unidos por un hilo; el tubo de cartón; Whatsapp, Telegram o el servicio de mensajería que utilicen desde sus celulares; o usando un medio no guiado como el aire.

Una vez establecidas las parejas de estudiantes y el medio de transmisión, indicale a uno de los integrantes de la pareja que le envíe un mensaje a su compañero. Para esto, entregale en un papel varias frases escritas que deberá enviar utilizando el medio de comunicación asignado. El contenido de las frases utilizadas en este momento no es relevante, como tampoco si se repiten para todas las parejas.

Luego podés preguntar a los estudiantes: *¿quiénes son emisores?, ¿quiénes receptores? y ¿cuál es el medio de comunicación que están usando?*. Se espera que en este momento, los estudiantes puedan indicar que:

- el **emisor** es el estudiante que transmite el mensaje,
- el **receptor** es el estudiante que recibe el mensaje y,
- el **medio de transmisión** dependerá en cada caso**:** los vasos unidos por el hilo, el tubo, el aire o el servicio de mensajería usado.

Una vez que todas las parejas hayan logrado transmitir sus respectivos mensajes, se analizarán en forma colectiva las siguientes situaciones:

- *¿Qué ocurre cuando varias parejas de estudiantes se envían simultáneamente mensajes?*
- *¿Todos los medios de transmisión funcionaron de la misma manera?*

Se espera que en esta discusión surja el concepto de **ruido** en la comunicación, como una señal no deseada que se mezcla con la señal útil que se quiere transmitir.

Cierre

A modo de cierre se puede concluir que, de acuerdo al medio de transmisión utilizado, puede o no haber ruido afectando la recepción del mensaje.

En el caso de los mensajes transmitidos por un servicio de mensajería, no hay ruido y los mensajes deberían recepcionarse sin problemas, siempre y cuando los teléfonos celulares utilizados para realizar la actividad cuenten con conexión a Internet. Mientras que en los otros casos se trata de medios que agregan ruido, distorsionando en mayor o menor nivel el mensaje recibido. Por ejemplo, cuando se usa el aire como medio de transmisión, los mensajes se confunden más que en otros casos, si es que todos hablan a la vez. Con los otros dos medios utilizados es muy probable que los mensajes se distorsionen durante la transmisión incorporando por ejemplo un eco. Esto también dependerá del contenido del mensaje: en mensajes largos aumenta la posibilidad de alteración del mismo.

Secuencia didáctica 2: Protocolos de comunicación

Bajada

Dado que existen numerosas tecnologías, fabricantes de dispositivos, y diversidad de sistemas operativos y software, para que los sistemas puedan comunicarse y de esta manera los mensajes puedan enviarse y recepcionarse, es necesario establecer reglas que todos los actores intervinientes conozcan y respeten. Un **protocolo** es justamente el conjunto de reglas que definen los mensajes que se envían, el formato de los mismos y las acciones relacionadas a la recepción y transmisión de un mensaje.
Esta secuencia didáctica propone tres actividades que, en forma gradual acercan el concepto de protocolo de comunicación y los elementos que lo componen.
En la primera actividad se plantea una situación donde surge la necesidad de contar con un protocolo que permita que dos entidades se comuniquen. La segunda actividad propone un juego sencillo que pretende aproximarse al concepto de protocolo de comunicación y los elementos que lo componen. La tercera actividad retoma la actividad “Representación de textos” de la secuencia didáctica 1 del anexo 10, e introduce la necesidad de establecer reglas específicas para la comunicación, que tanto el emisor como el receptor conozcan y apliquen.

Objetivos

Que los estudiantes logren:

- Comprender el concepto de protocolo en los sistemas de comunicación y los elementos que lo componen.

Actividad

Titulo

¿Debemos ser políglotas?

Objetivos

Que los estudiantes puedan:

- Concebir la necesidad de contar con reglas estrictas para establecer una comunicación entre un emisor y un receptor, a través de un lenguaje común.

Modalidad de trabajo

Grupos reducidos.

Materiales y recursos utilizados

Lapiz y papel.

Bajada para el aula

Para comenzar la actividad, planteá a los estudiantes la siguiente situación hipotética a

modo de ejemplo: "*Supongan que son los capitanes de un barco que debe cruzar por un canal y se encuentran con otro barco con bandera rusa que, por su posición, pueden pensar que intenta hacer la misma acción que ustedes".*
Luego, para orientar el tema a desarrollar, preguntá:

- *¿Cuál es el problema que plantea esta situación?*
- *¿Cómo pueden los barcos cruzar el canal sin provocar un accidente naval?*
- *¿Cómo se podría resolver?*

En este momento, dividí a los estudiantes en grupos reducidos y pediles que propongan posibles respuestas para las preguntas dadas.
Luego de unos minutos de debate en cada grupo, realizá una puesta en común para poder analizar las distintas soluciones y propuestas. Se espera que surjan ideas tales como:

- "*El problema es que los capitanes de ambos barcos 'hablan' distintos idiomas y esto podría provocar que no puedan coordinar sus acciones*".
- "*No hay problema, porque ambos se comunican en inglés o usan clave morse"*.

Si bien estas dos respuestas condicionan las siguientes, en ambas surge el mismo concepto: **para comunicarse necesitan un lenguaje en común**.
Luego de establecer esta premisa, se debe resolver qué lenguaje en común se utilizará y cómo se organizarán para cruzar el canal. Aquí pueden surgir ideas tales como:

- "*Ambos capitanes saben que se debe utilizar la clave morse o el inglés, o el lenguaje que haya surgido y coordinan comunicándose en ese lenguaje*".
- *"En realidad, no alcanza con solo poder comunicarse en un mismo lenguaje sino que ambos capitanes deben conocer y aplicar los procedimientos o reglas para cruzar el canal. Es decir, las maniobras de paso y las prioridades de cada uno".*

Entonces, *¿debemos ser políglotas?*. De acuerdo a lo analizado en esta actividad, resulta necesario contar con un lenguaje común con el cual comunicarse, que establezca las reglas de comunicación entre el emisor y el receptor, no siendo vital ser políglota.

Cierre

En esta actividad surge la primera noción de **protocolo de comunicación**, como el conjunto de reglas que se deben cumplir para que la información emitida pueda ser interpretada correctamente al recibirla. Esto involucra la necesidad de establecer un **lenguaje común**, un **formato para los mensajes** que se desean transmitir y un **orden en que los mismos deben ser transmitidos**.
Además de poder sistematizar y enunciar esta primera noción conceptual de protocolo de comunicación, también se puede proponer a los estudiantes que planteen situaciones de la vida cotidiana donde existen protocolos establecidos. Se espera que surjan ideas que vayan desde un simple saludo formal a las normas de tránsito en una ciudad

---

Actividad

Nos comunicamos con emoticones

Objetivos

Que los estudiantes puedan:

- Aproximarse a los conceptos que componen un protocolo de comunicación mediante un juego.

Modalidad de trabajo

Se trabajará en parejas.

Materiales y recursos utilizados

Un juego de fichas con los siguientes emoticones:

Desarrollo

Para comenzar la actividad, dividí a los estudiantes en parejas. Cada estudiante dispondrá de una copia de la Ficha Nº 1. Indicales que por el momento descarten el emoticón  .
El objetivo de esta actividad es proponer un juego de intercambio de mensajes entre los integrantes de las parejas, utilizando las fichas con emoticones. Esta comunicación debe respetar las siguientes reglas:

1. Un integrante de la pareja presenta un emoticón que puede ser:  ,  o  .
2. Si presenta una sonrisa  , el otro integrante responde con una sonrisa  .
3. Si presenta una carita triste  , el otro integrante ofrece ayuda presentando el  .
4. Si presenta un saludo  , el otro integrante responde saludando también:  .
5. Luego se repite desde el paso 1.

Designá a uno de los integrantes de cada pareja para que inicie la comunicación y pedile que intercambie mensajes utilizando las reglas establecidas previamente. Ambos estudiantes deberán tomar nota de los intercambios realizados. Se pedirá que repitan el juego al menos 3 veces usando diferentes fichas iniciales.
En algún momento del juego solicitale al estudiante que inicia la comunicación que incorpore al conjunto de emoticones el correspondiente a  descartado inicialmente. No des ninguna indicación adicional de modo tal que cada pareja decida cómo usarlo. Luego de esta intervención, pediles que repitan el juego 2 veces más.
Cuando todas las parejas finalicen sus intercambios, hacé una puesta en común de las

distintas situaciones presentadas y cómo fueron resueltas por cada equipo.
En este momento, es importante retomar las ideas sobre protocolos de comunicación trabajadas en la actividad anterior y los elementos que lo componen, relacionando los mismos con los elementos intervinientes en el juego. Para esto, se pueden plantear los siguientes interrogantes: *¿observás alguna relación entre los elementos de este juego y un protocolo de comunicación?, ¿cuál es?*. Se espera que los estudiantes hayan observado que los emoticones representan los mensajes posibles a enviar y, que el orden correcto que establece el protocolo está dado por las reglas definidas. A partir de esto, preguntale a los estudiantes:

- *¿Qué pasa si uno de los 2 integrantes de la pareja no respeta las reglas?*
- *¿Qué sucedió cuando se incorporó el emoticón*  *?*
- *¿Alguno de ustedes, al iniciar la comunicación, intentó presentar el emoticón*  *? ¿Por qué?*

Se espera que a partir de las respuestas recibidas, se pueda concluir que:

- Las reglas se deben respetar en forma estricta: si el estudiante que inicia la comunicación presenta un emoticón no establecido por las reglas, el receptor de la misma no sabe qué acción debe tomar. Esto podría dar lugar a que no responda, o responda como le parezca, siendo ambas situaciones no deseables. Se podría, en este momento, relacionar con la primera actividad de esta secuencia didáctica, mostrando los posibles problemas que podrían ocurrir en el caso de los barcos que intentan cruzar el canal, por ejemplo quedarse parado bloqueando el paso, que ambos inicien el cruce en forma simultánea, que uno de los barcos haga una maniobra peligrosa etc.
- Las reglas deben ser conocidas por ambos estudiantes: tanto el emisor como el receptor deben comprender el significado de la regla para poder aplicarla.

Cierre

A modo de cierre, reafirmá el concepto de protocolo de comunicación, como el conjunto de reglas y la estructura de mensajes, que permiten que dos o más entidades se comuniquen.
En este punto, podés hacer una analogía con las redes de computadoras: los protocolos determinan los mensajes a transmitir por la red, el orden en que los mismos deben transmitirse, qué hacer luego de transmitir y/o recibir un mensaje, entre otras acciones.

Ficha N°1

Ficha para el estudiante

Nos comunicamos con emoticones

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 2: Protocolos de comunicación

Bajada

Mensajeate con tu compañero, usando los siguientes emoticones y las reglas de intercambio propuesta y, anotá cada intercambio realizado. Repetí el juego al menos 3 veces usando diferentes fichas iniciales y luego seguí las indicaciones de tu docente.

Emoticones a utilizar:

Reglas de intercambio:

1. Un integrante de la pareja presenta un emoticón que puede ser: , o .
2. Si presenta una sonrisa , el otro integrante responde con una sonrisa .
3. Si presenta una carita triste , el otro integrante ofrece ayuda presentando el .
4. Si presenta un saludo , el otro integrante responde saludando también: .
5. Luego se repite desde el paso 1.

Actividad

Titulo

Mensajes sin contexto

Objetivos

Que los estudiantes puedan:

- Acercarse al concepto de codificación de mensajes en un protocolo de comunicación mediante el uso de tablas de representación de texto.
- Reconocer la necesidad de establecer las reglas del protocolo a utilizar antes de comenzar la interacción.

Modalidad de trabajo

Se trabajará en parejas.

Materiales y recursos utilizados

Las tablas de codificación XRAYA y RAYASGRIEGAS de la actividad “Representación de textos”, de la secuencia didáctica 1 del anexo 10.

Tablas de codificación:

(reservado): -					
A: X	**F**:XX-	**K**:X-XX	**O**:X----	**T**:X-X-X	**Y**:XX-X-
B: X-	**G**:XXX	**L**:XX--	**P**:X---X	**U**:X-XX-	**Z**:XX-XX
C: XX	**H**:X---	**M**:XX-X	**Q**:X--X-	**V**:X-XXX	**.** :XXX--
D: X--	**I**: X-- X	**N**:XXX-	**R**:X--XX	**W**:XX---	**,** :XXX-X
E: X-X	**J**: X-X-	**Ñ**:XXXX	**S**:X-X--	**X**:XX--X	:XXXX-

Tabla 1- XRAYA

Letra del alfabeto griego	Representación
(reservado)	-
α	X
β	X-
γ	XX

δ	X--

Tabla 2 - RAYASGRIEGAS

Bajada para el aula

Para comenzar la actividad entregá la Ficha Nº 2 a los estudiantes y retomá la actividad "Representación de textos" del anexo 10, donde se utilizaron 2 codificaciones distintas, denominadas XRAYA y RAYASGRIEGAS, para representar textos. Recordá a la clase la finalidad de dichas tablas y contales que en esta actividad se emplearán para transmitir mensajes codificados en el aula.

En un primer momento, solicitá a los estudiantes que se agrupen de a pares, luego pedile a cada integrante que escriba en un papel 3 mensajes compuestos por los símbolos "X" y "-". Estos mensajes codificados luego serán intercambiados entre los integrantes de cada pareja. Todos los estudiantes dispondrán de ambas tablas para codificar (el emisor) y decodificar (el receptor) sus mensajes. De ser posible, cada integrante de la pareja debe ubicarse en el aula de forma tal que los receptores de los mensajes no puedan ver cómo codifica los mensajes el emisor. Indicá que el mensaje puede ser cualquier texto que pueda codificarse con alguna de las dos tablas. Por ejemplo, "CASA" o "δαβ".

Inicialmente no se darán indicaciones a los estudiantes sobre qué tabla utilizar para codificar y decodificar los mensajes. En caso que se plantee la duda en el aula, se puede proponer que quienes codifiquen el mensaje lo hagan con una tabla al azar, elegida por ejemplo lanzando una moneda. En cualquier caso, el receptor no deberá saber qué tabla se usó para la codificación.

Luego que cada integrante del equipo escriba sus mensajes, debe entregárselo a su pareja para que ésta los decodifique. Los estudiantes compararán ambos mensajes para verificar si hay coincidencias entre el mensaje original y el decodificado.

Es muy probable que no haya coincidencias entre todos los mensajes enviados y los decodificados por el receptor. Por ello, preguntale a los estudiantes acerca de las causas de esta situación: *¿qué pasa si se permite que el emisor y el receptor en un sistema de comunicaciones puedan elegir distintas codificaciones a la hora de enviar un mensaje?.*

En este caso, si un protocolo de comunicación habilita a que el emisor y el receptor usen distintas codificaciones, resultará en que el receptor podría obtener, como resultado de la decodificación un mensaje distinto al enviado por el emisor.

Cierre

A modo de cierre, explicá que la actividad propuesta es una analogía de la codificación de mensajes utilizada en los protocolos de comunicación de redes y que para que la comunicación sea exitosa ambos integrantes del grupo, emisor y receptor, deberán previamente ponerse de acuerdo sobre qué tabla de codificación utilizar o si es posible usar ambas. En caso de utilizar las dos tablas, el emisor deberá indicar en cada mensaje enviado la tabla utilizada para que el receptor pueda posteriormente decodificarlo.

La **codificación de los mensajes** es una parte constitutiva de la **definición de un protocolo de comunicación de redes**.

Ficha N°2

Ficha para el estudiante

Mensajes sin contexto

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 2: Protocolos de comunicación

Bajada

A) Codificá y envía a tu compañero 3 mensajes compuestos por los símbolos “X” y “-” utilizando las tablas XRAYA y RAYASGRIEGAS de la actividad “Representación de textos”, de la secuencia didáctica 1 del anexo 10
B) ¿El mensaje codificado que enviaste es el mismo que recibió tu compañero?, ¿por qué?.

Tablas de codificación:

(reservado): -					
A: X	**F**:XX-	**K**:X-XX	**O**:X----	**T**:X-X-X	**Y**:XX-X-
B: X-	**G**:XXX	**L**:XX--	**P**:X---X	**U**:X-XX-	**Z**:XX-XX
C: XX	**H**:X---	**M**:XX-X	**Q**:X--X-	**V**:X-XXX	**.** :XXX--
D: X--	**I**: X-- X	**N**:XXX-	**R**:X--XX	**W**:XX---	**,** :XXX-X
E: X-X	**J**: X-X-	**Ñ**:XXXX	**S**:X-X--	**X**:XX--X	:XXXX-

Tabla 1- XRAYA

Secuencia didáctica 3: Servicios en Internet

Bajada

Internet es un conjunto descentralizado de redes interconectadas que usan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen formen una red lógica única de alcance mundial.
La mayoría de los servicios de Internet se implementan siguiendo el **modelo cliente-servidor**. El mismo consta de al menos dos aplicaciones: un cliente y un servidor. El **cliente** es quien realiza solicitudes y el **servidor** es el que responde a las mismas. Si pensamos en la vida real, un ejemplo de un sistema que respeta este modelo, podría ser una ventanilla de atención de trámites, donde hay un empleado esperando para atender a las personas que lleguen a hacer un trámite, cumpliendo el rol de servidor, y personas que llegan demandando atención, que cumplen el rol de clientes.
Uno de los servicios más populares de Internet que sigue el modelo cliente-servidor, es la **Web o WWW** (de sus siglas en inglés World Wide Web), que permite acceder a documentos vinculados entre sí a través de hiperenlaces. Estos documentos, conocidos como páginas web, pueden contener textos, imágenes, videos u otros contenidos multimediales y se visualizan con un navegador, el **cliente web**. Las páginas web están contenidas en sitios y son los **servidores web** quienes permiten su acceso y distribución. El acceso a una página web involucra una serie de pasos, en general no perceptibles para el usuario, que forman parte del protocolo de comunicación de la web: **HTTP** (de sus siglas en inglés, Hypertext Transfer Protocol).
Esta secuencia didáctica propone dos actividades que abordan los conceptos mencionados previamente. La primera actividad tiene como propósito acercar los conceptos básicos del modelo cliente-servidor mediante el desarrollo de una aplicación implementada en Alice, que simula la venta de boletos de pasajes aéreos. La segunda actividad aborda el funcionamiento del protocolo HTTP, pilar fundamental de la Web.

Objetivos

Que los estudiantes logren:

- Comprender el funcionamiento del modelo cliente-servidor.
- Experimentar y comprender el funcionamiento del servicio web y el protocolo HTTP, como un ejemplo del funcionamiento de los servicios sobre Internet .

Actividad

Titulo

Venta de boletos

Objetivos

Que los estudiantes puedan:

- Identificar los componentes intervinientes en el modelo cliente-servidor mediante la experimentación con una aplicación simple.
- Adaptar una aplicación en Alice que simula de manera simplificada el funcionamiento del modelo cliente-servidor, propio de los servicios de Internet.

Modalidad de trabajo

Actividad individual.

Materiales y recursos utilizados

Computadoras con Alice 2.4. Y el programa servidor-web.a2w, disponible en: http://linti.unlp.edu.ar/programar/servidor-web.a2w.

Bajada para el aula

Esta actividad está organizada en 3 momentos distintos. Se trabajará con un programa realizado en Alice 2.4. disponible en: http://linti.unlp.edu.ar/programar/servidor-web.a2w, el cual muestra la interacción entre 3 personajes: uno de ellos simula a un servidor y los dos restantes son los clientes de un servicio de venta de pasajes aéreos. El programa provisto para este ejercicio es un prototipo que contempla las siguientes restricciones: una lista fija de destinos posibles, el destino ingresado debe coincidir exactamente con alguno de los almacenados en esta lista respetando el uso de mayúsculas y minúsculas y una cantidad máxima de 40 pasajes para la venta por cada destino.

Momento 1: analizando la simulación

Para comenzar la actividad, pedí a los estudiantes que descarguen el programa servidor-web.a2w en sus computadoras, lo ejecuten y analicen cómo se comportan los personajes que componen la simulación. Preguntales: *¿qué mensajes se intercambian?, ¿quién los envía?, ¿quién los recibe y en qué orden?.*
Podés pedirle a los estudiantes que registren lo observado en una tabla donde se relacione cada personaje de la aplicación con los mensajes que envía y recibe.
Se espera que los estudiantes observen que se trata de un programa que cuenta con 3 personajes: dos personas, Juan y Ada, interesados en viajar y una vendedora de boletos, Shannon, que siempre está a la espera de clientes que se acerquen a comprar sus boletos. La simulación comienza cuando uno de los clientes se acerca a comprar un boleto y luego de un intercambio de mensajes con la vendedora se registra o no la venta de acuerdo a la disponibilidad de pasajes al destino solicitado. Luego se acerca el siguiente cliente y realiza el mismo intercambio de mensajes. Las preguntas realizadas por la vendedora son siempre las mismas y se hacen siempre en el mismo orden.

Momento 2: agregando un nuevo cliente a la simulación

Solicitá a los estudiantes que modifiquen el programa dado agregando un nuevo cliente que se integre a la simulación. Este nuevo cliente tiene un comportamiento similar a los restantes.

Se espera que los estudiantes analicen el programa y observen que existe una lista de clientes donde se encuentran los personajes Juan y Ada, y deduzcan que deben crear un nuevo cliente, agregando un nuevo objeto, e insertarlo en la lista de clientes para poder integrarlo a la simulación.
En http://linti.unlp.edu.ar/programar/servidor-web-docente.a2w, podés encontrar una solución implementada.

Momento 3: clientes, servidores y el protocolo de comunicación

Por último, realizá una puesta en común con la clase y dialogá a partir de estas preguntas:

- *¿Podría haber más clientes?*
- *¿Cuáles son los mensajes y las reglas implicadas?.*
- *¿Cuál es la función del personaje Shannon?*

Se espera que los estudiantes observen que Shannon queda siempre a la espera de un requerimiento de cualquiera de los clientes. En esta aplicación podríamos agregar más clientes y todos tendrían la misma lógica, harían lo mismo. Del análisis de la simulación, los estudiantes deberán deducir que las reglas del protocolo, en el caso de una venta exitosa, establecen la siguiente secuencia de comunicaciones:

1. Un cliente inicia la comunicación al decir *"Quiero un pasaje"*.
2. Shannon pregunta el destino*: "Hola, ¿a qué lugar viaja? (Ezeiza o Salta)".*
3. El cliente indica el destino: *"Salta"* o *"Ezeiza".*
4. Si cuenta con pasajes para ese destino, Shannon consulta la cantidad: *"¿Cuántos pasajes desea comprar?".* Si no cuenta con pasajes para ese destino, Shannon informa: "No tenemos pasajes para 'el destino solicitado' ".
5. El cliente indica la cantidad con un número.
6. Si se cuenta con la cantidad suficiente Shannon da por finalizada la compra e informa: "*Ya registré su compra".* En caso contrario, Shannon informa: *"No tenemos 'esa cantidad' de asientos libres para 'el destino solicitado' ".*
7. Luego Shannon solicita que pase el siguiente cliente: *"Siguiente".*

Analizá junto con los estudiantes que las frases expresadas en este diálogo, incluída las respuestas de los clientes, como la cantidad de pasajes a comprar y el destino, son los mensajes utilizados por este protocolo.
En este momento el docente podrá explicar que Shannon cumple el rol de servidor, dado que atiende las solicitudes de pasajes de Juan y Ada. A su vez, Juan y Ada, se comportan como clientes puesto que su función es solicitar pasajes.

Cierre

A modo de cierre, podés explicar a los estudiantes que típicamente en Internet se utilizan aplicaciones que siguen los lineamientos de un modelo denominado **cliente-servidor** y que funciona de manera similar al presentado en la simulación. En este modelo, una aplicación, denominada **cliente,** en el caso de esta actividad representada por los personajes Juan y Ada, es quien realiza los requerimientos o pedidos y el **servidor**, representado por el personaje Shannon, es quien resuelve los pedidos de los clientes y les envía respuestas, estando disponible todo el tiempo para recibir pedidos.
En las aplicaciones desarrolladas bajo el modelo cliente-servidor pueden existir tantos clientes como el servidor pueda atender.

En Internet, la mayoría de los servicios se implementan bajo este modelo. El caso más paradigmático es la Web, donde el navegador de Internet es el cliente, y cada vez que queremos acceder a una página, el mismo envía un requerimiento al servidor donde está alojada dicha página. El servidor es una aplicación que atiende y responde estos requerimientos. El protocolo de comunicación utilizado entre un cliente web y un servidor web es HTTP que será objeto de la siguiente actividad.

Actividad

Titulo

Servidor web humano

Objetivos

Que los estudiantes puedan:

- Aproximarse al funcionamiento básico de la Web y del protocolo HTTP mediante un juego simple de roles.
- Analizar el funcionamiento de un protocolo similar a HTTP y sus posibles mejoras.

Modalidad de trabajo

Grupos de 4 estudiantes.

Materiales y recursos utilizados

Se necesitará un juego de tarjetas con solicitudes y respuestas por cada 4 estudiantes.

Tarjetas de solicitudes:

Solicitud: Página Ada Lovelace	Solicitud: Imagen Ada_Lovelace.jpg	Solicitud: Imagen Ada_Lovelace_2.jpg
Solicitud: Página Margarita Manterola	Solicitud: Video Margarita_Manterola.mp4	Solicitud: Video Margarita_Manterola2.mp4
Solicitud: Página Victoria Bajar	Solicitud: Imagen Victoria_Bajar.jpg	Solicitud: Video Victoria_Bajar.mp4

Tarjetas de respuestas:

Respuesta: Página Ada Lovelace	Respuesta: Imagen Ada_Lovelace.jpg	Respuesta: Imagen Ada_Lovelace_2.jpg
Respuesta: Página Margarita Manterola	Respuesta: Video Margarita_Manterola.mp4	Respuesta: Video Margarita_Manterola2.mp4
Respuesta: Página Victoria Bajar	Respuesta: Imagen Victoria_Bajar.jpg	Respuesta: Video Victoria_Bajar.mp4

Bajada para el aula

Esta actividad plantea un juego de roles, donde algunos estudiantes tendrán el rol de servidores web y otros el de clientes web. Cada servidor tiene 3 páginas para entregar. El contenido de las mismas se compone según el siguiente esquema:

- Página Ada Lovelace: texto y dos imágenes.
- Página Margarita Manterola: texto y dos videos.
- Página Victoria Bajar: texto, una imagen y un video.

Los clientes podrán solicitar dichas páginas con las tarjetas de solicitudes y los servidores responderán a cada solicitud con las tarjetas de respuestas.

Antes de comenzar la actividad, se deberán recortar las tarjetas de solicitudes y respuestas por la línea punteada de manera tal de separar cada juego de tarjetas. Se debe tener un juego de estas tarjetas por cada 4 estudiantes.

Para dar comienzo a la actividad formá grupos de 4 estudiantes, donde un estudiante cumplirá el rol de servidor y los otros tres el rol de clientes. Es importante que ninguno de los estudiantes conozca previamente cuál es el rol asignado: a un estudiante se le entregarán 3 juegos de respuestas y a los restantes 3 estudiantes un juego de tarjetas de solicitud a cada uno.

Luego de entregar todas las tarjetas, podés preguntar si identifican cuál es su rol en función de las tarjetas recibidas. Se espera que aquellos estudiantes que recibieron las tarjetas de solicitud deduzcan que son clientes y aquellos que recibieron tarjetas de respuesta, servidores.

Cada estudiante deberá recortar las tarjetas de manera que quede una única petición o una única respuesta en cada porción de papel.

Luego, indicale a los estudiantes que quienes son servidores se queden en sus bancos, mientras que cada estudiante cliente debe realizar las siguientes acciones: acercarse al servidor de su grupo, entregar una solicitud, esperar la respuesta del servidor y volver a su banco, este proceso lo repite tantas veces como solicitudes tenga que entregar. Luego, preguntar: *¿cuál creen que es la primera solicitud que se debería enviar?*. Frente a este desafío, es probable que los estudiantes no encuentren fácilmente una respuesta, aunque posiblemente se aproximen a la idea de que en primer lugar debería ir la solicitud de la página completa y luego las restantes, recordando algunas experiencias de navegación en Internet en las que las páginas se van completando y esto es perceptible sobre todo cuando la conexión es lenta. También puede indagarse sobre si luego de solicitar la página completa el cliente debe seguir un orden en particular para las restantes solicitudes. Aquí se espera que respondan que para el resto de los elementos es indistinto el orden de las

peticiones.
Reflexionar junto con los estudiantes que el modelo que estamos siguiendo es el cliente-servidor, donde los clientes son los que inician las acciones de petición de páginas o recursos y los servidores siempre están a la espera de peticiones para resolver y responder.
La actividad planteada, es una analogía del funcionamiento de la versión 1.0 del protocolo HTTP, que es el fundamento de la Web. Las entidades que toman el rol de cliente y hacen las solicitudes son los **navegadores web** y las que toman el rol de servidor, atendiendo las solicitudes, son los **servidores web**. En primer lugar el navegador solicita una página y luego de procesarla, solicita los recursos que la misma requiere, es decir, todos las imágenes, videos y otros recursos necesarios para su visualización en el navegador.
Finalmente planteá los siguientes interrogantes:

- *¿Cuántas solicitudes se realizan por cada página requerida?.*
- *¿Se les ocurre alguna manera de realizar menos peticiones por página?, ¿cómo?*

Se espera que los estudiantes hayan observado que por cada página solicitada tienen que hacer 3 peticiones y que si la página incluyera más recursos, la cantidad de peticiones se incrementa. Con tu ayuda podrían pensar alternativas para hacer menos peticiones y mejorar la comunicación, por ejemplo entregar al servidor todas las solicitudes relacionadas a una misma página juntas, en lugar de a una a la vez y repitiendo las acciones: acercarse, entregar una solicitud y volver a su banco para cada solicitud. De esta manera mejorarían los tiempos de respuesta, resultando más rápida la comunicación. Para ello, es necesario introducir algunos cambios en las reglas de la comunicación, es decir **modificar el protocolo de comunicación entre los clientes y el servidor**. En este momento proponeles a los estudiantes que realicen una variación al protocolo propuesto: *un estudiante del grupo clientes entregará a su servidor todas juntas las tarjetas de petición vinculadas a una página, y el servidor a su vez le entregará todas las respuestas juntas*.
Luego, podés preguntar si observan mejoras con estos cambios en el protocolo. Se espera que los estudiantes hayan observado que han realizado menos acciones dado que se han acercado al servidor una sola vez, han entregado todas sus solicitudes juntas y han vuelto a su banco, ahorrando esfuerzo y tiempo.
En este punto se puede reflexionar junto con los estudiantes que cada petición involucra demoras propias de la comunicación de redes, en el ejemplo esas demoras están representadas con acciones como “regresar al banco antes de hacer una nueva solicitud al servidor”.

Cierre

A modo de cierre podés explicar que la primera versión del protocolo de esta actividad, es similar al protocolo HTTP 1.0, donde se envían las solicitudes al servidor web y se reciben las respuestas, de a una por vez. La primera versión del protocolo HTTP utiliza para cada petición una conexión nueva. Es decir, que cuando un navegador web solicita una página que contiene varias imágenes, videos u otros recursos, el servidor web establece una conexión distinta por cada uno de estos elementos a enviar. En el caso de la actividad planteada esta nueva conexión está simbolizada con la acción de entregar una solicitud acercándose al banco del servidor.
En las versiones posteriores del protocolo HTTP, el navegador web envía a través de una única conexión, todas las solicitudes relacionadas a la página que quiere mostrar y recibe las respuestas desde el servidor en una sola conexión.

Asimismo, el docente podrá comentar que la mayoría de los protocolos de redes de datos y de Internet en particular, siguen el modelo cliente-servidor, ejemplo de ello son los protocolos de servicios de correo electrónico, transferencia de archivos, etc. Se tomó el protocolo HTTP a modo de ejemplo, por ser el protocolo más popular de Internet.

¿Sabés quiénes son Ada Lovelace, Margarita Manterola y Victoria Bajar?: Estas 3 mujeres fueron personas muy destacadas en las Ciencias de la Computación. **Ada Lovelace** fue una matemática y escritora británica que desarrolló el primer algoritmo destinado a ejecutarse en un computadora, por lo cual es considerada la primera programadora. Se denominó **Ada** a un lenguaje de programación en su honor. **Margarita Manterola** es una programadora argentina quien se ha convertido en una de las pocas desarrolladoras mujer del proyecto Debian, distribución de GNU/Linux utilizada y desarrollada a nivel mundial. Participa también en el programa Debian-Women que promueve la participación de mujeres en la programación enfocándose en el desarrollo del proyecto Debian. **Victoria Bajar** fue la primera graduada de la carrera Computador Científico en la Argentina. Realizó importantes aportes para el desarrollo computacional argentino trabajando con la primera computadora traída a la Argentina con fines científicos, Clementina. Posteriormente se mudó a México donde se abocó al desarrollo de currículas de carreras de nivel superior del área informática.

Ficha N°3

Ficha para el estudiante

Servidor web humano

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 3: Servicios en Internet

Bajada

Recortá por las líneas punteadas las fichas del juego “Solicitud” o “Respuesta” de acuerdo a lo indicado por tu docente y utilizalas siguiendo las consignas que te darán.

Fichas “Solicitud”:

Solicitud: Página Ada Lovelace	Solicitud: Imagen Ada_Lovelace.jpg	Solicitud: Imagen Ada_Lovelace_2.jpg
Solicitud: Página Margarita Manterola	Solicitud: Video Margarita_Manterola.mp4	Solicitud: Video Margarita_Manterola2.mp4
Solicitud: Página Victoria Bajar	Solicitud: Imagen Victoria_Bajar.jpg	Solicitud: Video Victoria_Bajar.mp4

Fichas “Respuesta”:

Respuesta: Página Ada Lovelace	Respuesta: Imagen Ada_Lovelace.jpg	Respuesta: Imagen Ada_Lovelace_2.jpg
Respuesta: Página Margarita Manterola	Respuesta: Video Margarita_Manterola.mp4	Respuesta: Video Margarita_Manterola2.mp4
Respuesta: Página Victoria Bajar	Respuesta: Imagen Victoria_Bajar.jpg	Respuesta: Video Victoria_Bajar.mp4

Secuencia didáctica 4: Las tecnologías de acceso para redes hogareñas

Bajada

Esta secuencia didáctica propone una actividad que permite trabajar conceptos relacionados a las redes hogareñas y las tecnologías asociadas.
Se propone una única actividad que, a través de un juego, analiza los distintos dispositivos que componen una red hogareña y la interconexión de los mismos.

Objetivos

Que los estudiantes logren:

- Reconocer los elementos que componen una red hogareña.
- Identificar los dispositivos que permiten interconectar redes y acceder a Internet.

Actividad

Titulo

Dominó de redes

Objetivos

Que los estudiantes puedan:

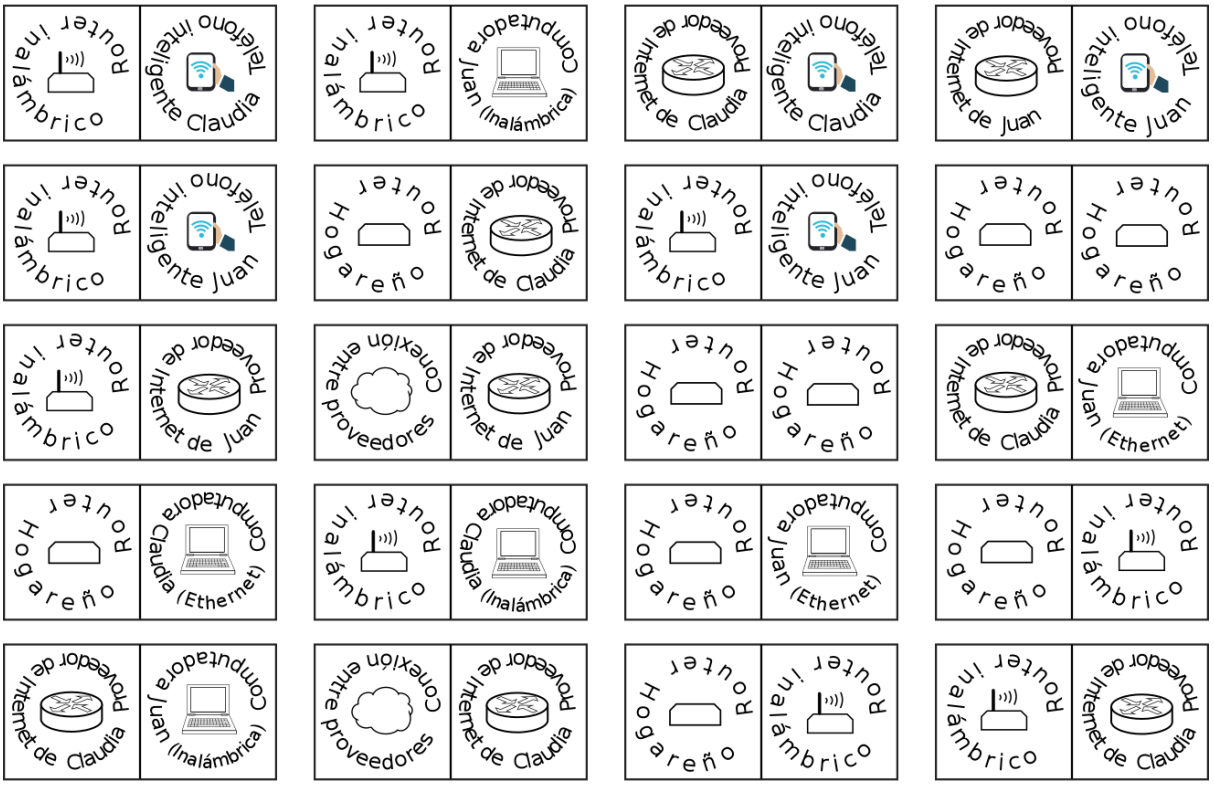
- Identificar, a través de un juego de dominó con fichas especiales, los distintos recursos necesarios para una comunicación a través de Internet.
- Establecer la secuencia que posibilita la comunicación entre los extremos.

Modalidad de trabajo

Se trabajará en parejas.

Materiales y recursos

Un juego de fichas de dominó por cada pareja de estudiantes:



Antes de comenzar el juego, realizá una puesta en común sobre los dispositivos involucrados en una red y su interconexión al momento de realizar una comunicación. Para esto, se puede plantear que cada vez que se realiza una comunicación a través de Internet, por ejemplo, desde una computadora o teléfono inteligente, los datos pasan por una serie de dispositivos y/o puntos de conexión, algunos de los más relevantes son los siguientes:

1. El celular o computadora donde se generan los datos en formato digital: dispositivo del emisor.
2. Un **router** inalámbrico o con cables, dependiendo de qué dispositivo se utilice y cómo se conecte. El router enviará los datos recibidos al **proveedor de servicios** o ISP (por sus siglas en inglés de *Internet Service Provider)* del emisor.
3. El ISP cuenta a su vez con una red con routers y otros dispositivos (que no se tendrán en cuenta en esta actividad para reducir su complejidad).
4. Los datos llegarán al proveedor de Internet, ISP, del receptor.
5. De allí al router del receptor.
6. Y finalmente al dispositivo del receptor.

Presentar el juego, indicando que el mismo sigue reglas similares al dominó, donde cada ficha menciona 2 recursos de red por los que pueden pasar los datos en una comunicación entre dos extremos. Las fichas que mencionan un mismo recurso pueden conectarse entre ellas, por ejemplo las fichas “Computadora Juan - Router inalámbrico” y “Router inalámbrico - Router” se pueden conectar.

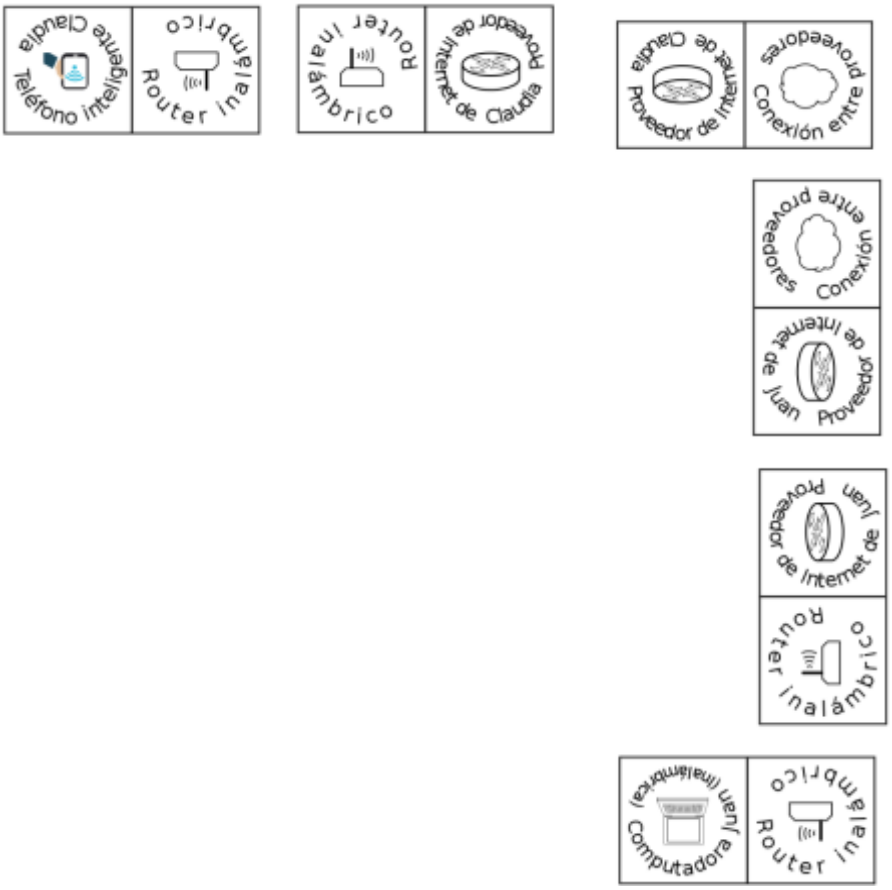
Este juego tiene una restricción adicional: el tablero final debe representar el camino por el que pasan los datos en una comunicación a través de Internet. Existen distintas combinaciones correctas y también fichas que no son válidas. Las fichas con 2 routers pueden utilizarse como las fichas dobles del dominó para crear bifurcaciones.

El disparador de esta partida podría ser una situación como la siguiente: "*Claudia está en su casa y le envía un mensaje a su amigo Juan que vive en Brasil a través de la mensajería de Facebook".*

La misión de los jugadores será jugar una partida de dominó, armando el camino que deberán recorrer los datos para que Claudia se comunique con su amigo en Brasil. El primer jugador que complete el tablero resolviendo la comunicación pedida, es el ganador.

Durante el desarrollo del juego, acompañá a los estudiantes en la elección de las fichas que pueden usar para construir el camino, recordando la funcionalidad de cada uno de los recursos y cuál es el rol que cumple este recurso en la comunicación. Podés advertir que la conexión entre las distintas fichas representa el camino que siguen los datos al intercambiar información entre dos computadoras o teléfonos inteligentes a través de Internet.

Una posible solución del juego es:



Cierre

A modo de cierre, explicale a los estudiantes que en una comunicación en Internet participan múltiples dispositivos específicos de redes y proveedores de servicios o ISP, además del emisor y receptor.

En el juego de dominó resuelto, se simuló el envío de un mensaje a través una red social desde un teléfono inteligente o computadora, y se identificaron algunos de los dispositivos que participan en la comunicación, a través de la red, junto con los ISP de cada extremo, emisor y receptor, usando diferentes routers para encaminar los mensajes desde un extremo al otro.

Ficha N°4

Dominó de redes

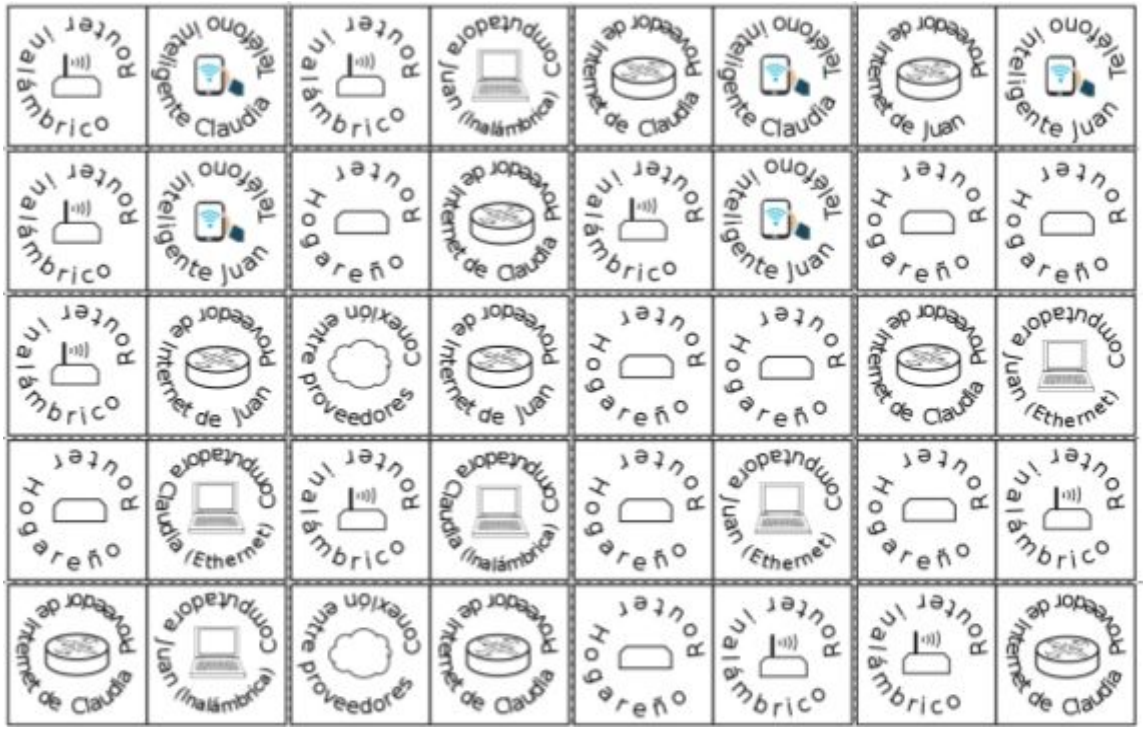
Servidor web humano

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 4: Las tecnologías de acceso para redes hogareñas

Bajada

¡Los invitamos a jugar un juego similar al Dominó tradicional!. Aquí, cada ficha contiene 2 recursos de red.

Recortá por las líneas punteadas las fichas del dominó. Cada una representa un par de recursos que participan en la comunicación. Seguí las indicaciones dadas en clase para resolver el juego.



Secuencia didáctica 5: Identificación de los recursos en Internet

Bajada

Los dispositivos conectados a Internet necesitan ser identificados de forma unívoca. Para ello se usan las **direcciones IP** asignadas a su placa de red. Estas direcciones son administradas por organismos internacionales y permiten tanto identificar a la red como al dispositivo en particular dentro de esa red.
Además cada placa de red tiene una **dirección MAC***,* que es un identificador único de dicha placa de red.
La dirección IP es lo que se conoce como dirección lógica y la dirección MAC es lo que se conoce como dirección física.
Sin embargo, cuando las personas se refieren a los dispositivos u otros recursos lo hacen a través de nombres, que son más fáciles de recordar. Para ello, existe un servicio que traduce los nombres de los dispositivos a su correspondiente dirección IP. De esta tarea se encarga el servicio de **DNS** (de sus siglas en inglés, *Domain Name System*).
Esta secuencia didáctica está integrada por 3 actividades que proponen trabajar sobre los distintos mecanismos para referenciar recursos en una red.
En la primera actividad de esta secuencia, se simula el comportamiento del protocolo de resolución de direcciones **ARP** (de sus siglas en inglés, *Address Resolution Protocol*), el cual dada la dirección IP de un dispositivo permite obtener su dirección MAC. En la segunda actividad, se simula la resolución de nombres a través del servicio DNS, introduciendo por otro lado la noción de decisión de ruteo. Por último, en la tercera actividad, se relacionan los conceptos vistos con los dispositivos disponibles en la escuela.

Objetivos

Que los estudiantes logren:

- Comprender la forma en que se identifica unívocamente un recurso en Internet, desde una computadora hasta una página web.
- Comprender el concepto de identificación física de un dispositivo en la red.
- Acercarse al funcionamiento básico de algunos dispositivos de red como el **hub**, el **switch** y el **router**.

Actividad

Titulo

Enviando mensajes internos

Objetivos

Que los estudiantes puedan:

- Aproximarse al funcionamiento del protocolo ARP que permite obtener la dirección física o MAC a partir de la dirección lógica o IP.
- Acercarse al funcionamiento básico de un hub y un switch.

Modalidad de trabajo

Actividad con toda la clase.

Materiales y recursos utilizados

Se trabajará con la tabla de correspondencias “DNI-Ubicación en el aula”, una por cada estudiante, y las tarjetas de mensajes, una por cada mensaje a enviar.

Tabla de correspondencias:

Correspondencias DNIs - Ubicación en el aula	
DNI del estudiante	**Ubicación del estudiante**
	F: **C:**
	F: **C:**

Tarjetas de mensajes:

Mensaje	
Dirección del Emisor (origen)	**F:** **C:**
Dirección del Receptor (Destino)	**F:** **C:**
Mensaje	

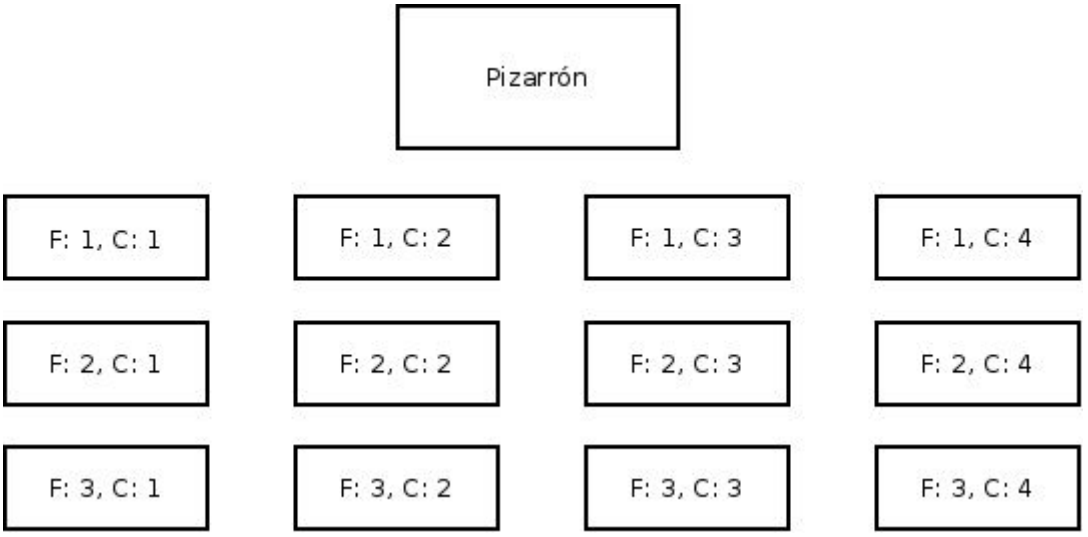
Desarrollo

En─esta actividad, los estudiantes se enviarán mensajes entre ellos identificando al destinatario mediante su DNI. Para que el mensaje llegue a destino, deberán obtener la información de ubicación dentro del aula, es decir en qué fila y columna se encuentra. Se considerará al DNI como la “**dirección lógica**” y a la información de su ubicación, expresada como fila (F) y columna (C), como la “**dirección física**”. Habrá un mensajero por aula, encargado de entregar los mensajes.

La actividad consiste en un juego de roles: hay un estudiante en particular al que se le asigna el rol de mensajero y representa al dispositivo, hub o switch, que interconecta las computadoras, formando una red local y, los restantes estudiantes tienen el rol de las computadoras que envían y reciben mensajes.

Para dar comienzo a la actividad, debes elaborar un mapa del aula enumerando los bancos del aula por fila y columna, de la forma detallada en la siguiente figura, y solicitarás a cada

estudiante que registre su número de fila y columna y lo anote en una hoja. El mapa del aula solamente será conocido por el mensajero.



Luego, elegí el mensajero del aula, encargado de entregar los mensajes desde el estudiante origen al estudiante destino. Para llevar este proceso adelante, el mensajero usará el mapa del aula.

Para dar comienzo al intercambio de mensajes, pedile a los estudiantes, excepto al mensajero, que escriban su número de DNI en un papelito y luego juntá todos los papelitos en una bolsa. Cada estudiante, a excepción del mensajero, tomará un papel de la bolsa, sin mirar. En el caso que un estudiante saque su propio papelito deberá devolverlo a la bolsa y tomar otro. Este papel contendrá el destinatario del mensaje a enviar.

A continuación, solicitá a los estudiantes que completen la tarjeta de mensaje con la información con la que cuentan: DNI del destinatario y el mensaje que desean enviar. Preguntarles*: ¿es posible completar la tarjeta de mensaje?, ¿falta algún dato?.*

Dado que cada estudiante solo conoce su propia ubicación en el aula, se espera que respondan que con la información que disponen, no pueden completar la tarjeta de mensaje. La dirección del destino no puede ser completada ya que solo tienen el DNI del destinatario y no saben dónde está ubicado. En este punto, podés enfatizar que se desconoce la lógica con la que fueron asignadas las direcciones físicas, fila y columna, con lo cual no se puede inferir.

En este momento, deberá surgir la idea que es necesario averiguar la dirección física del destinatario del mensaje, fila y columna en el aula. Podés ayudar a resolver este problema, para ello a modo de ejemplo, elegí a un estudiante para que envíe su mensaje realizando lo necesario para averiguar los datos que necesita. Con tu ayuda, el estudiante elegido completará la tarjeta de mensaje y preguntará en voz alta: *¿quién tiene el número de DNI del destinatario?*. Esto debería provocar que el estudiante con dicho DNI responda. De esta manera, el estudiante interesado en enviar el mensaje, completará su tabla de correspondencia “DNI-ubicación en el aula” agregando los datos de su compañero. Luego, completará la tarjeta del mensaje y se la entregará al mensajero.

En este momento, podés preguntar: *¿el mensajero puede entregar directamente el mensaje?*. Se espera que los estudiantes consideren que el mensajero conoce el mapa del

aula, es decir, sabe cómo acceder a cada dirección física y puede hacer la entrega del mensaje.
Pedile a los estudiantes que, de a uno y en forma ordenada, envíen sus mensajes repitiendo los pasos realizados para el envío del primer mensaje. Esto permitirá que todos experimenten el proceso y que al terminar el mismo, dispongan de información en su tabla de correspondencias “DNI-ubicación en el aula”.
Finalmente, podés preguntar:

- *¿Qué sucedería si el mensajero no conociera el mapa del aula?, ¿cómo se enviaría el mensaje?*
- *Si tuvieran que enviar nuevamente un mensaje al mismo destino, ¿pueden completar la tarjeta de mensaje sin volver a preguntar por el DNI?, ¿por qué?*

Es probable que la respuesta a la primera pregunta no resulte sencilla, por este motivo se puede reflexionar junto con los estudiantes sobre alternativas de envío de mensajes si el mensajero no conoce el mapa del aula. Por ejemplo podría enviar una copia del mismo mensaje a todos los estudiantes, asegurándose de esta manera que el destinatario lo reciba. Podés preguntar: *si un estudiante recibe un mensaje que no es para él, ¿qué hará con el mensaje?.* Se espera que respondan que lo descartará, solo retendrá los mensajes que son dirigidos a él.
Asimismo, se espera que los estudiantes adviertan que si un estudiante tiene que enviar un nuevo mensaje al mismo destino, conocido, podrá completar la tarjeta de mensajes sin necesidad de preguntar quién tiene el DNI, dado que esa información la tiene registrada en su tabla de correspondencia.

Cierre

A modo de cierre, podés explicar que la actividad desarrollada esboza los conceptos de:

- identificación física de un dispositivo en la red para poder entregar mensajes y,
- el funcionamiento del protocolo ARP para obtener la dirección física o MAC a partir de la dirección lógica o IP.

En este punto, explicitar la analogía entre el juego y el mundo de las redes: *los estudiantes que envían y reciben mensajes, son los extremos de la comunicación, las computadoras, y el estudiante que entrega los mensajes, el mensajero, simboliza al dispositivo que las conecta*.
A continuación podés construir la analogía, con las siguientes ideas:
Un estudiante se quiere comunicar con otro del que conoce, en un principio, su dirección lógica, representada por el DNI; en el mundo de las redes, el DNI se corresponde con la **dirección IP.**
Para que el mensaje llegue al destino final, además se debe conocer la dirección física, representada en esta actividad por la ubicación de fila-columna del estudiante, y conocida como **dirección MAC** en redes. Dicha dirección, se averigua a través del protocolo **ARP** y se guarda en la **tabla ARP** de cada computadora, la cual contiene las correspondencias IP-MAC. En esta actividad, se encuentra simulada por la tabla de correspondencia “DNI-ubicación en el aula”.
Asimismo, para interconectar las computadoras en una red se necesitan otros dispositivos que en esta actividad, están representados por el mensajero. Dependiendo de su comportamiento, éste puede simular un **hub** o un **switch**. Simula un hub si el mensajero no conoce el mapa del aula y, en este caso, cuando recibe un mensaje debe enviarlo a todos

los estudiantes para que el destinatario lo reciba. Mientras que en el caso que el mensajero conozca el mapa del aula, éste puede reenviarlo directamente al destinatario, simulando el funcionamiento de un switch.
Al llegar a este momento, podés elaborar junto con los estudiantes una tabla como la siguiente, que sistematiza las ideas surgidas.

	Juego	**REDES**
Extremos de la comunicación	Estudiantes	Computadoras
Dirección Lógica	DNI del estudiante	Dirección IP de la computadora
Dirección Física	Ubicación del estudiante en el aula, fila-columna	Dirección MAC de la computadora
Resolución de direcciones	Tabla DNI-ubicación en el aula	Protocolo ARP
Dispositivos para interconectar computadoras	Mensajero	Hub o Switch

Ficha N°5

Ficha para el estudiante

Enviando mensajes internos

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 5: Identificación de los recursos en Internet

Bajada

Comunicate con tus compañeros utilizando las tarjetas de mensajes y siguiendo las pautas dadas en clase.

Tabla de correspondencias DNI-Ubicación en el aula:

Correspondencias DNIs - Ubicación en el aula	
DNI del estudiante	**Ubicación del estudiante**
	F: **C:**
	F: **C:**

Tarjetas de mensajes (una por cada mensaje a enviar):

Mensaje	
Dirección del Emisor (origen)	**F:** **C:**
Dirección del Receptor (Destino)	**F:** **C:**
Mensaje	

Actividad

Titulo

Enviando mensajes intergrupo

Objetivos

Que los estudiantes puedan:

- Comprender la funcionalidad del servicio de DNS.
- Conocer el funcionamiento básico de un router.

Modalidad de trabajo

Actividad con toda la clase.

Materiales y recursos utilizados

Se trabajará con las tarjetas de mensajes, una por cada mensaje a enviar.

Tarjeta de mensajes:

Mensaje	
Dirección lógica del Emisor	**DNI del origen:**
Dirección lógica del Receptor	**DNI del Destino:**
Mensaje	

Desarrollo

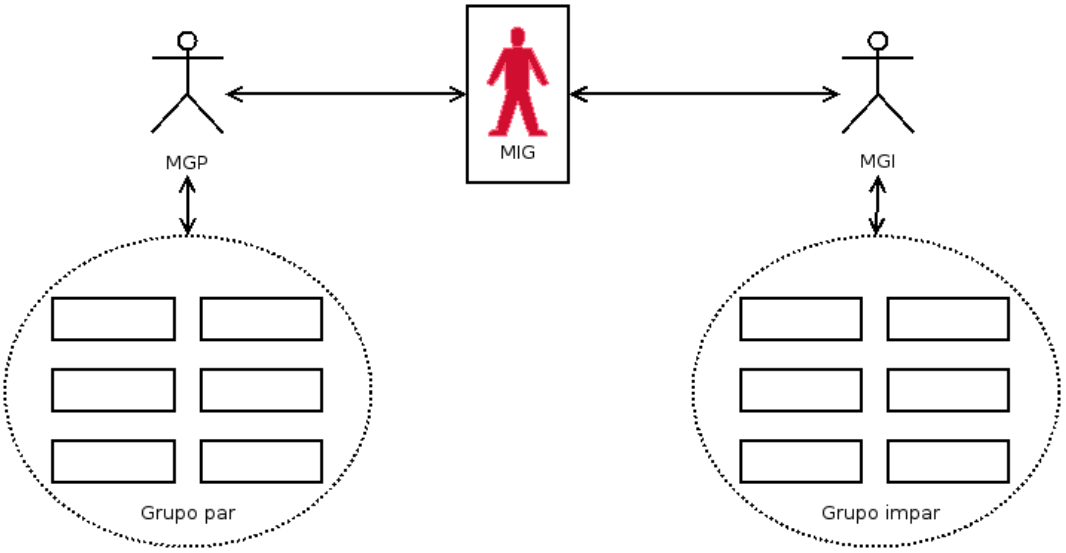
Esta actividad se organiza en 3 momentos: en el primer momento se arman dos grupos que intercambiarán mensajes, en el segundo se simula el envío del mensaje y los procesos involucrados en el mismo y en el tercero, se simula la entrega del mensaje al destino.
Para el desarrollo de esta actividad, deberás contar con una lista con los nombres, apellidos y DNIs de los estudiantes.

Momento 1: preparando el escenario

Separá a los estudiantes en dos grupos, de acuerdo a si su DNI es par o impar, y ubicá a cada grupo en un determinado sector de pupitres o bancos del aula.
Cada asiento se identificará con una numeración, idéntica a la usada en la actividad anterior.

Se deberán elegir también 3 mensajeros: 1 mensajero intergrupo 

, conocido solamente por los otros 2 mensajeros, y 2 mensajeros de grupo  , conocidos por los integrantes de su grupo y por el mensajero intergrupo.
El mensajero intergrupo, **MIG**, es el encargado de decidir a qué mensajero de grupo, **MG**, entregar el mensaje.
Los dos mensajeros de grupo, **MG**, son los encargados de entregar el mensaje al estudiante que corresponda de su grupo.
Para identificar a los mensajeros grupo, podés usar las siglas **MGP** para el mensajero del grupo par y **MGI** para el del grupo impar. En la siguiente figura se ilustra el esquema de comunicación entre los grupos a modo de ejemplo. Podés usar una figura similar al comenzar la actividad.



Momento 2: averiguando los DNIs de mis compañeros

Advertí a los estudiantes que en esta actividad deben enviar un mensaje a un compañero del otro grupo usando su nombre y apellido. Podés preguntar: *¿qué datos se necesitan para poder completar la tarjeta del mensaje a enviar?.* Se espera los estudiantes hayan observado que necesitan conocer el DNI destinatario del mensaje. En este momento, podés recordar que contás con un listado completo con los nombres y DNIs de los estudiantes. Por lo tanto, este inconveniente podría resolverse preguntándote en voz alta cuál es el DNI para un nombre dado. Espera unos minutos para que los estudiantes completen sus tarjetas de mensajes, y luego preguntales: *¿quién creen que debe continuar el proceso de envío?.*

Momento 3: transmitiendo los mensajes

En este momento se espera que los estudiantes deduzcan que la persona que puede continuar el proceso de envío debe ser el mensajero del grupo al que pertenece el emisor, MGP o MGI. Se puede reflexionar junto con los estudiantes que solo se pueden enviar

mensajes a otro grupo a través del MG del emisor, quien enviará el mensaje al MIG para que éste lo envíe al mensajero del otro grupo.
Preguntá a los estudiantes: *¿qué dato piensan utiliza el MIG para decidir a qué MG enviar el mensaje?*. Se espera que adviertan que, dado que los grupos están separados según su DNI, éste es el dato que debe evaluar el MIG para elegir el MG: DNI par o impar.
Finalmente para que el destino reciba su mensaje, el MG del grupo destino debe decidir a qué pupitre entregar el mensaje. Podés ayudar a los estudiantes a pensar sobre esta cuestión, preguntándoles: *¿con qué mecanismo el MG puede determinar el pupitre del destino?.* Hacé una puesta en común de las posibles soluciones, enfatizando en que la forma de averiguar la dirección física a partir de la dirección lógica es similar al funcionamiento del protocolo ARP, trabajado en la actividad previa.
Se les puede preguntar a los estudiantes: *¿es necesario que un mensaje se envíe a través del MIG si el origen y el destino se encuentran en el mismo grupo?*. Los estudiantes deberían inferir que en este caso el único mensajero necesario es el MG del grupo.
Indagá a los estudiantes sobre el rol que cumplen los MG y el MIG, teniendo en cuenta que los primeros conectan integrantes de un mismo grupo usando direcciones físicas (fila y columna) y el segundo a distintos grupos utilizando direcciones lógicas (DNI). En este punto, probablemente deberás guiar a los estudiantes para que relacionen los MG con los **switchs** o **hubs** presentados en la actividad anterior. Este tipo de dispositivos conectan redes locales reenviando datos en función de las direcciones físicas. Por otro lado, el MIG utiliza direcciones lógicas y conecta distintos grupos, de la misma forma que los **routers** conectan distintas redes.

Cierre

A modo de cierre, podés explicar que para las personas usualmente los nombres son más fáciles de recordar que los números y que en Internet también se puede identificar los recursos a través de su nombre. Cuando accedemos a una página de Internet lo hacemos por su nombre, no por su dirección en el servidor. Sin embargo, las computadoras y dispositivos de red solo reconocen direcciones IP, que son números, para identificarlos. Por ello, cuando se accede a algún recurso en Internet a través de su nombre, como por ejemplo a la página www.educ.ar, es necesario traducir este nombre a una dirección lógica (dirección IP). Para esto existe el **servicio de DNS**, que se encarga de traducir los nombres a direcciones lógicas. El rol que asumiste en la clase cuando te consultaron respecto a cuál era el DNI de un estudiante dado su nombre, es similar a la función del **DNS** en una red.
Además, cuando un mensaje se transmite de una red a otra, se necesita un dispositivo que interconecte redes y encamine los mensajes entre ellas. Este dispositivo se denomina **router**. En esta actividad, el mensajero MIG cumplé ese rol, decidiendo si el mensaje debe entregarlo al mensajero MGP o MGI, dependiendo de si el DNI del destino es par o impar.

Ficha N°3

Ficha para el estudiante

Enviando mensajes intergrupo

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 5: Identificación de los recursos en Internet

Bajada

Usá una tarjeta como la siguiente, para enviar un mensaje a tus compañeros de acuerdo a las pautas presentadas en la clase.

Mensaje	
Dirección lógica del Emisor	**DNI del origen:**
Dirección lógica del Receptor	**DNI del Destino:**
Mensaje	

Actividad

Titulo

Inspeccionando la identidad de los dispositivos

Objetivos

Que los estudiantes puedan:

- Averiguar las direcciones físicas y lógicas de las computadoras y/o celulares disponibles en la escuela.

Modalidad de trabajo

Actividad individual.

Materiales y recursos utilizados

Se utilizarán los teléfonos celulares de los estudiantes y/o las computadoras del aula.

Bajada para el aula

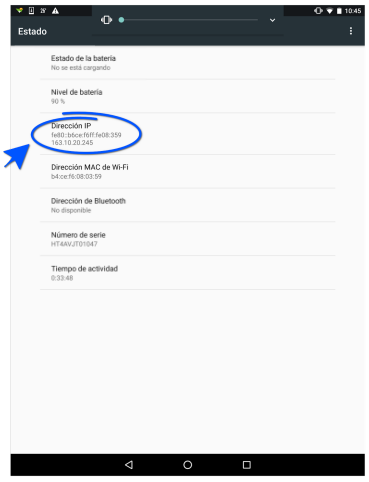
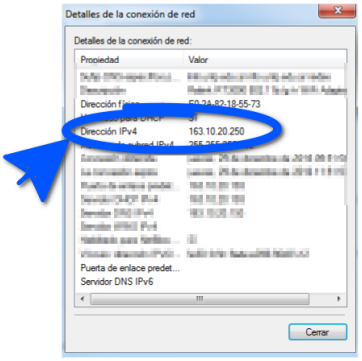
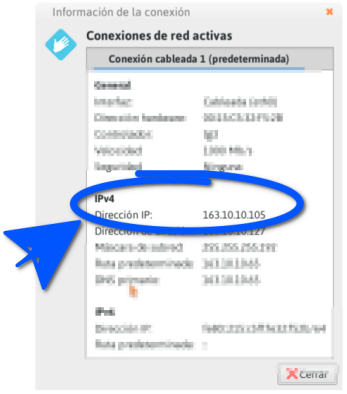
En esta actividad, se trabajará con las herramientas disponibles en cada dispositivo, computadora o netbooks y los teléfonos celulares de los estudiantes, para inspeccionar las direcciones IP y MAC de los mismos.

Para dar comienzo a la actividad, se pueden retomar algunos conceptos que comenzaron a trabajarse en la actividad anterior. Planteá los siguientes interrogantes en forma colectiva: *¿cómo se identifican las computadoras de la escuela en Internet?, ¿tienen una dirección física?, ¿qué ocurre con los celulares?.*

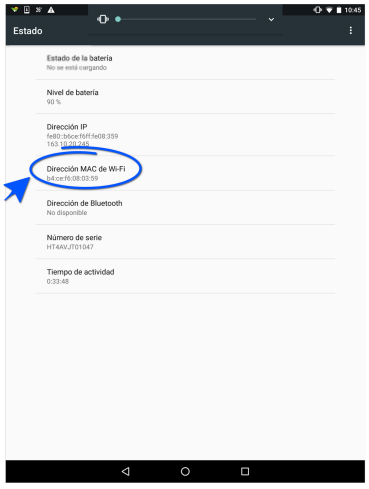
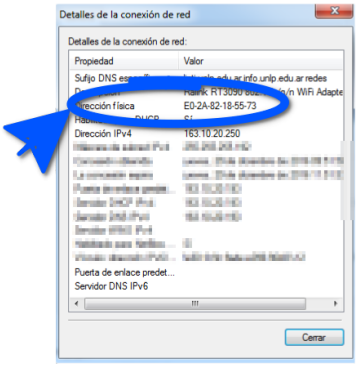
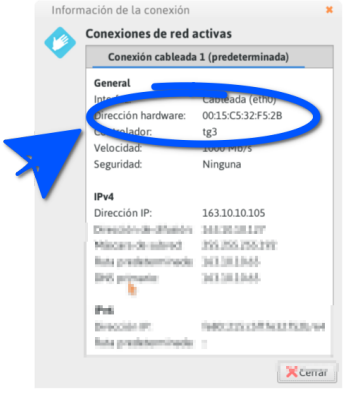
Probablemente estas preguntas no resulten sencillas de responder, algunos estudiantes podrán aproximarse a la idea que las computadoras y celulares tienen una dirección IP, recordando que esta información en varias ocasiones es requerida por algunos juegos. Sin embargo, la dirección MAC es raramente utilizada. En este punto podrás orientar a los estudiantes relacionando las ideas de dirección lógica, dirección IP, y física, dirección MAC, de las actividades previas.

Luego, podés preguntar: *¿hay alguna manera de averiguar la dirección IP de tu computadora y/o celular?, ¿cómo?*. Posiblemente algunos estudiantes lo hayan experimentado y sepan cómo hacerlo, en ese caso se les puede pedir que muestren a sus compañeros los pasos a seguir; o en caso contrario, indicales cómo hacerlo.

En las imágenes que se presentan a continuación, se muestran ejemplos de cómo encontrar esta información, según los distintos sistemas operativos.

Android	Windows 7	Linux
		

A continuación, preguntá a clase: *¿saben cómo averiguar la dirección MAC de sus computadoras y/o celulares?*. De la misma manera que en el caso anterior, si algún estudiante conoce cómo hacerlo, pedile que muestre a sus compañeros los pasos a seguir; o en caso contrario, indicale cómo hacerlo.
En las imágenes a continuación se muestran ejemplos de cómo encontrar esta información en distintos sistemas operativos. En estos casos, la dirección MAC se muestra en el mismo contexto que la dirección IP.

Android	Windows 7	Linux
		

En este momento, podes explicar que en Internet un dispositivo se identifica de forma unívoca a través de su dirección IP, la cual debe ser indicada en forma explícita por el emisor al enviar el mensaje. En el caso de las computadoras, netbooks y teléfonos celulares, la dirección IP es utilizada internamente por las aplicaciones de Internet, como las redes sociales, el navegador web o el correo electrónico. Por otro lado, están las direcciones MAC que son direcciones físicas grabadas en la placa de red de los

dispositivos. Dicha dirección, permite la entrega de mensajes en la red local, de manera tal que la computadora o teléfono celular procese el mensaje si va destinado a esa dirección física y lo descarte en caso contrario.

Cierre

A modo de cierre, enfatizá que tanto las direcciones lógicas como físicas de una computadora y/o teléfono celular se pueden consultar a través de alguna herramienta de software provista por los mismos. En algunas situaciones prácticas, esta información es requerida. Por ejemplo, el administrador de una red local puede restringir el acceso a la misma por la dirección MAC, ya sea por cuestiones de seguridad o para controlar la cantidad de conexiones. En este caso, cada vez que intentemos utilizar dicha red, debemos informar nuestra dirección MAC al administrador. Los juegos en red, habitualmente solicitan la dirección IP para unirse a un servidor.

La dirección IP es un código numérico que identifica a equipos o dispositivos de una red. Su nombre IP proviene de las siglas en inglés **Internet Protocol**. Según su versión pueden ser **IPv4** o **IPv6**.
Las direcciones IPv4 tienen el siguiente formato: 203.0.113.200. Suele escribirse de esta manera, en lugar de cómo un solo número grande, por una cuestión práctica y de facilidad de lectura. Estas direcciones se expresan como cuatro números decimales, que pueden variar entre 0 y 255, separados por punto. Cada número de la dirección IPv4 es la representación decimal de un número binario de 8 bits.
Las direcciones IPv6 surgieron porque la cantidad de direcciones en la versión IPv4 ya no resultan suficientes para la creciente cantidad de dispositivos con capacidad de conectarse a Internet: empiezan a acabarse las IP para identificar a los miles de millones de dispositivos de Internet. IPv6 asigna 128 bits a cada dirección IP en vez de sólo 32 como IPv4. Las direcciones IPv6 tienen el formato: 2001:db8::1:0:0:1, donde los números están en codificación hexadecimal; cada parte de la dirección separada por el símbolo ‘:’ contiene 16 bits y el símbolo ‘::’ representa una secuencia de ceros en la dirección. Este nuevo formato de direcciones aumenta en gran medida el número de direcciones IP disponibles: pasan de 2^{32} a 2^{128}.

Secuencia didáctica 6 : Ruteo y transporte de datos en Internet

Bajada

En las redes en general y en Internet en particular, para que la comunicación sea posible y un mensaje llegue desde un origen a un destino, debe existir al menos un camino entre los mismos. Dos computadoras pueden estar directamente conectadas, o pueden estarlo a través de diversos medios y dispositivos intermedios. Pueden existir rutas o caminos alternativos.
Los **routers** son dispositivos de red que conectan a una red con otra y cumplen la función de encaminar mensajes. Cuando un mensaje llega a un router el mismo debe evaluar cuál es el mejor camino para llegar al destino, en base a su información de **ruteo** (lista de posibles rutas) y reenviar el mensaje según esa decisión.
Para poder transportar datos a través de Internet utilizando el camino más eficiente, los routers utilizan “mapas de la red” que les permiten calcular el mejor camino en base a la información asociada a cada **ruta**. Esta información puede ser, por ejemplo, la velocidad del camino, la confiabilidad del mismo, entre otras características.
En la primera actividad de esta secuencia, se analizan las posibles rutas entre un origen y un destino, evaluando cómo elegir una de ellas y acercándose a la función básica de los routers.
En la segunda actividad, a través de un juego de roles, los estudiantes experimentarán cuál es la información que los routers necesitan para poder reenviar los mensajes a través de la red y algunas de las acciones que los mismos realizan para cumplir la función de ruteo.

Objetivos

Que los estudiantes logren:

- Comprender cómo es el transporte de datos en Internet.
- Conocer las funciones básicas de un router.

Actividad

Titulo

Viajes y tiempos

Objetivos

Que los estudiantes puedan:

- Comprender cómo se llevan a cabo las decisiones de ruteo.

Modalidad de trabajo

Actividad individual

Materiales y recursos utilizados

Papel y lápiz, y una hoja impresa con una tabla de rutas como la siguiente:

Punto 1	**Punto 2**	**Medio de transporte**	**Duración minutos/horas**
Casa	Terminal de Ómnibus	Colectivo (ruta A)	30 minutos
		Colectivo (ruta B)	50 minutos
Terminal de Ómnibus	Aeropuerto Ezeiza	Colectivo	40 minutos
Aeropuerto Ezeiza	**Aeropuerto México**	Avión	10 horas

Desarrollo

La actividad propone organizar un viaje utilizando diferentes medios de transporte, identificando detalladamente los pasos a seguir para llegar al destino deseado. El objetivo es que los estudiantes comprendan la forma en que se conectan dos puntos geográficamente distantes y cómo se puede actuar en caso que surja algún inconveniente de conexión.

Plantear a los estudiantes que deben organizar un viaje imaginario, partiendo desde su casa con destino a México, e indicarles que en la Ficha N° 7 disponen de una tabla de rutas posibles. En este momento, pedile a cada uno que elija y dibuje en una hoja las rutas posibles a seguir para llegar a México, indicando los medios de transporte que necesitarán utilizar para desplazarse entre los puntos intermedios del viaje.

Revisá junto con los estudiantes los distintos puntos intermedios del recorrido para que los tengan presentes al momento de elegir las rutas:

- Casa
- Terminal de Ómnibus
- Aeropuerto de Ezeiza
- Aeropuerto de Internacional de México

Podés dar algunas pautas para realizar el dibujo de las rutas: los puntos del recorrido se representan con círculos y los traslados o rutas a seguir, con flechas. Las flechas que unen círculos tienen como origen el punto de partida y como destino el punto de llegada y se etiquetan con el medio de transporte y el tiempo estimado de traslado de dicho transporte. A modo de ejemplo, para dibujar la ruta entre la casa y la terminal de ómnibus se espera que los estudiantes realicen un gráfico similar al siguiente:

Luego de dar un tiempo para que cada estudiante realice su plan de viaje o dibujo, planteá

los siguientes interrogantes:

- *¿Cuál es el tiempo de demora total del viaje utilizando la ruta A en el primer tramo?*
- *¿Qué sucedería si la ruta A no estuviera disponible?, ¿cómo lo solucionarían?, ¿cómo impactaría esto en la demora total del viaje?*
- *¿Qué pasaría si la ruta A no estuviera disponible y no hubiese habilitada una ruta alternativa ?, ¿perderíamos el viaje a México?*

Se espera que los estudiantes lleguen a la conclusión que si se utiliza la ruta A en el primer tramo, el tiempo total de demora del viaje es 11 horas y 10 minutos. También se espera que respondan que si la ruta A no estuviera disponible, podrían usar la ruta B, incrementando la duración total de viaje. En caso que la ruta B no estuviera habilitada, y la ruta A no estuviera disponible, no se podría realizar el primer tramo del viaje, desde la casa a la terminal, con lo cual no se llegaría a tomar el micro y se perdería el vuelo.

Otras preguntas que podés hacer para reflexionar sobre las elecciones de rutas podrían ser:

- *¿Qué criterios tendrían en cuenta para elegir entre la ruta A y la ruta B si ambas están disponibles?*
- *¿El camino más rápido o el más corto siempre es el mejor?, ¿qué pasa si es menos confiable, por ejemplo hay más congestión de tránsito y mayor probabilidad de demoras?*

A partir de las respuestas de los estudiantes, podrás advertir que hay dos conexiones posibles entre la casa y la terminal de ómnibus y, si bien en un principio se eligió la ruta A por considerarla más rápida, en la vida real la decisión no es tan simple, existen otros factores como demoras inesperadas. Por ejemplo en determinado momento del día suele haber mayor cantidad de tránsito en un camino que en otro, alterando el tiempo estimado para el viaje.

En este momento podés explicar que problemas similares a los planteados en esta actividad ocurren en las redes de datos cada vez que se intenta comunicar por Internet dos dispositivos, por ejemplo, cuando mandamos un correo electrónico, mensajes por Whatsup o Telegram, solicitamos una página web, etc. A menudo existen varios caminos posibles para comunicar dos dispositivos a través de Internet y es la tarea de dispositivos especializados llamados **routers**, decidir cuál puede ser el mejor camino en base a distintos criterios.

Cierre

A modo de cierre, podés reflexionar con los estudiantes que la elección de rutas que propone la actividad desarrollada, es análoga a la que hacen los routers de Internet cuando se conectan dos dispositivos. Es común tener varias rutas, al menos dos con distintas características para poder transmitir datos entre dos dispositivos. En caso de estar disponibles varias rutas a un mismo destino, se elige aquella que resulte la mejor de acuerdo a un determinado criterio, por ejemplo la más corta, es decir donde hay menos routers por los que pasar, o la más rápida dependiendo de la velocidad del medio de transmisión. También puede ocurrir que alguna ruta esté muy congestionada o incluso no disponible, con lo cual, aunque ésta sea la mejor ruta, el router deberá elegir otro camino para enviar el mensaje. En caso que se contara con una única ruta para conectar dos puntos y ésta no estuviera disponible, no sería posible transmitir datos entre esos dos puntos.

Ficha N°3
Ficha N°7

Ficha para el estudiante

Viajes y tiempos

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 6: Ruteo y transporte de datos en Internet

Bajada

Armá tu propio plan de viaje o mapa para trasladarte desde tu casa a México siguiendo las indicaciones dadas en clase y usando la siguiente tabla de rutas. Indicá los medios de transporte que vas a utilizar para trasladarte entre los puntos intermedios del viaje.
¿Empezamos el viaje?

Punto 1	**Punto 2**	**Medio de transporte**	**Duración minutos/horas**
Casa	Terminal de Ómnibus	Colectivo(ruta A)	30 minutos
		Colectivo(ruta B)	50 minutos
Terminal de Ómnibus	Aeropuerto Ezeiza	Colectivo	40 minutos
Aeropuerto Ezeiza	**Aeropuerto México**	Avión	10 horas

Actividad

Titulo

Paquetes y tabletas[1]

Objetivos

Que los estudiantes puedan:

- Comprender el funcionamiento de un router y las decisiones que se pueden tomar al momento de recibir y enviar un mensaje.

Modalidad de trabajo

Se trabajará con todo el curso.

Materiales y recursos utilizados

Se trabajará con las plantillas 1 y 2: la plantilla acciones, una por cada estudiante mensajero y, la de tabletas, una por cada estudiante que envía mensajes.
La cantidad de plantillas de tabletas a imprimir depende de cuánto tiempo se disponga para el desarrollo del juego.

Plantilla 1 - Acciones

Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	No entregar esta tableta
Entregar la tableta ahora	No entregar esta tableta

Plantilla 2 - Tabletas

Origen:	Origen:
Contenido:	Contenido:
Destino:	Destino:

[1] Esta actividad está inspirada en https://goo.gl/mlLcYJ

Origen: Contenid: Destino:	Origen: Contenido: Destino:
Origen: Contenido: Destino:	Origen: Contenido: Destino:

Desarrollo

Esta actividad propone un juego de roles cuyo objetivo es que los estudiantes se envíen mensajes. En esta oportunidad se utilizarán unas tarjetas llamadas “tabletas” que restringen la cantidad de caracteres y números que las mismas pueden contener, resultando en la necesidad de enviar varias tabletas para un mismo mensaje. Éstas simulan un comportamiento análogo a los paquetes de datos de Internet. Por otro lado, algunos estudiantes cumplirán el rol de mensajeros y realizarán algunas acciones que representan en forma simplificada el comportamiento de los routers en Internet.

Para dar comienzo a la actividad, elegí a tres estudiantes y asignales el rol especial de mensajeros, y al resto organizalos en parejas y asignales el rol de emisores y receptores de mensajes.

Para el desarrollo de la actividad es fundamental que los integrantes de cada pareja estén separadas, por ejemplo, en 2 aulas, o en lugares del aula distantes, según tus posibilidades, para que no puedan comunicarse.

Cada mensajero dispondrá de una copia de las “tarjetas de acción” de la Ficha N° 8. Las mismas son de tres tipos:

- Entregar la tableta ahora.
- Entregar esta tableta luego de la siguiente.
- No entregar esta tableta.

Explicar a los estudiantes cómo deben ser los mensajes que se pueden enviar entre las parejas: contienen más de 6 caracteres y no son frases en castellano. Por ejemplo, podría

elegirse mandar un número de teléfono o cadenas de texto sin sentido como “XXIHUAJKIL”. En este punto recuperar las ideas de las actividades previas y mostrar que los mensajes deben contener la dirección de origen y de destino. La restricción del idioma es para evitar que los mensajes se re-armen infiriendo su significado.

A continuación, indicá a los estudiantes que recorten las “tabletas” de la Ficha N° 8. Con el objetivo de comprobar si los estudiantes están entendiendo las reglas del juego, preguntales: *¿podés escribir un mensaje en una tableta?.* Se espera que los estudiantes hayan observado que en una tableta se puede escribir un mensaje de a lo sumo seis caracteres o números, entonces no se puede escribir el mensaje entero en una tableta. Luego, pedile a uno de los integrantes de cada pareja que piense un mensaje respetando la estructura de los mismos y que lo escriba en sus tabletas, teniendo en cuenta que un mensaje ocupará más de una tableta. Es importante destacar en este momento que todas las tabletas deben incluir el nombre del destinatario.

Solicitar a cada estudiante mensajero que mezcle las tarjetas de acción y, a los estudiantes que enviarán mensajes, que entreguen las tabletas que los contienen, a uno de los tres mensajeros. Por cada tableta recibida, el mensajero debe tomar una carta de acción y llevar a cabo dicha acción. Estos pasos deben repetirse para todas las tabletas de todos los estudiantes.

El inicio del juego probablemente sea un poco caótico y frustrante, y la entrega de mensajes posiblemente no se pueda concretar. Podés intervenir y conversar con los estudiantes sobre los problemas que están teniendo los mensajeros para entregar los mensajes y los destinatarios en la recepción de los mismos. Ayudá a los estudiantes a resolver las situaciones planteadas con preguntas como las siguientes:

- *¿Conocés el orden de las tabletas para rearmar el mensaje original?, ¿qué problema te genera?, ¿se te ocurre cómo solucionarlo?*. Se espera que los estudiantes concluyan que no conocen el orden de las tabletas y que esto provoca un problema al destinatario del mensaje porque no sabe cómo reconstruirlo. Es probable que las propuestas de solución no surjan rápidamente. Orientalos con algunas ideas como por ejemplo que resulta necesario numerar secuencialmente las tabletas y transmitir ese número, con lo cual debe formar parte de la información de la tableta, allanando el camino para que surja la propuesta de usar una de las 6 celdas de la tableta para anotar el número de tableta. Luego podés preguntar, si esto trae algún problema, *¿cuál?*. Conversá sobre el tamaño de los mensajes. Seguramente los estudiantes deduzcan que hay menos espacio para los datos reales del mensaje, un casillero se lo consume el número de tableta. Entonces, *¿usamos la misma cantidad de tabletas para nuestros mensajes?.* Se espera que los estudiantes observen que necesitarán más tabletas para mandar los mensajes dado que una de las celdas se utiliza para guardar información “de control”.
- *¿Estás seguro que las tabletas llegaron al destino?, ¿se podrían perder?. Si pensás que efectivamente alguna tableta se perdió ¿qué se puede hacer para recuperarla?.* Posiblemente algunos estudiantes piensen que no hay posibilidades de pérdida de tarjetas y otros hayan experimentado la pérdida de alguna. Podés ayudarlos a construir estas ideas por ejemplo, entregando más tabletas para que comprueben que efectivamente hay posibilidades de extravío: a un mensajero puede habérsele caído la tableta, o puede haber realizado la acción “No entregar esta tableta” que consiste en descartarla, etc. En este momento dialogá con los estudiantes sobre propuestas de solución para el caso de pérdida de tabletas y orientalos en la elaboración de las

mismas con algunas preguntas: *¿le podríamos pedir al destinatario que le avise al mensajero cuando recibe una tableta?, si no avisa nada, ¿podríamos pensar que la tableta no se entregó?.* Con las respuestas recogidas construí junto con los estudiantes una propuesta de solución similar a la siguiente: *el mensajero le pedirá al destinatario que le avise cuando recibe una tableta y no le enviará una nueva hasta no recibir confirmación de la enviada previamente. En caso de no recibir la confirmación en un tiempo preestablecido, el mensajero podría considerar que la tableta se perdió y volverá a enviarla, para ello el mensajero mantendrá una copia de la tableta hasta su confirmación*. Reflexioná junto con los estudiantes sobre *cómo* identificar estas respuestas de confirmación. Podría surgir la idea de contar con una tableta especial para representar la confirmación que enviará el destinatario al mensajero.

- *¿Le podemos entregar las tabletas que componen un mensaje a distintos mensajeros?.* Podés reflexionar junto con los estudiantes que al ser la función del mensajero la entrega de tabletas de acuerdo al destino de la misma, no se presenta ningún inconveniente si una tableta se la entregamos a un mensajero y otra, del mismo mensaje, a un mensajero diferente. Las tabletas que componen un mensaje pueden tomar distintos caminos e igualmente llegar al destino. Como la llegada de las tabletas puede ocurrir en distinto orden resulta fundamental haberlas numerado.

Cierre

A modo de cierre, podés explicar a la clase que el juego propuesto es una analogía de los mecanismos de transporte de datos que se usan en Internet: las tabletas representan los **paquetes** o **datagramas** en Internet y su contenido, partes o fragmentos de un mensaje; en las tabletas se guardan además de los datos, información de “control” como ser el número de orden y si se trata de una confirmación, por ejemplo. Análogamente, en Internet los paquetes contienen **datos** y un **encabezado** con información de control. Dado que los paquetes tienen un tamaño preestablecido, el tamaño del encabezado reduce la cantidad de datos que se pueden transferir, pero agrega funcionalidades útiles.

Podés comentar que en Internet los mensajes se dividen en paquetes o datagramas para su transporte y que además, los canales por los que viajan estos paquetes no siempre son confiables: los paquetes a veces se dañan o se pierden. En este momento podes reflexionar con los estudiantes sobre *qué* impacto tiene perder paquetes de datos, pudiendo surgir ideas como: *dependiendo de la aplicación que estemos usando en Internet o lo que estemos haciendo, la pérdida de un paquete podría no impactarnos, por ejemplo si se transmite un video puede que no se perciba la pérdida de un paquete (un cuadro del video), mientras que si estamos transfiriendo un archivo, perder un paquete implica la alteración del contenido del archivo.*

En esta actividad el mensajero además de entregar tabletas, realiza una serie de acciones análogas a las de los routers de Internet:

- “Entregar la tableta ahora” representa el reenvío de un paquete recibido para que siga su camino.
- “Entregar esta tableta luego de la siguiente” significa que el paquete recibido sufrirá una demora antes de ser enviado por el router, debido a que tiene menos prioridad que otro paquete encolado o el router está procesando otro paquete.
- “No entregar esta tableta” representa la acción de descartar un paquete, lo cual puede ocurrir por ejemplo porque su tiempo de vida se agotó.

Los protocolos de Internet que se encargan del transporte y ruteo de datos son: **IP**, **TCP** y **UDP** (de sus siglas en inglés de *User Datagram Protocol*).

Ficha N°8

Ficha para el estudiante

Paquetes y tabletas

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 6: Ruteo y transporte de datos en Internet

Bajada

A) Si sos un mensajero, recortá las siguientes tarjetas de acción y seguí las indicaciones dadas en clase para saber cómo usarlas.

Plantilla 1: Acciones de los mensajeros

Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	Entregar esta tableta luego de la siguiente
Entregar la tableta ahora	No entregar esta tableta
Entregar la tableta ahora	No entregar esta tableta

B) Si tu rol en el juego es el de enviar y recibir mensajes, utilizá las tabletas de la plantilla siguiente para enviar un mensaje a tu compañero según las indicaciones dadas en clase.

Plantilla 2: Tabletas

Origen: Contenido: Destino:	Origen: Contenido: Destino:
Origen: Contenid:	Origen: Contenido:

Destino:	Destino:
Origen: Contenido: Destino:	Origen: Contenido: Destino:

Secuencia didáctica 7: Detección de errores

Bajada

Al inicio de este anexo se mencionó que un sistema de comunicaciones está integrado por emisores, receptores y medios de transmisión. En este contexto es crucial establecer mecanismos para verificar la integridad de un mensaje transmitido, es decir, si fue modificado por terceros o por ruido en el medio de transmisión.
La **confiabilidad** o **fiabilidad** en un sistema de comunicaciones se refiere a la probabilidad de un buen funcionamiento del mismo y depende de numerosos factores. Uno de ellos, el que se trabajará en esta secuencia didáctica, es la capacidad de poder verificar la integridad de los mensajes. En el mundo de las redes de computadoras, esta verificación se conoce como **detección de errores**.
Esta secuencia didáctica propone una única actividad en la cual se trabaja con un sistema que realiza dicha verificación.

Objetivos

Que los estudiantes logren:

- Comprender el concepto de detección de errores en la transmisión de mensajes en la red, a fin de garantizar su integridad en el envío.

Actividad

Titulo

Verificando la integridad del mensaje

Objetivos

Que los estudiantes puedan:

- Experimentar con un mecanismo que permite controlar la integridad de los mensajes transmitidos en un sistema de comunicaciones.

Modalidad de trabajo

Se trabajará en parejas.

Materiales y recursos utilizados

Se trabajará con la siguiente tarjeta de mensajes.

Mensaje	
DNI Remitente	

DNI Destino	
Mensaje	
Chequeo de Integridad	

Bajada para el aula

Esta actividad guarda cierta similitud con las anteriores: es un juego en el que los estudiantes, organizados en parejas, envían y reciben mensajes. El docente será el mensajero y se usarán los números de DNI de los estudiantes, direcciones lógicas, para identificar a los destinatarios.

Para comenzar la actividad, solicitá a los estudiantes que escriban un mensaje a algún compañero del que conocen su DNI, utilizando las tarjetas de la Ficha N° 9. Los mensajes a enviar tendrán un tamaño máximo de 30 caracteres.

Pedir a los estudiantes que observen las tarjetas y verifiquen si tienen todos los datos necesarios para enviar el mensaje. En este momento, cada estudiante podrá completar su tarjeta de mensajes con el DNI del origen, el DNI del destino, el mensaje que se quiere enviar, pero seguramente advertirá que se requiere el dato “chequeo de integridad” que desconoce.

En este punto podés proponer a los estudiantes que el valor de “chequeo de integridad” lo completen de acuerdo a la cantidad de vocales que contiene el mensaje: si el número de vocales es par, se completará con una letra “P” y si es impar con una “I”.

Un ejemplo de mensaje a transmitir puede ser el siguiente:

Mensaje	
DNI Remitente	**11.111.111**
DNI Destino	**22.222.222**
Mensaje	**Aquí me pongo a cantar Al compás de la vigüela, Que el hombre que lo desvela Una pena estrordinaria, Como la ave solitaria Con el cantar se consuela.**
Chequeo de Integridad	**P**

En este juego actuarás como mensajero y serás el encargado de distribuir los mensajes de acuerdo a los DNIs indicados en el campo “DNI Destino”. Para realizar el envío del mensaje, recoge todas las tarjetas y altera el mensaje, por ejemplo borrá o tachá algún carácter en algunos de los mensajes. Luego, entrega todos los mensajes a sus destinatarios y preguntales a los estudiantes: *¿pueden darse cuenta si el mensaje recibido fue alterado?.* Se espera que los estudiantes observen que en la tarjeta se incluye el indicador, “P” o “I” indicando si el mensaje contiene una cantidad par o impar de vocales y, propongan contar las vocales del mensaje recibido analizando si se trata de un número par o impar. De esta manera pueden comprobar si la celda de integridad indica lo mismo.

Para finalizar la actividad, proponer a los estudiantes que analicen las siguientes situaciones:

- *¿Qué dificultades presenta este sistema de detección de errores?*
- *¿Es posible saber en qué palabra se produjo la alteración?*
- *¿Se puede verificar correctamente la integridad si se borran dos vocales?*
- *¿El destinatario puede verificar si se eliminó una consonante?*

Se espera que los estudiantes adviertan que esta propuesta sólo detecta errores si se altera una cantidad impar de vocales, dado que si se altera una cantidad par, se mantendría el indicador “P” y sería imposible determinar que hubo una modificación. Con este método no es posible detectar alteraciones en las consonantes, ni verificar en qué carácter fue el error, por lo tanto sólo nos permite detectar algunos errores y no posibilita corregirlos.

Cierre

Para concluir la actividad, podés explicar que al enviar información a través de una red o al almacenarla en un medio digital, es posible que las interferencias o el ruido puedan alterar el contenido de la misma. En la actividad desarrollada se introdujeron alteraciones en las tarjetas de mensajes y se propuso un método simple para detectar, en algunos casos, estos errores. En el transporte de datos en redes y en Internet se usan distintas técnicas para detectar errores y algunas incluso permiten corregirlos. Esta actividad aplica el concepto de **bit de paridad**, que establece agregar un 0 si la cantidad de 1s del mensaje es par, en caso contrario agregar un 1.

Ficha N°9

Ficha para el estudiante

Verificando la integridad del mensaje

Anexo N° 3: Las redes de datos e Internet
Secuencia didáctica 7: Detección de errores

Bajada

Utilizando la siguiente tarjeta, enviá tu propio mensaje según las pautas dadas por tu docente.

Mensaje	
DNI Remitente	
DNI Destino	
Mensaje	
Chequeo de Integridad	

Secuencia didáctica 8: Búsqueda de información en la Web

Bajada

Internet, como la gran red de redes, alberga millones de recursos accesibles a través de múltiples servicios, siendo la Web uno de ellos. Es innegable la utilidad de la información publicada en los distintos sitios web, pero en la actualidad acceder a información relevante implica conocer el funcionamiento de aquellas herramientas que nos dan acceso a la misma, es decir los **buscadores web**.
Los algoritmos que implementan los buscadores web incluyen, entre otras funciones, distintas formas de mostrar a los usuarios los resultados de las búsquedas. Esto impacta directamente en lo que el usuario puede considerar relevante o no, pensando que aquellas referencias que se encuentran primero son las más importantes.
En esta secuencia didáctica se introducirá el concepto de búsqueda en la web y una primera aproximación a cómo funcionan los buscadores. Para ello, se proponen dos actividades. La primera es una actividad sencilla que permite contextualizar el problema: existe mucha información en la web diseminada en miles de millones de páginas y encontrarla no es una tarea que pueda realizarse manualmente. La segunda actividad presenta un mecanismo de ordenación por relevancia para visualizar los resultados de una búsqueda.

Objetivos

Que los estudiantes logren:

- Comprender que los buscadores web son herramientas necesarias para poder obtener información relevante en la Web.
- Comprender que los buscadores aplican distintos mecanismos para ordenar los resultados de las búsquedas.

Actividad

Titulo

¿Por qué necesitamos un buscador?

Objetivo

Que los estudiantes puedan:

- Reconocer la necesidad de contar con herramientas que asistan a los usuarios a la hora de encontrar información en la Web.

Modalidad de trabajo

Grupos reducidos.

Materiales y recursos utilizados

Computadoras o teléfonos celulares con acceso a Internet.

Bajada para el aula

Retomar el tema trabajado en la secuencia didáctica 3 sobre servidores y clientes web, en esta oportunidad explicando que la **Web** es un gran repositorio de información formado por páginas web y montado sobre Internet. La información en la Web se accede a través de los navegadores web o clientes web y la misma está almacenada en servidores web.
Para dar comienzo a la actividad organizá a los estudiantes en grupos reducidos y luego podés preguntarles: *¿cómo acceden a una página web de interés?*. Se espera que algunos estudiantes respondan que necesitan saber la "dirección" del sitio web o de la página aunque probablemente varios respondan que usan "Google".
Luego, si disponés en la escuela de computadoras o teléfonos celulares con conexión a Internet, podés pedirles a los diferentes grupos que accedan a algunas páginas, podría ser la de la escuela, la de Wikipedia, a alguna página de una institución muy conocida por la comunidad de la escuela, por ejemplo la de un club o un centro cultural y, observá junto con ellos, cómo acceden a los sitios. Seguidamente, preguntales:

- *¿Saben cuántos sitios web hay en Internet?*
- *¿Pueden buscar información en Internet sin usar un buscador, por ejemplo Google?*

Se espera que los estudiantes elaboren la idea que actualmente en Internet hay demasiados sitios web como para poder accederlos manualmente, sin contar con una aplicación que nos ayude a buscar información en ellos. Además, que seguramente no conozcan todas las direcciones de los sitios web solicitados.

Cierre

A modo de conclusión, podés ayudar a tus estudiantes a elaborar la idea que los buscadores en Internet permiten encontrar información en la Web. Esto se hace simplemente escribiendo una frase o palabra relacionada con el contenido que te interesa buscar, no siendo necesario conocer los nombres de todos los sitios web en los que podés encontrar dicho contenido. Además es imposible saber de antemano cuáles son todos los sitios web que contendrán información de interés y que con solo conocer la dirección de unos pocos buenos buscadores en la web se puede encontrar información relevante en millones de sitios web.

En 2005 había más de 11.500 millones de páginas web públicas accesibles a través de buscadores:
https://es.wikipedia.org/wiki/World_Wide_Web#Estad.C3.ADsticas.
Existen sitios donde es posible obtener varias de estas estadísticas. Por ejemplo según http://www.internetlivestats.com/total-number-of-websites/ hay más de 1000 millones de sitios web al momento de escribir este manual .

Actividad

Titulo

El orden de los resultados, ¿altera el producto?

Objetivos

Que los estudiantes puedan:

- Comprender cómo los buscadores en la Web ordenan los resultados mostrados según su relevancia.

Modalidad de trabajo

Grupos reducidos.

Materiales y recursos utilizados

Computadoras o teléfonos celulares con conexión a Internet.

Bajada para el aula

Esta actividad se planea en 3 momentos.

Momento 1: buscando información en la Web

Comenzar la actividad proponiendo a los estudiantes que busquen las frase "*cuidado del medio ambiente*" en distintos buscadores web. Se sugiere utilizar los siguientes buscadores, aunque se pueden proponer otros:

- Google: www.google.com.ar/
- Yahoo: search.yahoo.com/
- Bing: www.bing.com/
- DuckDuckGo: duckduckgo.com/

Una vez realizada esta búsqueda, podés preguntar: *¿cuántos resultados obtuvieron?, ¿fueron los mismos resultados los obtenidos con los diferentes buscadores?*. Dada la gran cantidad y variedad de resultados que seguramente se obtendrán y con la idea de saber qué criterio usan para seleccionar la información que les interesa, les podés preguntar: *¿cómo hacen para procesar o elegir el que les resulta más útil o interesante?.* Posiblemente surjan muchas propuestas, algunas podrán estar vinculadas a la preferencia por algún buscador en particular, otras resultarán inesperadas, sin embargo es una buena oportunidad para advertir que no solo es importante que el buscador encuentre sitios que contengan los términos buscados, sino que también necesitamos que nos ayude a ordenarlos por **relevancia**. Ahora podes preguntar: *¿qué es la relevancia en una búsqueda?, ¿la relevancia se mantiene entre los distintos buscadores?, ¿para todos Uds. son igualmente relevantes los resultados obtenidos?.* Un supuesto ampliamente aceptado postula que los primeros resultados que se muestran en la búsqueda son los más destacados o significativos para la búsqueda.

Momento 2: calculando la relevancia

Con la intención de indagar sobre cómo los buscadores determinan la relevancia del resultado de una búsqueda, podés preguntar a los estudiantes: *¿saben cómo los buscadores ordenan los resultados de las búsquedas?, ¿cómo piensan que lo harán?.*

Frente a estas preguntas, es probable que los estudiantes no encuentren fácilmente una respuesta, aunque posiblemente se aproximen a la idea que algunos buscadores para ordenar por relevancia cuentan la cantidad de ocurrencias de una o varias palabras en las páginas: mientras más veces aparezca, más relevante se considera esa página y aparecerá más arriba en la lista de resultados.

Luego podes explicar que existen otras formas más sofisticadas de determinar la relevancia de un sitio en una búsqueda y que es la que se trabajará en esta actividad y que está vinculada con la "popularidad" de las páginas. Esta técnica consiste en determinar la relevancia de la información de acuerdo a *cuántas referencias externas existen a esa página*. Los buscadores utilizan para esto los hipervínculos o enlaces que relacionan una página con otra.

En este momento podés proponer a la clase simular el funcionamiento de un algoritmo de ordenación de resultados de búsqueda según la relevancia en el sentido explicado. Para ello, vas a enunciar una serie de palabras y les vas a pedir a los estudiantes que evoquen qué película les trae primero a la mente cada palabra, solo vas a aceptar una respuesta por estudiante. Por ejemplo, podrían ser las siguientes palabras: acción, romance, comedia, drama, argentina, china, animación, nombres de actores, actrices o directores, entre otras. Esta lista se puede adaptar a la cantidad de tiempo disponible para la actividad, inclusive estas palabras pueden surgir del mismo grupo de estudiantes.

Luego, anotá en el pizarrón las respuestas de los estudiantes y, a partir de ellas, elaborá una tabla por palabra a buscar, que sistematice las respuestas. Por ejemplo, si la lista de palabras que enunciaste es: *Histórica*, *Ciencia Ficción* y *Steven Spielberg*, podrían surgir los siguientes nombres de películas:

Histórica	
Película	**Estudiantes que la mencionan**
300	3
Troya	1
La lista de Schindler	1
Rescatando al soldado Ryan	1

Steven Spielberg	
Película	**Estudiantes que la mencionan**
La guerra de los mundos	2
Rescatando al soldado Ryan	1
Tiburón	1
Indiana Jones	2
ET	2
Inteligencia Artificial	1

Ciencia ficción	
Película	**Estudiantes que la mencionan**
ET	5
Inteligencia Artificial	1
La guerra de los mundos	2
Alien	4

Luego de haber armado las tablas que asocian películas con palabras y evocaciones de estudiantes, proponé a la clase que busquen en las tablas una frase que contenga las palabras enunciadas por vos. Solicitá a un estudiante que proponga una búsqueda usando 2 de esas palabras, podría ser “Ciencia ficción y Steven Spielberg”. En un primer momento, podés pedir a los estudiantes que realicen la búsqueda eligiendo las películas que contengan ambos palabras. Se espera que los estudiantes lleguen al siguiente resultado:

- ET
- Inteligencia Artificial
- La guerra de los mundos

Anotá en el pizarrón el primer resultado obtenido y luego armá junto con los estudiantes una tabla de relevancia del resultado de la búsqueda según la “popularidad” de las películas, en este caso serían las que fueron evocadas más veces por los estudiantes:

Película	Cantidad total de menciones	Cantidad de evocaciones de “Ciencia Ficción”	Cantidad de evocaciones de “Steven Spielberg”
ET	7	5	2

Inteligencia Artificial	2	1	1
La guerra de los mundos	4	2	2

Luego preguntá: *¿cómo se ordenará el resultado de la búsqueda?.* Se espera que los estudiantes respondan que el orden del resultado es el siguiente:

1. ET
2. La guerra de los mundos
3. Inteligencia Artificial

Luego, organizá a los estudiantes en grupos reducidos y pedile a cada grupo que haga más búsquedas, usando un solo término y dos términos.

Una vez finalizadas estas actividades, planteá a modo de reflexión, los siguientes interrogantes:

- *¿Podríamos haber elegido de forma diferente las películas, para incluir más?, ¿cómo?, ¿cuál sería el resultado?.* Una posibilidad es elegir también a las películas que estén asociadas con una sola de las palabras, de esta manera incluiríamos a las películas que están solamente en la tabla “Ciencia Ficción” o “Steven Spielberg”. El nuevo resultado incluiría 7 peliculas: 3 que contienen ambas palabras y 4 que contienen solo una de las palabras (Rescatando al soldado Ryan, Tiburón, Indiana Jones y Alien). Sin embargo, aunque pareciera en un principio que este resultado es más amplio, podría incluir películas que no son de interés para el usuario.
- *¿Se podría haber contabilizado de forma diferente las evocaciones de las películas?, ¿de qué forma se les ocurre?.* Una posibilidad podría ser calcular el máximo valor entre las menciones de ambas palabras, por ejemplo: si buscamos “Ciencia ficción y Steven Spielberg”, con este nuevo esquema de puntajes ET tendrá 5 puntos en lugar de los 7 asignados con el esquema propuesto anteriormente. En este ejemplo, el resultado final, es decir el orden de relevancia del resultado de la búsqueda, se mantiene. Sin embargo, en otros casos el sistema que se utilice para determinar la relevancia podría llevar conducirnos a diferentes órdenes de relevancia.

Momento 3: los buscadores y el cálculo de la relevancia

En este momento podés hacer una analogía entre la actividad realizada en el momento anterior y el funcionamiento de los buscadores de Internet: la cantidad de menciones de cada película en un cierto tema podría equipararse a la cantidad referencias a una página web. Así la **relevancia** estaría dada por la cantidad de veces que una página fue referenciada para un cierto tema desde otra.

A modo de discusión final, comentar que existen muchas otras formas de indicar la relevancia de una página respecto de un tema dado. Las reglas para determinar ésto las definen los buscadores y no son fijas dado que las empresas que mantienen los buscadores siempre están incorporando cambios a los algoritmos de búsqueda. En este momento, podés pedir a los estudiantes que vuelvan a realizar la búsqueda inicial del término “*cuidado del medio ambiente*” en los distintos buscadores propuestos y comprueben que cada buscador muestra los resultados en distinto orden, dado que cada uno calcula la relevancia de forma distinta.

Cierre

A modo de cierre podés reflexionar junto con tus estudiantes que el orden en que los buscadores presentan los resultados de sus búsquedas, influye notoriamente en el acceso a los sitios referenciados. Si siempre aparecen primeros los mismos sitios web en los resultados, de alguna manera se induce a que accedas a ellos y eso redunda en que esos sitios son los más referenciados en determinados temas. Los sitios que se muestran entre los primeros enlaces son los que usualmente los usuarios consultan, rara vez un usuario consulta una página que se encuentra en la segunda o tercera página de resultados de la búsqueda.
En este momento también podés comentar que la relevancia en las búsquedas puede estar influenciada debido a que algunas empresas e instituciones pagan a las empresas que mantienen los buscadores para que sus sitios aparezcan entre los primeros resultados, sin embargo usualmente éstos aparecen con la indicación que se trata de anuncios o resultados patrocinados y deben ser interpretados como publicidades.

Modelo de evaluación

En este anexo se proponen dos actividades integradoras que articulan los conceptos abordados. Podrán desarrollarse tanto en grupos reducidos o en forma individual de acuerdo a tu consideración. Luego de realizar las mismas, se podrá evaluar el desempeño individual interrogando a los integrantes de cada grupo sobre los conceptos trabajados.
En la primera actividad integradora se busca poner en acción los conceptos aprendidos sobre redes experimentando con la red de la escuela y la hogareña (si se dispone). En la segunda actividad busca consolidar y reflexionar sobre protocolos.

Actividad Integradora 1

Titulo

Las redes hogareñas y su acceso a Internet

Objetivo

Que los estudiantes puedan:

- Relacionar los conceptos de redes aprendidos con los dispositivos de red disponibles de la escuela.

Modalidad de trabajo

Actividad individual o en grupos reducidos.

Materiales y recursos utilizados

Las computadoras de la escuela; los celulares de los estudiantes, y un editor de imágenes, como MiniPaint, disponible en https://lihuen.github.io/miniPaint

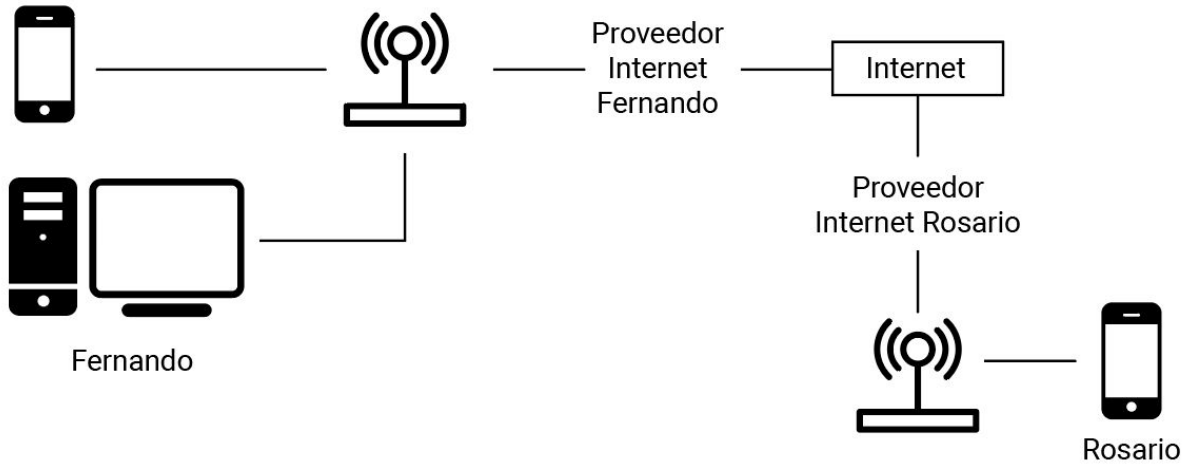
Bajada para el aula

Para dar comienzo a la actividad, planteá a los estudiantes los siguientes interrogantes: *¿cómo accedés a Internet desde la red de la escuela?, ¿podés hacerlo desde tu teléfono celular?, ¿qué necesitás para poder conectarte a Internet?, ¿con qué fichas del 'dominó de redes' de la primera actividad de la secuencia didáctica 4 está relacionado?.*

Se espera que los estudiantes respondan que la escuela accede a Internet a través de un proveedor de servicios o ISP y que para conectarse a Internet desde su teléfono celular la escuela debería contar con un router WiFi y tener permiso de acceso. Luego, solicitar a los estudiantes que verifiquen si tienen conexión a Internet; y, a partir de ahí, analizar de qué forma funciona.

Luego podés plantear una analogía entre la disponibilidad de Internet en la escuela y en sus hogares (en caso que dispongan). Para ello, proponé que armen un gráfico, una especie de maqueta simple en la que se representen los distintos dispositivos y otros elementos que participan en el camino que recorre un mensaje enviado entre dos compañeros. Para tener presente los elementos involucrados pueden utilizar, a modo de ayuda, las fichas del juego de dominó de la primera actividad de la secuencia didáctica 4.

Para armar el gráfico, se podrá utilizar el editor de imágenes MiniPaint o cualquier otro con el que se encuentren familiarizados. Una posible maqueta que representa los dispositivos involucrados y el camino transitado por un mensaje entre los estudiantes Fernando y Rosario, podría ser:



Esta maqueta ejemplo representa a los dispositivos y elementos que participan en la comunicación entre Fernando y Rosario, cuando se están mensajeando: Fernando podría estar en su casa usando una aplicación de mensajería desde su computadora y Rosario en una plaza comunicándose con la misma aplicación desde su celular.

Se espera que la maqueta de los estudiantes muestre que cada teléfono celular o computadora dispone de una placa de red que le permite conectarse en forma inalámbrica o cableada (Ethernet). Y que para poder conectarse a Internet, en el hogar deben contar con una conexión con un proveedor de Internet, ISP, a través de un router, lo mismo ocurriría si se están comunicándose desde la escuela.

Preguntar: ¿de qué manera se identifican en la red la computadora de Fernando y el celular de Rosario?
Se espera que los estudiantes respondan que la computadora y el celular pueden identificarse por su dirección IP y por su dirección MAC. Y que al momento de enviar el mensaje, para indicar la dirección del destino debe especificarse la dirección IP del destino.
Para finalizar podés plantear las siguientes preguntas:

- *¿Qué ocurre si alguno de los tramos de la ruta no está disponible, por ejemplo si a Rosario no le funciona el enlace con su proveedor?*
- *¿Piensan que ocurre lo mismo en la ruta que conecta a los proveedores de Internet?, ¿por qué?.* Aqui podés recordar a la clase las ideas de la primera actividad de la secuencia didáctica 6, “Viajes y tiempos”.

Se espera que los estudiantes infieran que si no está disponible alguno de los tramos de la ruta, en este caso el enlace con el proveedor, el mensaje solo podrá llegar a destino en caso de que haya un camino alternativo, mientras que si es el único camino posible el mensaje no podría hacerlo.
Análogamente si la ruta que conecta a los ISP no está disponible, no podrían comunicarse, sin embargo allí es más factible que existan varios caminos alternativos para garantizar la disponibilidad de las rutas en Internet.

Actividad Integradora 2

Titulo

Programando protocolos

Objetivos

Que los estudiantes puedan:

- Desarrollar un protocolo simple de una situación conocida como es el cruce de calles con un semáforo.
- Implementar una aplicación en Alice que simule el protocolo definido.

Modalidad de trabajo

Grupos reducidos.

Materiales y recursos utilizados

Computadoras con Alice 2.4.

Bajada para el aula

Analizar la siguiente situación: *“dos autos se encuentran en un cruce de calles donde hay un semáforo para controlar el tránsito”.*
Discutir con la clase las reglas que permiten resolver el paso de los autos en esta situación. En grupos reducidos, describir el escenario en términos de objetos intervinientes y escribir las reglas que aplicarán para el cruce de calles.
Luego, utilizando la herramienta Alice, desarrollar una simulación simple que permita mostrar cómo funciona el protocolo establecido.

Esta actividad articula los conceptos de programación trabajados en anexos anteriores con el concepto de protocolos trabajado en este anexo.

Glosario

ARP: Protocolo estándar de resolución de direcciones el cual, dada una dirección lógica, dirección IP, encuentra la dirección física asociada, dirección MAC.

Bit de paridad: Es un dígito binario que indica si el número de bits con un valor de *1* en un conjunto de bits es par o impar. Los bits de paridad conforman el método de detección de errores más simple.

Cliente: Es una aplicación instalada en un dispositivo digital que accede a un servicio remoto en otro dispositivo denominado servidor.

Dirección IP: Es un número que identifica de manera lógica a una placa de red de un dispositivo digital.

Dirección MAC: Es un número que identifica de manera física a una placa de red de un dispositivo digital. Utiliza 48 bits que corresponde de forma única a la placa.

DNS: Es un servicio de nomenclatura para identificar dispositivos conectados a redes como Internet o una red privada. Su función principal es traducir nombres de recursos con la dirección IP en la red.

Familia de protocolos TCP/IP: Es el conjunto de protocolos de red estándares, en los que se sustenta Internet y que implementan la transmisión de datos entre computadoras.

Hiperenlace o hipervínculo: Es un elemento constitutivo de una página web que referencia a otro recurso, por ejemplo a otra página y que permite leer en Internet de forma “hipertextual” en contraposición a la lectura secuencial.

Hipertexto: Es el texto mostrado en la pantalla de una computadora o de otros dispositivos electrónicos como teléfonos celulares o tablets, que contiene referencias a otros textos a través de hiperenlaces o hipervínculos, permitiendo a los usuarios acceder inmediatamente a ellos. Las páginas de Internet adoptaron está manera de mostrar su contenido .

HTTP: Es un protocolo de comunicación que permite las transferencias de información en la web. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes y servidores) para comunicarse.

Hub: Dispositivo usado para transmitir datos en una red cableada. Su principal característica es que retransmite por todos los puertos la señal recibida por uno de sus puertos, a excepción del puerto por el que la recibió.

Modelo cliente-servidor: Es un modelo de desarrollo de aplicaciones distribuidas en el que las funciones que cumplen cada componente del modelo, cliente y servidor, están claramente diferenciadas. Las componentes que prestan servicios, se denominan servidores y las que acceden a ellos se denominan clientes.

Placa de red: Es un componente de hardware, también conocido como tarjeta de red, incorporado en la computadora o en otros dispositivos electrónicos como teléfonos celulares o tablets, que permite conectar a los dispositivos con una red informática para compartir datos y recursos y, acceder a servicios.

Protocolo: En informática y telecomunicaciones, un protocolo de comunicaciones es un sistema de reglas que permite que dos o más entidades de un sistema de comunicación interactúen para transmitir información.

Router: Dispositivo que permite conectar redes. Su principal función es enviar datos a una dirección IP destino.

Servicio web: Es un aplicación de software que brinda acceso a los recursos de una página web de un sitio determinado, a los clientes web y les envía los recursos solicitados.

Servidor: Es un programa que permite responder solicitudes de clientes. Los servidores se pueden utilizar para diferentes tipos de servicios, el más conocido en Internet es el servicio web.

Switch: Dispositivo que permite transmitir datos en una red cableada, interconectando otros dispositivos. A diferencia del hub retransmite solo la información hacia el/los puerto/os que tiene asociado/s con el destinatario, mejorando el rendimiento y la seguridad de las redes.

TCP: es un protocolo estándar de control de transmisión que permite entregar los datos al destino, sin errores y en el orden en que se transmitieron en la red.

UDP: es un protocolo estándar de intercambio de datos a través de una red, a diferencia de TCP, no verifica el orden en que arriban los mensajes ni si han llegado correctamente. Su principal diferencia es que permite un intercambio más fluido ya que no realiza retransmisiones en caso de error.

ANEXO IV

PROYECTOS EDUCATIVOS CON RITA

La búsqueda y adquisición de recursos digitales en Internet, así como la producción de los propios, ofrece un gran potencial cuando de aprender en la escuela se trata. Por lo tanto, los proyectos educativos se presentan como una posibilidad didáctica para que docentes y estudiantes de diferentes años y asignaturas puedan trabajar en torno a la creación, uso y apropiación de diversos recursos con distintos objetivos y formatos.

En este anexo se trabaja en el desarrollo de proyectos educativos orientados a la programación usando RITA (Robot Inventor to Teach Algorithms): una herramienta libre, de código fuente abierto y desarrollada en el LINTI-UNLP, especialmente diseñada para programar robots virtuales que compiten en un campo de batalla.

Anexo 4: Proyectos educativos con Rita

La búsqueda y adquisición de recursos digitales en Internet, así como la producción de los propios, constituye un gran potencial cuando de aprender en la escuela se trata. Por lo tanto, los proyectos educativos se presentan como una posibilidad didáctica para trabajar, docentes y estudiantes, de diferentes años y asignaturas, en torno a la creación, uso y apropiación de diferentes recursos con distintos objetivos y formatos.

En este anexo se trabaja en el desarrollo de proyectos educativos orientados a la programación usando RITA (Robot Inventor to Teach Algorithm): una herramienta libre, de código fuente abierto y desarrollada en el LINTI-UNLP, especialmente diseñada para programar robots virtuales que compiten en un campo de batalla.

Índice

- Secuencia didáctica 1: Recursos libres en la Web
  - Actividad: Buscando recursos libres
  - Actividad: Publicando recursos libres
- Secuencia didáctica 2: Empiezo a programar con RITA
  - Actividad: Descubro los bloques de RITA
  - Actividad: El robot dibuja un rectángulo imaginario
- Secuencia didáctica 3: Estrategias de robots en RITA
  - Actividad: Y si el robot choca contra un muro, ¿qué hace?
  - Actividad: El robot que enfrenta a quien le disparó
  - Actividad: El robot enfrenta a otro, solo si el arma está lista
  - Actividad: El robot que dispara y huye

Secuencia didáctica 1: Recursos libres en la Web

Bajada

Tanto en Internet como en otros medios de comunicación digital es posible encontrar contenidos artísticos, textos, videos, audios, películas y otros tipos de información. Todos ellos, al igual que los libros, los artículos de los diarios y las pinturas que están en soporte de papel y telas están cubiertos por un marco legal que permite a los autores reservarse una serie de derechos sobre sus obras. En esta secuencia didáctica se exploran distintas formas de acceder a recursos libres que se pueden utilizar en un marco legal correcto. Se presentan dos actividades que abordan esta problemática tanto en el momento de buscar recursos digitales como cuando se publican los mismos, contribuyendo de esta manera para otros usuarios de Internet puedan utilizarlos.

Objetivos

Que los estudiantes sean capaces de:

- Conocer diferentes repositorios digitales y explorarlos, atendiendo a los marcos legales y licencias de uso.
- Producir recursos digitales y publicarlos en repositorios libres.

Actividad

Título

Buscando recursos libres

Objetivos

Que los estudiantes logren:

- Explorar algunos repositorios digitales de recursos multimediales libres.
- Identificar los filtros de búsqueda y conocer las licencias de uso, a partir de la realización de búsquedas avanzadas en la Web.

Materiales

Computadoras con acceso a Internet.

Modalidad de trabajo

Actividad individual.

Bajada para el aula

Muchas veces recurrimos a Internet para buscar información o materiales multimediales para utilizar en nuestros trabajos. Los recursos que encontramos al realizar una búsqueda simple en la Web no siempre se pueden utilizar, modificar o compartir libremente debido a que muchos de ellos fueron publicados con licencias que restringen su uso. Aunque esta característica muy pocas veces es tenida en cuenta a la hora de acceder a los mismos, se agrava si el material que producimos con ellos, lo compartimos con otras personas.

Esta actividad, organizada en 3 momentos, propone explorar diferentes sitios web que cuentan con recursos multimediales cuyas licencias de uso, en la gran mayoría de los casos, permiten utilizarlos libremente.

Momento 1: videos libres

Para comenzar la actividad, preguntar a la clase: ¿alguna vez intentaron visualizar un video en Youtube (o en un sitio similar) y el mismo ya no estaba disponible?. Es muy probable que la gran mayoría responda afirmativamente. Luego, preguntar: ¿por qué les parece que un video puede dejar de estar disponible?, ¿les aparece algún mensaje en particular?,¿cuál?. Es probable que surjan respuestas tales como:

- El video utiliza música que no cuenta con permisos para su uso libremente y seguramente se muestre una imagen similar a la Figura 1, donde se indica que se ha silenciado el audio debido a que presenta problemas de derechos de autor.

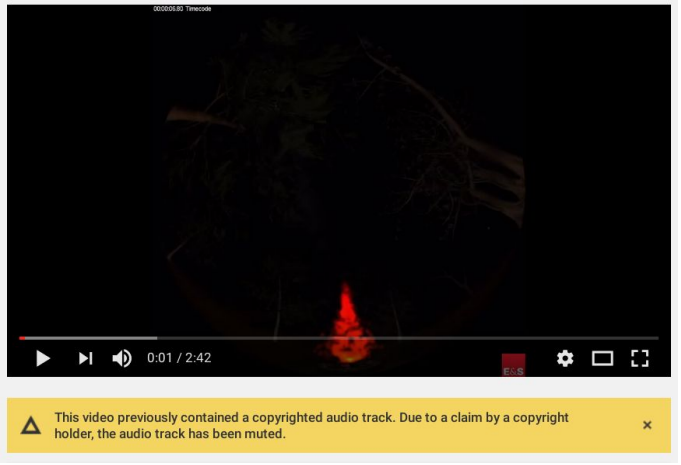


Figura 1. Imagen que muestra Youtube cuando el sonido no es libre

- El video es una copia de un contenido protegido, por ejemplo un anexo entero de una serie o una película, y aparece una imagen como la Figura 2, donde el mensaje menciona que el video fue bloqueado por violar el derecho de autor, es decir que quien lo publicó no cuenta con el permiso para hacerlo.

Figura 2. Imagen que muestra Youtube cuando se intenta acceder a un recurso que no está disponible

- El usuario que lo subió luego lo eliminó o lo ocultó. En este caso aparece una imagen similar a la mostrada en la Figura 2, pero esta vez indicando que el video fue eliminado por quien lo publicó.

Mencionar que algunos de estos problemas pueden evitarse si se usan **recursos libres**, tanto para la música de fondo de un video como para las imágenes mismas.
Luego preguntar a la clase: ¿cómo puedo saber si un video es libre o no?,¿qué significa que sea libre: puedo reproducirlo, modificarlo, distribuirlo?.
En el caso de Youtube, se dispone de una herramienta que permite filtrar las búsquedas de videos por esta característica: al presionar el botón de Filtros, es posible elegir “Creative Commons” dentro de la sección “Características”. Explicar que con el término “Creative Commons” se denomina a un conjunto de licencias[1] que permiten indicar de qué manera se pueden utilizar los recursos que se publican en Internet. Las **licencias Creative Commons** son las más utilizadas para compartir obras artísticas en Internet. Son de carácter gratuito y ofrecen a los autores varias posibilidades para citar, reproducir, crear obras derivadas y distribuir obras bajo ciertas condiciones.
El sitio oficial donde se encuentra el detalle de cada una de las licencias y su aplciación es https://creativecommons.org/licenses/?lang=es.
Proponer a los estudiantes que accedan al sitio de Youtube y busquen videos sobre el uso de recursos libres en Internet, indicándoles que algunos de ellos podrán serle de gran utilidad para las siguientes actividades del anexo.
Mencionar también que Youtube no es el único sitio donde encontrar videos libres, también se pueden encontrar en:

- https://vimeo.com/
- https://commons.wikimedia.org/wiki/Category:Videos
- http://publicdomainreview.org/

Momento 2: música libre

Pedir a los estudiantes que busquen sonidos que representen una alarma, las notas musicales de cada cuerda de una guitarra y un tema musical del género que más les guste, con los permisos necesarios para modificarlo y compartirlo. Debido a que es probable que no sepan dónde comenzar la búsqueda, se pueden sugerir buscar música en los sitios http://opsound.org o http://ccmixer.org y en http://www.youtube.com, usando un filtro de búsqueda para encontrar contenidos Creative Commons, o en http://publicdomainreview.org.
Proponer a los estudiantes que analicen cuáles recursos podrían utilizarse en una producción y cuáles no. Para ello, se pueden plantear las siguientes preguntas: ¿qué tipo de permisos cuentan sobre cada uno de los contenidos que descargaron?, ¿pueden copiar libremente el contenido?, ¿pueden modificarlo y redistribuirlo?, ¿es necesario que mencionen al autor o dueño de los derechos al usarlo?, ¿pueden utilizarlo con fines comerciales , por ejemplo cobrar para que otros accedan al mismo?.
Para responder estos interrogantes, los estudiantes deberán analizar los diferentes sitios y buscar algún texto que indique de qué manera se pueden utilizar los recursos encontrados, es decir, el tipo de licencia de uso de cada uno. La licencia indicará la forma en que el recurso puede ser utilizado: por ejemplo, puede usarse libremente si es de dominio público por su antigüedad o bien puede poseer una licencia Creative Commons o similar, donde se indica la forma de uso.

[1] Una licencia de uso es un documento que indica de qué manera se puede utilizar un recurso.

Es posible que de acuerdo al tipo de licencia de los contenidos encontrados no todos puedan distribuir versiones modificadas de algunos contenidos, no puedan utilizarlos con fines comerciales o estén obligados a mencionar al autor o dueño de los derechos si desean publicarlos.

> ¿Sabías que hay recursos que son de **dominio público**?. Los derechos de autor restringen el acceso y utilización de ciertas obras, pero los mismos expiran luego de cierta cantidad de años de fallecido el autor. A partir de transcurrido este lapso, las obras pueden ser utilizadas libremente por cualquier persona. Cuentos famosos como "Pulgarcito" y la "Bella Durmiente" se encuentran en esta categoría de obras.

Momento 3: imágenes libres

Proponer a los estudiantes que busquen en la Web imágenes de la flor nacional argentina: el ceibo. Seguramente el buscador les presentará muchos resultados. Luego, consultarles si tuvieron en cuenta los derechos de autor o permisos con los que fueron compartidas las imágenes encontradas. Probablemente algunos estudiantes exploren las opciones de búsquedas avanzadas investigando si existe alguna que les permita filtrar por derechos de autor o permisos, o quizás otros estudiantes consulten si existen algunos sitios donde comenzar la búsqueda tal como hicieron en el momento anterior.

Explicar a la clase que algunos buscadores poseen filtros que permiten restringir las búsquedas por el tipo de licencia de uso de las imágenes. En el caso del buscador de Google para encontrar imágenes con diferentes tipos de licencias, en particular, aquellas que permiten la libre distribución o modificación, existe una opción: búsqueda de imágenes-> Herramientas-> Derechos de autor. En el caso del buscador Bing, se cuenta con una opción similar en una solapa denominada Licencia.

Proponer que vuelvan a repetir la búsqueda inicial probando distintas opciones de licencias, usando el buscador que prefieran.

Analizar entre todos qué observan en los resultados de las búsquedas. Seguramente encuentren que hay muchos menos resultados cuando se filtra por derechos de autor.

Explicar que hay varios sitios en los cuales es posible encontrar imágenes libres. Los más conocidos son:

- Wikimedia: https://commons.wikimedia.org
- Openclipart:https://openclipart.org/
- Flickr[2]: https://www.flickr.com/

Proponer que accedan a OpenClipart y busquen:

- íconos que representen: una cámara fotográfica, una alarma o timbre y la acción de compartir.
- una imagen que represente una guitarra.
- imágenes que representen emociones o acciones opuestas, por ejemplo "triste"-"contento", "correr"-"caminar", "reir"-"llorar", etc.

[2] Nota: no todas las ima ǵ enes son libres, prestar atención cuando bajamos una imagen

Explicar que algunas de estas imágenes van a ser usadas más adelante en otras actividades.

¿Sabías que también hay **software libre**?. Así como vimos que las obras artísticas se pueden publicar con licencias que permiten su uso libremente, ocurre algo similar con el software. El software libre abarca a todo aquél software que puede ser copiado, modificado y distribuido libremente. Según su definición oficial. "l*os usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el softwar*e"
Podés encontrar más información en https://www.gnu.org/philosophy/philosophy.html donde se describe tanto la filosofía como los orígenes de esta corriente iniciada por Richard Stallman.
RITA una de las herramientas que usaremos en este anexo, es software libre, se puede descargar, compartir y modificar libremente.

Cierre

A modo de cierre dialogar sobre los beneficios de publicar recursos propios con licencias Creative Commons, de manera tal que otras personas que los necesiten puedan utilizarlos. Si bien la ventaja más evidente es que no se deben crear nuevos recursos desde cero cada vez que se realiza una producción, hay otros beneficios que quizás no sean tan evidentes, como ser: dar a conocer autores a través de sus obras, favorecer la cultura libre, promover que cada autor decida de qué manera quiere que sean utilizadas sus creaciones, entre otras.
Mencionar que existen otros sitios que también poseen varias referencias a recursos libres como:

- Página de Creative Commons: https://creativecommons.org/about/platform/
- Sitio oficial de NASA: https://www.nasa.gov
- Proyecto OpenStreetMap: https://www.openstreetmap.org/
- Internet Archive: https://archive.org/
- Música libre: https://www.jamendo.com/

Actividad

Título

Publicando recursos libres

Objetivo

Que los estudiantes puedan:

- Producir recursos digitales con recursos libres.
- Experimentar la publicación de materiales libres en repositorios libres.

Materiales

Computadoras con acceso a Internet.

Modalidad de trabajo

Grupos reducidos.

Bajada para el aula

Para comenzar la actividad proponer a los estudiantes que busquen imágenes y videos libres de su escuela y/o del barrio. Debido a que probablemente no encuentren gran cantidad, indicarles que en esta actividad van a generar materiales y van a publicarlo y compartirlo de manera tal que queden disponibles para que otras personas puedan utilizarlo libremente.

Si bien existen varios posibles formatos para armar este material, se propone armar un video o una página del repositorio de Wikipedia. Para esto, dividir a los estudiantes en grupos reducidos e indicar a cada grupo el tipo de trabajo a realizar: un video o una página en la Wikipedia.

Para aquellos grupos que deban realizar un video, indicarles que pueden usar las herramientas que más les guste para grabarlo y editarlo, pero que al finalizar el mismo, lo deben subir a Youtube indicando que el video posea una licencia Creative Commons que permita compartirlo y distribuirlo, para esto es necesario acceder a la pestaña “Configuración avanzada” en la herramienta “Creator studio” de Youtube, como se muestra en la Figura 3.

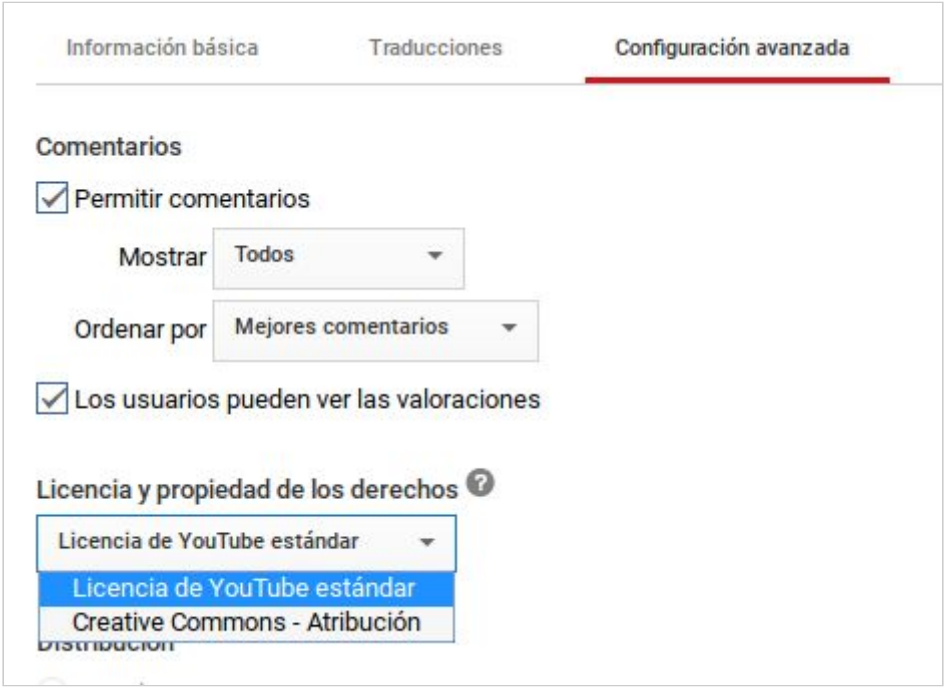


Figura 3. Pantalla de YouTube de configuración de licencias

Para aquellos grupos que deban trabajar con la Wikipedia se propone primero explorar el tutorial provisto en el sitio: https://es.wikipedia.org/wiki/Ayuda:Introducci%C3%B3n. Deben tener presente que los datos que se ingresen serán revisados y, por lo tanto deben ser verídicos, así como la redacción, la adecuada. Recordarles que muchas veces utilizamos los recursos de la Wikipedia para nuestras actividades escolares, con lo cual queremos mantener la confiabilidad de la enciclopedia.

En todos los casos, recordar a los estudiantes que pueden incluir fotos, videos y grabaciones propias, y que si incluyen materiales que obtuvieron de una búsqueda en Internet, deben asegurarse que los mismos cuentan con los permisos adecuados para ser incluidos en sus respectivos trabajos.

Entregar a los grupos la Ficha N° 1 donde encontrarán una guía para realizar la consigna propuesta.

Cierre

A modo de cierre dialogar con los estudiantes respecto de lo que significa promover la libertad de distribuir y modificar los trabajos creativos en Internet u otros medios digitales. Explicar que ésta es una corriente de pensamiento a la que se conoce como **cultura libre.** Según la Wikipedia, "la cultura libre es una corriente de pensamiento que promueve la libertad en la distribución y modificación de trabajos creativos basándose en el principio del contenido libre para distribuir o modificar trabajos y obras creativas, usando Internet así como otros medios". La cultura libre abarca tanto a los contenidos publicados bajo licencias Creative Commons, como al software libre y las obras de dominio público.

Lawrence Lessig, el creador de las licencias Creative Commons, escribió un libro denominado "Cultura libre", el cual se subtitula "Cómo los grandes medios usan la tecnología y las leyes para encerrar la cultura y controlar la creatividad". El libro está disponible en muchos formatos y puede descargarse de https://www.derechosdigitales.org/culturalibre/cultura_libre.pdf

Ficha N° 1

Ficha para el estudiante

Publicando recursos libres

Anexo N° 4: Proyectos educativos
Secuencia didáctica 1: Recursos libres en la Web

Bajada

Vamos a producir un material digital sobre la escuela y tu barrio para luego publicarlo como un recurso libre en Internet. En la clase te indicarán si tenés que crear un video para luego subirlo a Youtube o si trabajarás en la elaboración de una página de la Wikipedia.

A) Buscá en la Web los recursos que vas a necesitar para armar el material: fotos, textos, audios y videos. Tené en cuenta que todos los recursos que se utilicen deben tener permiso de sus autores para ser usados y compartidos. También podés utilizar recursos propios: fotos, audios o videos. Pero no te olvides que estos recursos también deberán estar publicados como recursos libres para indicar de qué manera pueden ser utilizados. Acá tenés algunos sitios donde buscar:

Imágenes	https://openclipart.org/ https://www.flickr.com/
Sonidos	https://www.jamendo.com/ http://opsound.org http://ccmixer.org
Videos	https://vimeo.com/ https://www.youtube.com https://commons.wikimedia.org/wiki/Category:Videos
Varios	https://www.nasa.gov https://www.openstreetmap.org/ Internet Archive: https://archive.org/ http://publicdomainreview.org. https://commons.wikimedia.org

- Si te asignaron la creación de un video, podés utilizar la herramienta que tengas disponible para generarlo e incluir todos los recursos que hayas encontrado o producido en la parte A) de esta ficha. Al subir el video, no olvides indicar que el video tiene licencia Creative Commons, de manera tal que otras personas pueden utilizarlo.
- Si te asignaron publicar en la Wikipedia, antes de comenzar, explorá el tutorial provisto en el sitio: https://es.wikipedia.org/wiki/Ayuda:Introducci%C3%B3n donde se muestra cómo editar un contenido. Es importante que la información que ingreses sea correcta y cuides la redacción dado que estos contenidos son revisados luego.

Secuencia didáctica 2: Empiezo a programar con RITA

Bajada

RITA (Robot Inventor to Teach Algorithm) es una herramienta didáctica, cercana a la cultura adolescente que promueve la enseñanza de la programación mediante un abordaje lúdico. Con RITA los estudiantes programan con bloques juegos de robots virtuales que compiten en un campo de batalla y cuyo desafío es programar estrategias de robots ganadores. Las batallas de robots en RITA no promueven la violencia, no involucran personas, no contienen sangre ni plantean situaciones de enfrentamiento racial, el objetivo es promover la competencia de estrategias, programas, en un sentido positivo. RITA es una herramienta libre, de código fuente abierto, disponible para su descarga desde https://github.com/vaybar/RITA.

En esta secuencia didáctica se introducen los bloques de RITA, sus usos y cuestiones propias del juego. La Figura 4 muestra la anatomía de un robot de RITA.

Figura 4. Anatomía de un robot en RITA

Los bloques de RITA representan acciones que el robot realiza en el campo de batalla. Varias de estas acciones son para desplazarse en el campo de batalla y en respuesta a eventos que ocurren en su entorno, por ejemplo cuando se choca contra un muro, cuando otro robot lo ataca, entre otros. Asimismo, RITA cuenta con un amplio conjunto de bloques para realizar operaciones aritméticas, trigonométricas, lógicas, manipular variables, etc.

Esta secuencia didáctica está organizada en 2 actividades: la primera propone a los estudiantes explorar y reconocer cómo funcionan los bloques más significativos de RITA y la segunda programar la primera estrategia de un robot virtual.

Objetivos

Que los estudiantes logren:

- Comprender el ambiente de programación y de ejecución de RITA.
- Introducirse en la programación de estrategias de robots en RITA.

Actividad

Título

Descubro los bloques de RITA

Objetivos

Que los estudiantes puedan:

- Identificar los bloques de RITA, su función y la ubicación en el menú de bloques.
- Comprender cómo giran y se desplazan los robots en el campo de batalla de RITA.

Modalidad de trabajo

En grupos reducidos.

Materiales y recursos utilizados

Una computadora con el programa RITA instalado y la Ficha N° 2.

Bajada para el aula

Para comenzar la actividad, entregar a cada estudiante la Ficha N° 2. Advertirles que deben completar la tabla usando RITA y que los bloques que se presentan en dicha tabla serán luego usados en la construcción de la estrategia de un robot virtual.
Organizar a los estudiantes en grupos reducidos, se recomienda que estén conformados por 2 o 3 estudiantes.
Solicitar a los grupos que abran el programa RITA. Advertirles que para comenzar, RITA les pedirá que creen un robot nuevo o usen uno creado previamente, como muestra la ventana de diálogo de la Figura 5. Indicar a los estudiantes que para esta actividad elijan crear un robot nuevo y que ingresen el nombre que deseen.

Figura 5 - Ventana de creación de un robot

Pedir a los estudiantes que analicen la disposición de los elementos de la pantalla principal de RITA. Acercarse a la computadora de cada grupo y mostrarles cada una de estas partes: una barra de menú en la parte superior, un conjunto de bloques disponibles del lado izquierdo, un área de trabajo, un tacho de basura, un ícono en la parte inferior izquierda para

probar el robot en el campo de batalla y un minimapa (visualizador en miniatura del área de trabajo). La Figura 6 muestra la pantalla de RITA que representa el área de trabajo.

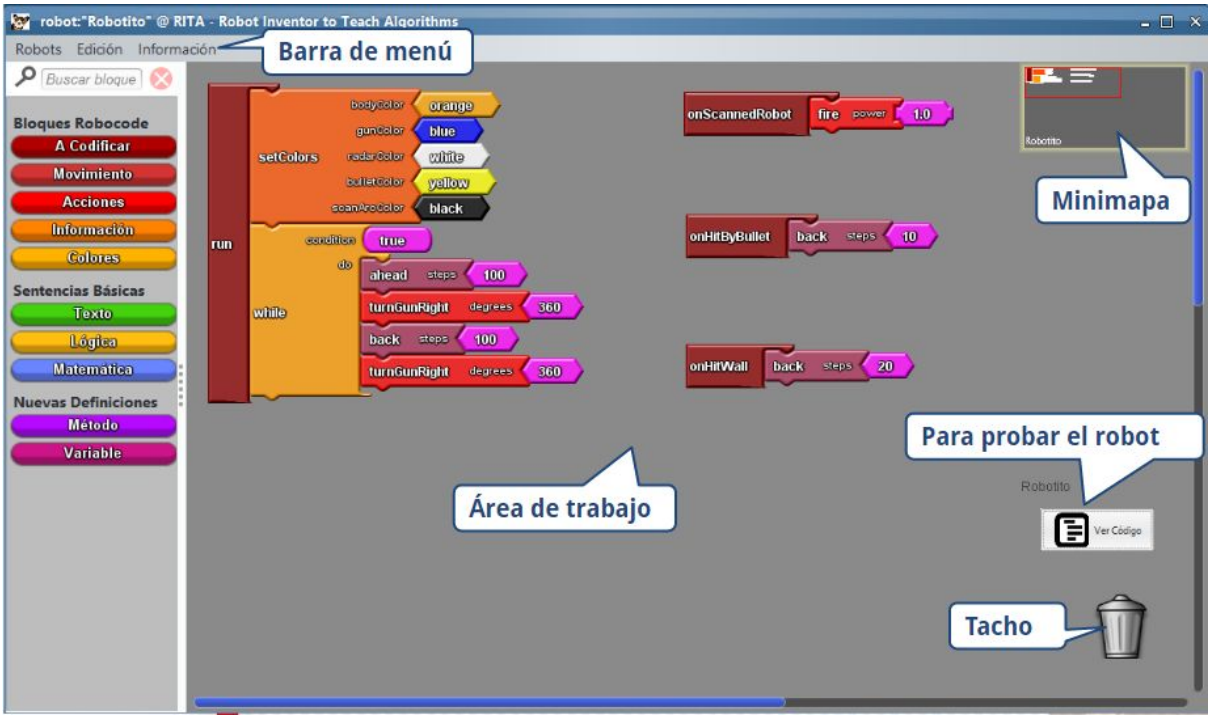


Figura 6 - Área de trabajo de RITA

Pedir a los estudiantes que averigüen dónde están los bloques de la Ficha Nº 2 y qué hace cada uno de ellos. En este punto advertir que tienen 10 conjuntos de bloques donde buscarlos y que el color del bloque les servirá de guía para encontrarlo. Considerar que al posicionar el mouse sobre los bloques de RITA aparece una leyenda que explica en castellano su funcionamiento.

Recorrer las mesas de trabajo de los diferentes grupos y preguntar: ¿qué bloques les resultan más simples de entender?, ¿cuáles más complejos?. Anotar aquellos que resultaron más complejos con la intención de explicarlos en una puesta en común.

Dar un tiempo para completar la Ficha Nº 2 y luego hacer la puesta en común tomando como insumo las anotaciones que hechas. A continuación, preguntar: ¿qué piensan qué hace el bloque “run”?, ¿para qué sirve?, ¿y el bloque “onHitWall”?. Posiblemente a los estudiantes les resulte difícil responder estas preguntas aunque podrían aproximarse a la idea que el bloque “run” es similar a la programación del evento: “When the world starts, do” de Alice. En este momento, explicar que el bloque “run” contiene las acciones que se ejecutan mientras no ocurran eventos en el entorno del robot, como por ejemplo chocar con otro robot, chocar con un muro, detectar otro robot, entre otros. Sobre el funcionamiento del bloque “onHitWall”, explicar que permite incorporar las acciones que realiza el robot cuando choca contra un muro.

En relación a los restantes bloques de la ficha, se puede explicar a la clase que los usarán para implementar la estrategia del robot y se ubicarán adentro del bloque “run” y “onHitWall”. Seguidamente preguntar: ¿por qué piensan que hay bloques que usan grados, como por ejemplo el “bearGunTo”, “turnTo” y “turnGunLeft”?. En este momento se puede explicar que

los robots se mueven en un campo de batalla representado en la pantalla como un plano y que usan un sistema de coordenadas (x,y) como muestra la Figura 7. El tamaño del campo de batalla dependerá de la resolución del monitor. Para girar el robot se necesita información de los grados a girar, esta información puede ser absoluta o relativa a la orientación actual del robot o de su arma. Algunos bloques hacen giros absolutos, por ejemplo 90º, 180º, 60º, de acuerdo a como se muestra en la Figura 7 y otros hacen giros relativos por ejemplo los que incluyen en su nombre la palabra “bear” .

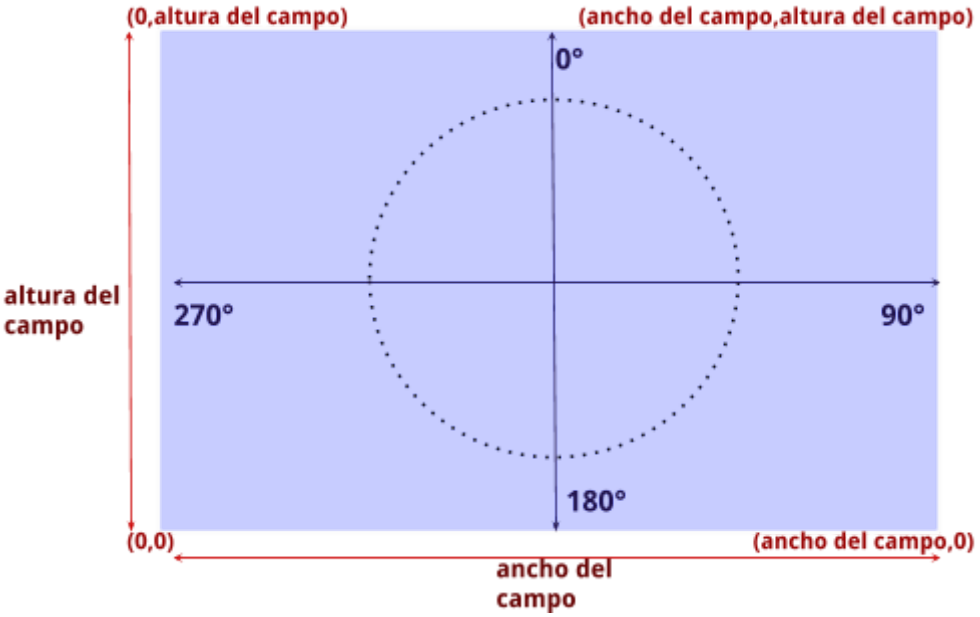


Figura 7 - Sistema de coordenadas en RITA. La altura predeterminada del campo es 600 y el ancho del campo es 800

La solución de la ficha propuesta es la siguiente:

Bloque	Grupo de bloques	Sirve para ...
	Acciones	Disparar con el arma.
	Información	Recuperar el valor de la energía del robot.
	Lógica	Iterar la cantidad veces indicada en “times”.
	Acciones	Girar el arma a la izquierda considerando un ángulo absoluto.
	Matemáticas	Representar número 1.

	A Codificar	Contener los bloques que representan el comportamiento del robot cuando no está respondiendo a eventos de su entorno.
	Matemáticas	Sumar dos valores.
	Movimiento	Girar el robot considerando un ángulo absoluto.
	Acciones	Girar el arma considerando un ángulo relativo a la ubicación actual del arma.
	Información	Recuperar el valor de la posición en el eje X del robot, en el campo de batalla.
	Lógica	Resolver la operación lógica “y”.
	Movimiento	Girar el robot a la derecha considerando un ángulo relativo a la posición actual del robot.
	A Codificar	Contener a los bloques que representan las acciones que realizará el robot cuando choca contra un muro.
	Variables	Definir una variable del programa.

Cierre

A modo de cierre se puede explicar que:

- Los robots en RITA se mueven en un plano de 2 dimensiones (x,y) cuyo tamaño es 800x600 (se trabajará con esta configuración).
- Los bloques se usan para programar un robot.
- Hay bloques que se usan para contener la estrategia de combate del robot como “run” y “onHitWall” que se trabajaron en esta actividad, a lo largo del anexo se verán otros.
- Hay bloques que se usan para darle al robot indicaciones de cómo moverse, por ejemplo: “turnTo”, “turnRight”; otros para obtener información de si mismo, por ejemplo: “gunReady”, “energy” y otros de su entorno, por ejemplo: “fieldWidth”.
- Hay bloques que implementan las estructuras de control, como “if...then”, “for...times..do”, “while…” y otros que permiten hacer operaciones lógicas como comparaciones: “==”, “>=” y otros aritméticas como “+”, “*”.

Ficha N° 2

Ficha para el estudiante

Descubriendo los bloques de RITA

Anexo N°4: Proyectos educativos
Secuencia didáctica 2: Empiezo a programar con RITA

Bajada

Explorá RITA y averiguá qué hacen y dónde están ubicados los bloques del siguiente cuadro. Para cada bloque del cuadro, anotá el grupo de bloques donde lo encontraste e indicá para qué pensás que sirve.

Bloque	Grupo de bloques	Sirve para ...
		
		
		
		
		
		
		
		
		
		
		
		

 onHitWall		
 nombreDeVariable		

Actividad

Título

El robot dibuja un rectángulo imaginario

Objetivos

Que los estudiantes puedan:

- Programar y ejecutar un robot sencillo en RITA.
- Comprender a partir de la puesta en acción, los bloques de desplazamiento y giro de RITA.
- Comprobar que hay varias soluciones para un mismo problema y que es posible mejorar las soluciones obtenidas.

Modalidad de trabajo

Actividad en grupos reducidos.

Materiales y recursos utilizados

Una computadora con RITA instalado y la Ficha N° 3.

Bajada para el aula

Para comenzar la actividad, entregar a cada estudiante la Ficha N° 3 y organizarlos en grupos. Indicar a los estudiantes que el objetivo de la actividad es construir un robot que dibuje un rectángulo imaginario, sin trazo, indefinidamente, empezando desde la esquina inferior izquierda y con el robot orientado hacia el norte. Las medidas del rectángulo son 500 de alto y 650 de ancho. Advertir que el robot programado competirá en el campo de batalla con un robot muy tranquilo llamado Mambo.

Momento 1: exploración

Indicar a los grupos que comiencen a trabajar y darles un tiempo para que resuelvan la consigna dada, podrían ser 20 minutos o lo que se considere adecuado en función del desempeño de los estudiantes.

Seguidamente, preguntar: ¿dónde ubican los bloques que dibujan el rectángulo? y ¿cómo piensan que podrían probarlo?. En relación a la primera pregunta, se espera que los estudiantes adviertan que los bloques que dibujan se ubican en el bloque “run”. Se puede recuperar la explicación del propósito del bloque “run” de la actividad previa.

En lo que refiere a la segunda pregunta, probablemente no todos los grupos encuentren respuesta. Indicarles que haciendo “clic” en el botón “Ver Código”, que se encuentra en el “área de trabajo”, se desplegarán 2 opciones, y que deben elegir la opción “Compilar y Ejecutar”[3]. Al elegir esta opción se abrirá una ventana de diálogo como muestra la Figura 8. Explicar que deben seleccionar los robots que competirán en el campo de batalla con el que se está programando. A partir de esta ventana se puede definir la ubicación y orientación inicial de cada robot, como muestra la Figura 9.

[3] En el anexo 8 se trabajará el concepto de compilación.

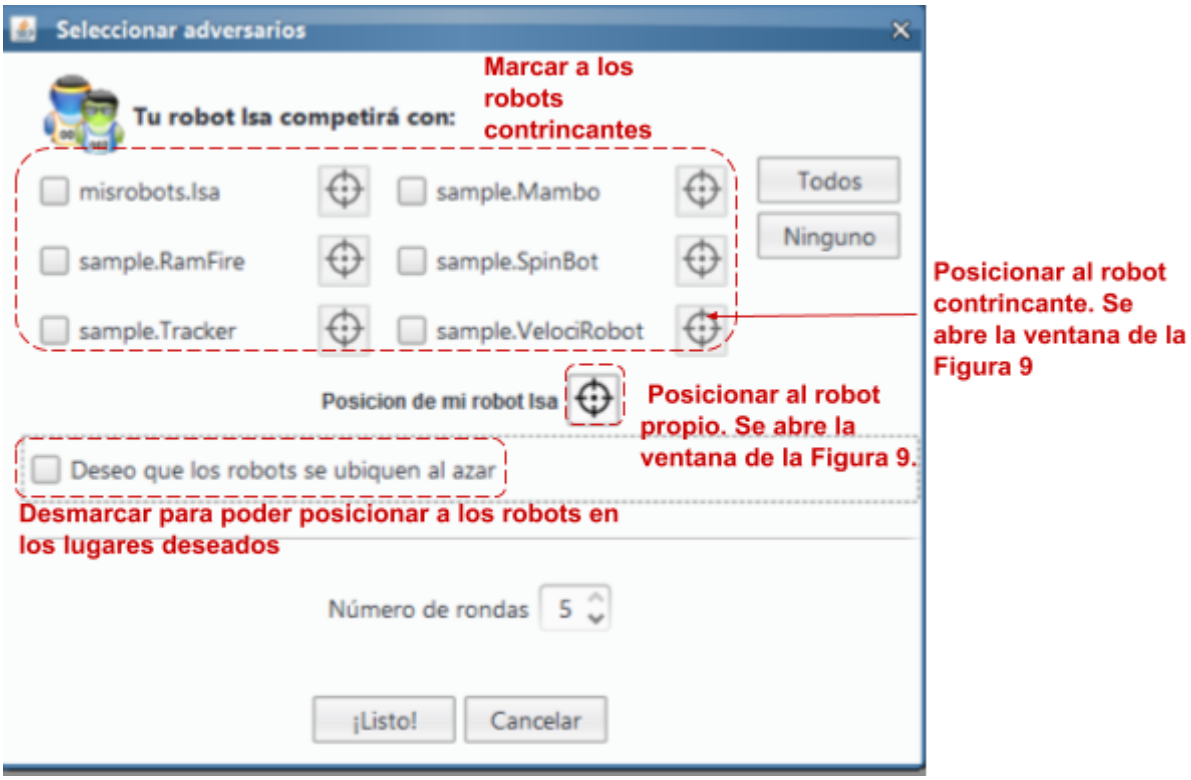


Figura 8 - Ventana de selección de robots que competirá con el que se está programando

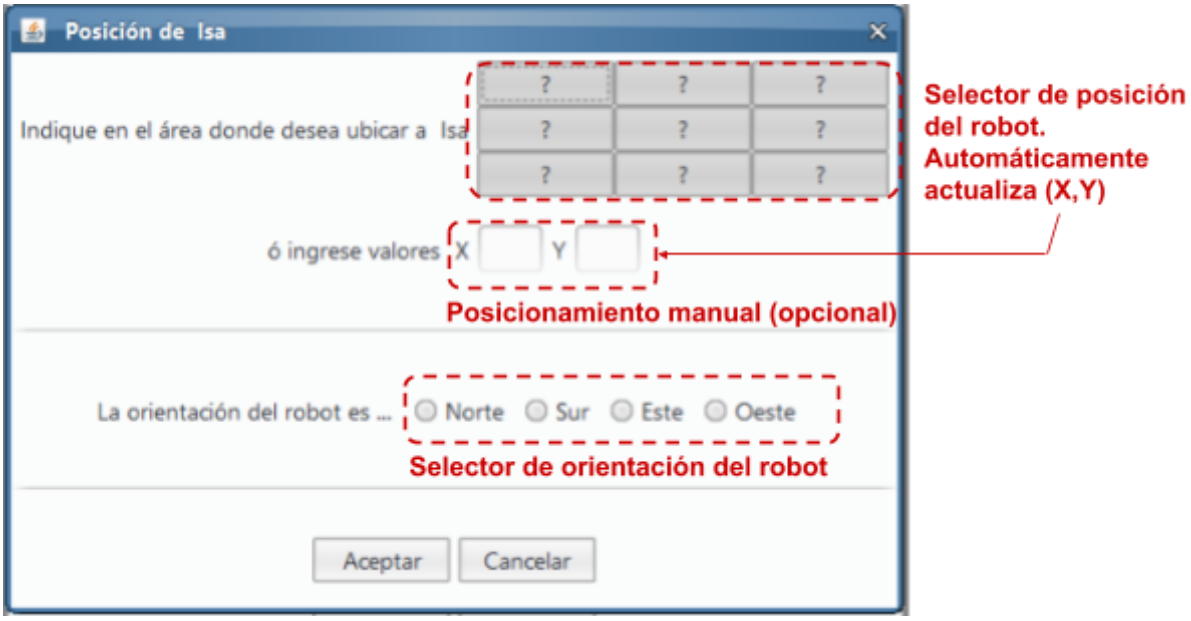


Figura 9 - Ventana de posicionamiento de los robots

Seguidamente explicar que, por tratarse de una competencia de robots, deberán tener las siguientes consideraciones para iniciar la competencia:

- Siempre se necesita un contrincante para hacer una competencia: al menos hay que elegir un robot de la ventana de selección de robots (Figura 8). Si no eligen contrincante, no hay competencia y no hay juego.

- En esta actividad hay que elegir un contrincante que no interfiera con el que dibuja el rectángulo: lo tiene que dejar dibujar y por ejemplo no chocarlo. Proponer que compitan con “Mambo” que no hace nada.
- En esta actividad es necesario configurar la posición y orientación inicial del robot que dibuja el rectángulo imaginario como lo indica la Figura 10: esquina inferior izquierda del campo de batalla, en la posición (132, 39) y orientado hacia el norte.

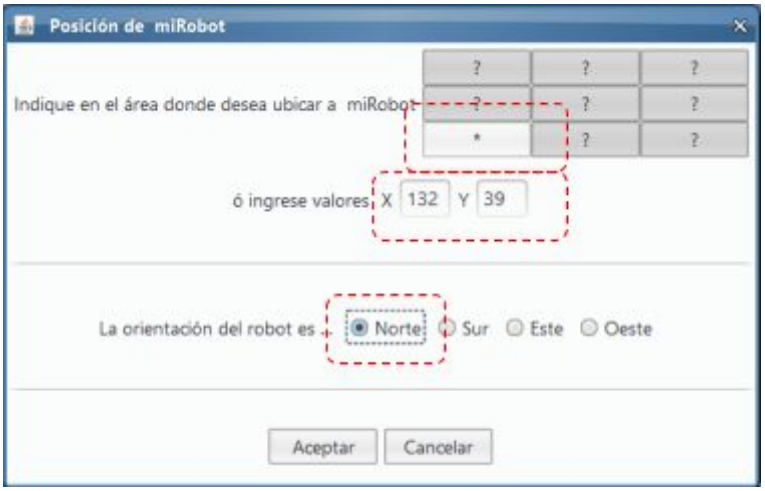


Figura 10- Configuración de la posición inicial del robot que dibuja el rectángulo imaginario

- Finalmente configurar la posición inicial del contrincante Mambo: centro del campo de batalla (centro de la grilla en la Figura 10). Los valores de posición y orientación no afectarán el resultado de la actividad, se puede elegir cualquiera.

Momento 2: poner en acción el robot

Proponer que cada grupo pruebe su robot. Probablemente algunos grupos tuvieron dificultades con los valores de los ángulos de giro y el robot no hace lo que se espera. En este momento se puede mostrar con un dibujo en el pizarrón que el robot que dibuja el rectángulo imaginario realiza una trayectoria similar a la de la Figura 11.

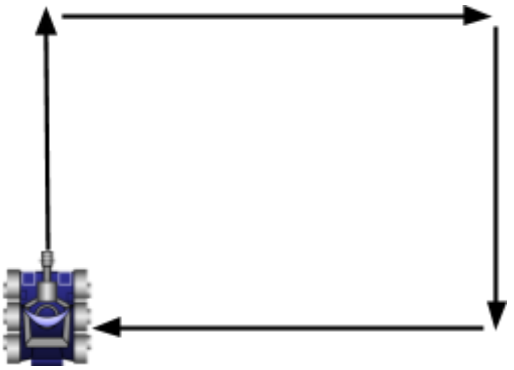


Figura 11 - Movimiento esperado del robot que dibuja el rectángulo imaginario

En este momento preguntar: ¿qué métodos usaron para girar?, ¿y qué ángulos?. Recoger las respuestas recibidas para sistematizar las siguientes ideas:

- si eligieron el bloque “turnTo” para girar el robot, deben ajustar el ángulo en los distintos momentos de giro a 0º, 90º, 180º y 270º. El ángulo de giro es absoluto.
- si usaron el bloque “turnRight” simplemente deberán ajustar el ángulo de giro a 90º en cada momento. El ángulo de giro es relativo a la posición del robot.

Preguntar: ¿el robot es capaz de dibujar rectángulos de manera repetitiva?. Es muy probable que no funcione repetitivamente y que surja el problema que una vez que se terminó de dibujar el primer rectángulo, el robot no quede correctamente orientado para dibujar el próximo. Pedir que analicen los bloques del método “run” y luego preguntar: ¿dónde está el problema?, ¿cómo se soluciona?.

Aquí pueden surgir varias propuestas de solución. Una de ellas es colocar como primer bloque del bloque “run”, al bloque “turnTo(0”): 

que gira al robot a 0º en forma absoluta (sistema de coordenadas de la Figura 4).

Seguidamente se puede preguntar: ¿qué bloque usaron para el robot avance?. Se espera que respondan que el bloque “ahead” se usa para avanzar con el robot una determinada cantidad de pasos. En este ejemplo los pasos del “ahead” son 500 en el eje “Y” y 650 en el eje “X”.

Cierre

A modo de cierre se puede reflexionar junto con con los estudiantes que:

- El problema propuesto tiene más de una solución y que en cualquiera de ellas se usan los bloques “ahead”, “turnTo” ó “turnRight”.
- En el caso de usar el bloque “turnTo”, el ángulo de giro es absoluto considerando el sistema de coordenadas de RITA, por tanto es diferente en cada esquina.
- En el caso de usar el bloque “turnRight” el giro es relativo a la orientación actual del robot y solo necesita girar 90º.
- En la puesta a prueba de la solución aparecieron cuestiones que no se contemplaron durante el desarrollo, esto permite la mejora continua de la solución.

Finalmente preguntar a los estudiantes ¿cuántos bloques diferentes usaron en la solución además de los de avance y giros?. Seguramente aquí surjan distintas respuestas:

- algunos grupos dibujaron el robot usando estructuras de control iterativas, iterando 4 veces el boque de avance y giro, y
- otros grupos construyeron una solución paso a paso.

Tres soluciones del robot que dibujan el rectángulo imaginario pueden descargarse de:
http://linti.unlp.edu.ar/anexo6/Sec2act2res1
http://linti.unlp.edu.ar/anexo6/Sec2act2res2
http://linti.unlp.edu.ar/anexo6/Sec2act2res3

Ficha N° 3

Ficha para el estudiante

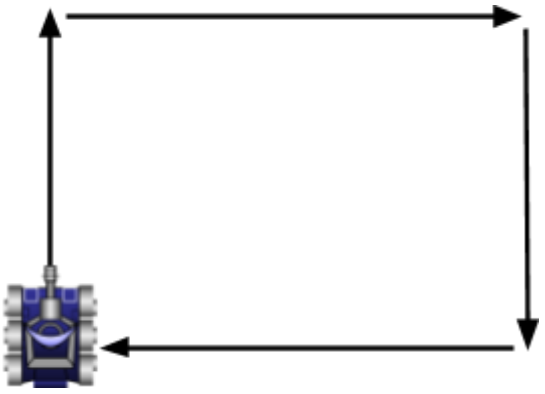
El robot dibuja un rectángulo imaginario

Anexo N° 4: Proyectos educativos
Secuencia didáctica 2: Programando con RITA

Bajada

Construí un robot en RITA que dibuje un rectángulo imaginario (sin trazo) indefinidamente, empezando desde la esquina inferior izquierda y con el robot orientado hacia el norte. Las medidas del rectángulo son 500 de alto y 650 de ancho. Seguí las indicaciones dadas en clase.

La trayectoria del robot será similar al siguiente gráfico:



Secuencia didáctica 3: Estrategias de robots en RITA

Bajada

Los robots en RITA despliegan su estrategia en el campo de batalla cuando compiten con otros robots. Pero….¿qué es la estrategia del robot? y ¿dónde la programamos?, las respuestas a estas preguntas serán la guía de esta secuencia didáctica. La estrategia es el comportamiento del robot, es lo que hace en el campo de batalla, y en varias situaciones son reacciones a eventos que ocurren en su entorno, el campo de batalla.

En esta secuencia didáctica se retoman los conceptos de programación orientada eventos introducidos en los anexos previos. Los robots de RITA reaccionan a eventos que pueden ocurrir en su entorno, el campo de batalla, como: chocar contra un muro, chocar contra un robot, ser alcanzado por el disparo de un robot y, escanear y detectar la presencia de un robot.

Cada actividad de esta secuencia didáctica introduce reglas de juego propias de RITA y propone analizar los movimientos del robot, tanto de su cuerpo como de su arma, con especial atención en la aplicación de conceptos de ángulos y sistemas de coordenadas cartesianas aprendidos en matemática. Asimismo en estas actividades se trabaja en la programación de estrategias defensivas, ofensivas o simplemente acciones de supervivencia en el campo de batalla.

Objetivos

Que los estudiantes logren:

- Comprender qué es un evento en RITA y cómo responden los robots a los eventos.
- Analizar problemas en los que ponen en acción conocimientos sobre coordenadas cartesianas y trigonometría aprendidos en matemática.
- Construir tanto individual como colectivamente, estrategias de robot que sean capaces de competir con otros robots en el campos de batalla de RITA.

Actividad

Título

Y si el robot choca contra un muro, ¿qué hace?

Objetivos

Que los estudiantes puedan

- Introducirse en el mundo de los eventos de RITA.
- Consultar los bloques de información para programar soluciones.
- Comprender cómo se desplaza y gira un robot en el campo de batalla de RITA.
- Programar una estrategia supervivencia.

Modalidad de trabajo

Actividad en grupos reducidos.

Materiales y recursos utilizados

Una computadora con RITA instalado y la Ficha N° 4.

Bajada para el aula

Para comenzar la actividad, entregar a cada estudiante la Ficha N° 4 y organizarlos en grupos. Luego indicar que en esta actividad programarán un robot que reacciona cuando choca contra uno de los muros del campo de batalla. Se puede explicar que un muro es un borde del campo de batalla.

Advertir a los estudiantes que se espera que el robot sea lo suficientemente inteligente como para continuar en el juego cuando choque contra un muro. En este momento explicar que cada vez que el robot choca contra un muro se detiene momentáneamente y es vulnerable a ataques, por lo tanto se debe programar una estrategia de supervivencia, que lo haga salir rápidamente de ese lugar. Seguidamente, preguntar: ¿qué bloque reacciona cuando “el robot choca contra un muro”?. Se espera que recuerden de la secuencia didáctica 2 que cada vez que el robot choca contra un muro se ejecuta el bloque “onHitWall”. En ese momento el robot deja de hacer lo que estaba haciendo y atiende el evento ocurrido.

Realizar una serie de preguntas para analizar colectivamente el problema a resolver.

Comenzar preguntando: ¿cuáles son los muros con los que podría chocar el robot?. Se espera que identifiquen los siguientes casos:

- el robot choca el muro norte
- el robot choca el muro sur
- el robot choca el muro este
- el robot choca el muro oeste

Luego preguntar: ¿cómo piensan detectar contra cuál de los muros podría chocar el robot?. Posiblemente no surja rápidamente la respuesta, en ese caso sugerir a los estudiantes que piensen en la posición actual del robot, para ello hacer una pregunta más simple: ¿cómo saben cuál es la posición actual del robot?. Se espera que recuerden que en la secuencia didáctica 2 analizaron el bloque “robotX” del grupo de “Información” y que éste devuelve la posición del robot en el eje X del campo de batalla. De manera análoga, pueden consultar la posición en el eje Y con el bloque “robotY”. Con estos datos se puede determinar la posición actual del robot en el campo de batalla. Seguidamente preguntar: ¿cuándo piensan que el robot está cerca de un muro?. Las propuestas pueden ser variadas, acordar con los estudiantes que el robot está cerca de un muro, si está a menos de 50 pasos.

Retomar la pregunta ¿cómo piensan detectar contra cuál de los muros podría chocar el robot?. Se puede orientar la respuesta dibujando en el pizarrón un gráfico similar a la Figura 12 y analizando con toda la clase las zonas de peligro de choque:

- si el valor de robotX es menor que 50, puede chocar contra el muro oeste.
- si el valor de robotX es mayor a 750, puede chocar contra el muro este.
- si el valor de robotY es menor que 50, puede chocar contra el muro sur.
- si el valor de robotY es mayor que 550, puede chocar contra el muro norte.

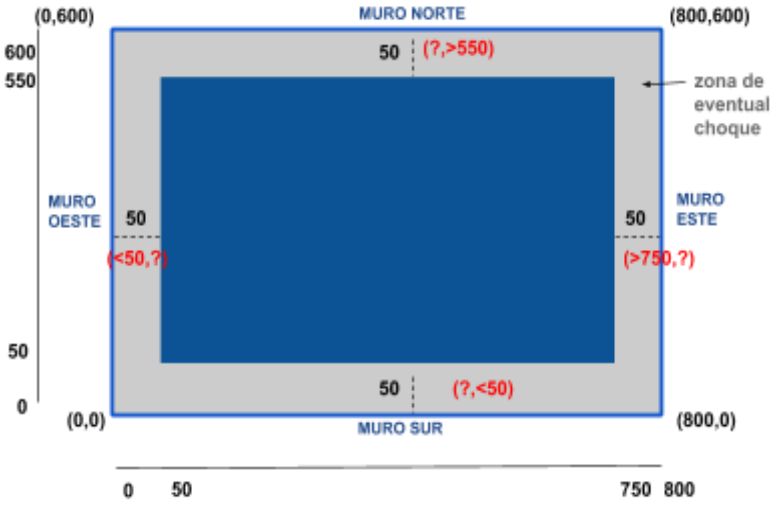
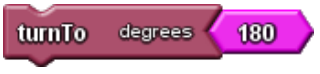


Figura 12 - Zona de choques contra muros

En este punto se puede explicar que el campo de batalla tiene un tamaño predeterminado de 600x800, sin embargo podría configurarse para adoptar otras medidas. Por este motivo, dentro del bloque “información” existen bloques que pueden ser consultados para conocer el tamaño real del campo de batalla:

- fieldWidth: ancho del campo de batalla
- fieldHeight: alto del campo de batalla

Advertir a los estudiantes que para la actividad propuesta se pueden usar los bloques de “Información” para consultar el tamaño del campo de batalla o directamente los valores predeterminados (600,800).

Una vez que los estudiantes pueden identificar el muro contra el que choca el robot, preguntar: ¿qué hace el robot cada vez que choca contra un muro?. Aquí se puede retomar la consigna inicial: el robot debe continuar el juego y salir rápidamente de ese lugar. Pueden surgir varias ideas, probablemente coincidan en cambiar de dirección del robot y continuar avanzando. Para analizar esta situación, proponer que consideren que el robot choca contra el muro norte y luego preguntar: ¿cómo posicionarían al robot en la dirección opuesta?. Se espera que adviertan que tienen que girar el robot, sin embargo no es sencillo saber cuál es dirección opuesta. La forma más simple de girar al sur es usar el bloque “turnTo” que posiciona el robot de manera absoluta, indicándole que gire a 180º: 

De manera análoga deberían pensar cómo cambiar de dirección cuando el robot choca contra el resto de los muros. Preguntar ¿cuáles son los giros para la dirección norte, este y oeste?. Se espera que adviertan que 0º, 90º y 270º son los valores que deben indicarle al bloque “turnTo” para girar al norte, este y oeste respectivamente.

Dar un tiempo para que todos los grupos desarrollen su solución y luego hacer una puesta en común. Advertir a la clase que el bloque “run” no es relevante en esta actividad y por lo tanto puede mantenerse la versión que RITA propone automáticamente.

La Figura 13 muestra una solución posible de un robot que detecta si chocó contra el muro “este” y en ese caso cambia a la dirección opuesta, “oeste”. En este ejemplo se tomó la información del ancho del campo de batalla del bloque “fieldWidth”. Advertir a la clase que podría hacerse la comparación: “robotX>750” si el campo de batalla es el predeterminado.

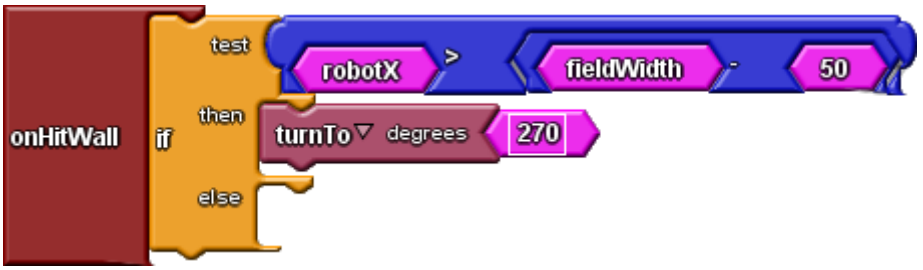


Figura 13-Estrategia de Supervivencia: el robot detecta si chocó contra el muro “este”

Una solución de esta actividad está disponible en:http://linti.unlp.edu.ar/anexo6/Sec3act1res1

Proponer a los estudiantes que ejecuten el robot y comprueben si la estrategia desarrollada funciona correctamente, en los 4 casos de choque. Se puede elegir a cualquier robot como contrincante dado que el robot desarrollado no interactúa con éste. Se recomienda que para realizar las pruebas, se ejecute 4 veces en posiciones y orientación de inicio diferentes de modo que en cada ejecución se provoque el choque contra cada uno de los muros. A modo de ejemplo:

- Para provocar el choque contra el muro norte, posicionar al robot usando la grilla de posicionamiento y seleccionar cualquier celda de la fila superior, con orientación norte.
- Para provocar el choque contra el muro este, posicionar al robot usando la grilla de posicionamiento y seleccionar cualquier celda de la columna derecha, con orientación este.
- Para provocar el choque contra el muro sur, posicionar al robot usando la grilla de posicionamiento y seleccionar cualquier celda de la fila inferior, con orientación sur.
- Para provocar el choque contra el muro oeste, posicionar al robot usando la grilla de posicionamiento y seleccionar cualquier celda de la columna izquierda, con orientación oeste.

Cierre

A modo de cierre de la actividad, hacer una puesta en común de las soluciones propuestas por los estudiantes, compararlas y explicar que:

- El bloque “onHitWall” indica al robot que chocó contra un muro. Todos los bloques contenidos en él representan la estrategia del robot cada vez que choca contra un muro. En el ejemplo, el robot cambia de dirección evitando ser un blanco fácil. Es una estrategia de supervivencia.
- Los bloques “fieldWidth” y “fieldHeight” permiten abstraernos del tamaño del campo de batalla. Si los usamos, nuestro robot va a funcionar para cualquier tamaño de campo de batalla. Si bien los valores predeterminados son 600 y 800, estos valores son configurables y podrían cambiar. La solución que usa los bloques “fieldWidth” y “fieldHeight” es más general.

Ficha N°4

Ficha para el estudiante

Y si el robot choca contra un muro, ¿qué hace?

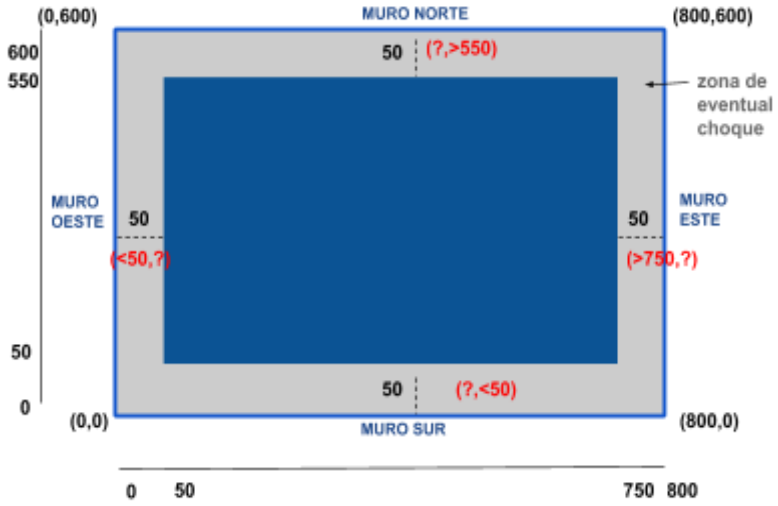
Anexo N° 4: Proyectos educativos
Secuencia didáctica 3: Estrategias de robots en RITA

Bajada

Esta actividad no pretende que tu robot se defienda o ataque, simplemente que circule sobre el campo de batalla. Cuando el robot se mueve y choca contra un muro, se detiene y si no tiene una estrategia pensada, simplemente seguirá tratando de moverse en la misma dirección sin poder lograrlo. Esta es una situación potencialmente peligrosa, otros robots podrían atacarlo fácilmente.

Por tanto, en caso de chocar contra un muro, tu robot debe contar con alguna estrategia para esquivarlo, ¡en definitiva este muro es un obstáculo!. ¿Te animás a agregarle inteligencia a tu robot para seguir por otro camino cuando choque contra alguno de los muros?

El siguiente gráfico puede ayudarte a analizar las zonas de peligro de choque contra los muros.



Actividad

Título

El robot que enfrenta a quien le disparó

Objetivos

Que los estudiantes puedan:

- Introducirse en la programación de estrategias de robots que interactúan con otros robots.
- Conocer y comprender el tipo de información que brinda el entorno de RITA para programar estrategias.
- Programar una estrategia defensiva.

Modalidad de trabajo

Actividad en grupos reducidos.

Materiales y recursos utilizados

Una computadora con el programa RITA instalado y la Ficha N° 5.

Bajada para el aula

Para comenzar la actividad, entregar a los estudiantes la Ficha N° 5 y organizarlos en grupos. Explicar a la clase que el objetivo de esta actividad es programar la estrategia de un robot defensivo, que dispara cuando es atacado por otro robot.

Dar un tiempo para que analicen el problema y exploren entre los bloques disponibles en RITA. Se puede orientar la solución con una serie de preguntas, como las siguientes:

- ¿Cómo saben de dónde viene el ataque?. Probablemente la respuesta no surja fácilmente, en ese momento explicar que entre los bloques de “Información” está el bloque “hitByBulletAngle” que devuelve el ángulo de dónde provino el disparo. Construir la siguiente idea: conociendo de dónde proviene el disparo, se puede deducir dónde está el robot que le disparó y defenderse, atacándolo.
- ¿Qué acción realizarían para apuntar hacia el robot contrincante?. Retomar la idea previa del uso del bloque “hitByBulletAngle”. Posiblemente alguno de los estudiantes propondrán apuntar el arma en la dirección de ese ángulo. Luego, preguntar: ¿cuál es el bloque que permite al robot girar el arma un ángulo determinado?. Se puede ayudar a construir la idea que en los bloques de la categoría “Acciones” se encuentran todas las acciones que puede hacer un robot. Se espera que respondan que el bloque “turnGunTo” permite al robot girar su arma un ángulo absoluto.
- ¿Qué bloque responde al evento “ser atacado por un robot”?. Advertir que busquen entre los bloques “a Codificar”. Se espera que respondan que el bloque “onHitByBullet” es el que permite al robot reaccionar cada vez que le disparan y que en dicho bloque deben programar la estrategia de defensa del robot.

Dar un tiempo para que todos los grupos desarrollen su solución y luego hacer una puesta en común. Una posible solución de un robot que se defiende cuando le disparan es la que se muestra en la Figura 14: el robot gira el arma hacia el ángulo de dónde proviene el ataque y dispara.

Figura 14 - Estrategia Defensiva: el robot se defiende cuando le disparan

Proponer a los estudiantes que ejecuten su robot y comprueben su estrategia. Para ello, recomendar que elijan como contrincante al robot “Tracker” o a “RamFire”, ambos están incluidos en RITA y sus estrategias son ofensivas. Advertir que no es necesario posicionar los robots en un lugar particular del campo de batalla.
Una solución de “El robot enfrenta a quien le disparó” está disponible en: http://linti.unlp.edu.ar/anexo6/Sec3act2res1

Cierre

A modo de cierre reflexionar junto con los estudiantes la siguiente característica del juego: si bien la estrategia consiste en que el robot reacciona apenas detecta que fue atacado por otro robot, disparándole, hay tiempos que se deben considerar; por ejemplo el tiempo de giro para posicionarse correctamente, el tiempo que lleva disparar y el tiempo de la trayectoria de la bala. Por lo tanto, no se puede garantizar que la acción resulte totalmente efectiva. Se pueden hacer sucesivas pruebas y ajustar estos tiempos.
Finalmente, explicar que así como el bloque “hitByBulletAngle” brinda información del ángulo del que el robot recibió el ataque, existen otros bloques de información que también resultarán útiles para programar estrategias de robots y que se irán trabajando a lo largo del anexo.

Ficha N°5

Ficha para el estudiante

El robot enfrenta a quien le disparó

Anexo N° 4: Proyectos educativos
Secuencia didáctica 3: Estrategias de robots en RITA

Bajada

En esta actividad tu robot tomará una actitud defensiva. Un robot enemigo acaba de dispararle y es una buena oportunidad para devolver el ataque. El robot debe responder el ataque en la misma dirección por donde fue recibido el impacto. ¿Te animás a posicionar tu robot de modo que se defienda cuando ocurra este tipo de ataque?

Actividad

Título

El robot enfrenta a otro, solo si el arma está lista

Objetivos

Que los estudiantes puedan:

- Programar una estrategia ofensiva.

Modalidad de trabajo

Actividad en grupos reducidos.

Materiales y recursos utilizados

Una computadora con el programa RITA instalado y la Ficha N° 6.

Bajada para el aula

Para comenzar la actividad, entregar a los estudiantes la Ficha N° 6 y organizarlos en grupos. Advertir que el objetivo de esta actividad es escribir una estrategia de robot ofensiva: cuando el robot detecta otro robot en el campo de batalla intentará dispararle, para ello verificará previamente que el arma esté lista. En este momento explicar que en RITA el arma del robot podría no disparar si fue usada exhaustivamente, lo que provoca que se caliente y se deba esperar a que se enfríe. La intención de los juegos en RITA es crear escenarios cercanos a la realidad y desalentar el uso desmedido de ataques.

Dar un tiempo a los estudiantes para que analicen en sus grupos el problema planteado y exploren en RITA los bloques que podrían usar para programar la estrategia.

Seguidamente preguntar: ¿cómo saben si el arma está lista para disparar?. Se espera que los estudiantes recuerden de la actividad previa que el bloque “Información” contiene información útil sobre el robot y su entorno y, que busquen allí información sobre el estado del arma. De ser necesario, se puede indicar que el bloque “gunReady” informa si el arma está lista o no.

En este momento, preguntar: ¿pensaron en cómo orientar el arma en la dirección del robot contrincante?. Probablemente no surja rápidamente la respuesta, en ese caso se puede ayudar haciendo una pregunta más simple: ¿cómo averiguan dónde está el robot detectado?. Ayudar a construir la idea que si se conoce dónde está el robot detectado, será posible orientar el arma en esa dirección y que además en RITA las ubicaciones están dadas por los valores de ángulos absolutos. Se espera que a partir de estas ayudas, los estudiantes busquen en el bloque “Información” un bloque que provea esa información. El bloque “scannedAngle” devuelve el valor del ángulo absoluto en el campo de batalla en el que se encuentra el robot detectado.

Finalmente, preguntar: ¿en qué bloque van a programar la estrategia del robot?. Aquí se puede retomar la idea trabajada en las actividades previas y advertir que esta estrategia es la reacción del robot al detectar otro robot en el campo de batalla. Se espera que los estudiantes busquen en el bloque “A Codificar” e identifiquen el bloque “onScannedRobot” como el evento de “robot detectado” o “robot escaneado”.

En este punto además, introducir una regla de RITA respecto al uso de radar: el radar se mueve cuando el arma ó el cuerpo del robot realiza algún movimiento. Este movimiento, que

es importante para poder detectar otros robots en el campo de batalla, consume parte de la energía del robot.
Dar un tiempo para que todos los grupos desarrollen su solución y luego hacer una puesta en común. Una posible solución de un robot que ataca cada vez que detecta un robot en el campo de batalla es la que se muestra en la Figura 15: si el arma está lista gira el arma hacia dónde está el robot detectado y le dispara.

Figura 15 - Estrategia ofensiva: El robot ataca cuando detecta un robot en el campo de batalla

Proponer a los estudiantes que ejecuten su robot y comprueben su estrategia. Para ello, recomendar que elijan como contrincante al robot SpinBot, incluido en RITA. SpinBot no tiene una estrategia en particular, es un robot movedizo, hace muchos giros en el campo de batalla y por eso es ideal para probar estrategias que buscan atacar. Advertir que no es necesario posicionar los robots en un lugar particular del campo de batalla.

Una solución de “El robot enfrenta a otro, solo si el arma está lista” está disponible en: http://linti.unlp.edu.ar/anexo6/Sec3act3res1

Cierre

A modo de cierre de la actividad, reflexionar junto con los estudiantes acerca de la estrategia ofensiva que elaboraron. Una estrategia ofensiva no se refiere a realizar disparos todo el tiempo, sino a proponer un conjunto de acciones que pueden resultar realizables, teniendo en cuenta las reglas del juego en RITA. A modo de ejemplo, que el arma esté lista es un condicionante para poder disparar, dado que una de las reglas del juego establece que el arma se calienta ante disparos sucesivos y no puede usarse hasta que se enfríe. Por ello fue necesario usar el bloque “gunReady”.

Ficha N°6

Ficha para el estudiante

El robot enfrenta a otro, sólo si el arma está lista

Anexo N° 4: Proyectos educativos
Secuencia didáctica 3: Estrategias de robots en RITA

Bajada

En esta actividad tu robot tomará una actitud ofensiva. Un robot ha sido detectado en el campo de batalla, y se espera que tu robot le dispare y trate de acertar. La única consideración especial es que no tratará de disparar en vano, solo lo hará si el arma está lista. ¿Te animás a escribir la estrategia ofensiva de tu robot?

Actividad

Título

El robot que dispara y huye

Objetivos

Que los estudiantes puedan:

- Ejercitarse en la programación de estrategias de robots complejas que hagan uso de los conceptos trabajados en las actividades previas.

Modalidad de trabajo

Actividad en grupos reducidos.

Materiales y recursos utilizados

Una computadora con el programa RITA instalado y la Ficha N° 7.

Bajada para el aula

Para comenzar la actividad, entregar a los estudiantes la Ficha N° 7 y organizarlos en grupos. Proponer a los estudiantes que imaginen el siguiente escenario: su robot avanza y choca contra otro robot en el campo de batalla. Esto representa una buena oportunidad para atacar al contrincante, sin embargo, es muy probable que el otro robot “piense” exactamente lo mismo: atacar. Para evitar este ataque o al menos disminuir las probabilidades, nuestro robot podría atacar y huir.

Debatir con la clase sobre el escenario planteado y seguidamente preguntar: ¿qué piensan que le podría ocurrir a nuestro robot si dispara e inmediatamente retrocede?. Probablemente los estudiantes respondan que nuestro robot será atacado de todos modos, porque su contrincante probablemente oriente su arma al ángulo de choque. Si nuestro robot no cambia de dirección, tiene altas probabilidades de seguir en un radio de alcance.

En este momento se puede reflexionar junto con los estudiantes sobre la siguiente idea: si nuestro robot huye cambiando de dirección, por ejemplo, en dirección perpendicular a la posición donde se encuentra el robot en el momento del choque, aumentan las posibilidades de esquivar el ataque del contrincante. Para trabajar sobre esta idea se puede hacer un gráfico esquemático en el pizarrón como el de la Figura 16.

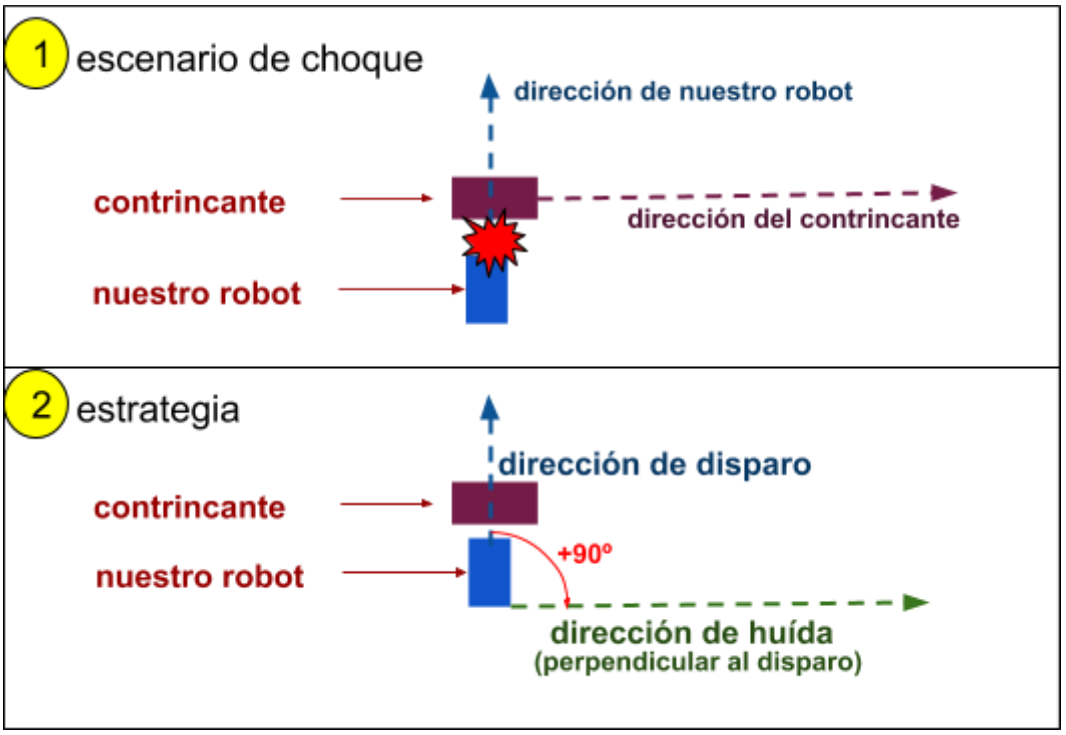


Figura 16 - Idea de la estrategia de disparo con huída perpendicular al disparo

Seguidamente, explicar a los estudiantes una característica del juego en RITA: cuando un robot choca contra otro, la información del ángulo en el que ocurre el choque se mantiene durante un período de tiempo corto, por lo tanto, esa información debe guardarse, si se va a necesitar.

Dar un tiempo para que cada grupo analice el problema planteado y luego hacer una puesta en común, con una serie de preguntas, para ayudar a construir las ideas de la estrategia del robot :

- ¿Cómo piensan guardar la información del ángulo donde ocurrió el choque?. Se espera que los estudiantes respondan que necesitan una variable. En este momento se puede explicar que las variables en RITA se definen usando el grupo “Nuevas Definiciones”. La forma de uso de las variables en RITA es la siguiente: para definir una variable, arrastrar el bloque “nombreDeVariable” y renombrar la variable. Luego para recuperar o modificar su valor, hacer *clic* con el botón derecho del mouse sobre la variable. En este ejercicio solo será necesario el bloque para recuperar el valor de la variable.
- ¿Qué bloque representa al evento “el robot choca con otro robot”?. Se espera que los estudiantes lo busquen entre los bloques “A Codificar” e identifiquen el bloque “onHitRobot”.
- ¿Cómo se puede obtener el ángulo en el que ocurrió el choque con el contrincante?. Se espera que los estudiantes recuerden, de las actividades previas, que la información sobre el robot y su entorno se encuentra en el grupo “Información” y que identifiquen el bloque “hitRobotAngle” como el que provee la información sobre el ángulo de choque.
- ¿Cómo se puede posicionar el robot en forma perpendicular a la orientación actual?. Es probable que la respuesta no surja inmediatamente, ayudar a los estudiantes realizando un gráfico en el pizarrón similar al de la Figura 13. Mostrar que si se resta o se suman

90º al ángulo de choque, este nuevo valor tendrá una orientación perpendicular al anterior.

Dar un tiempo para que todos los grupos desarrollen su solución, podrían ser aproximadamente 30 minutos y luego hacer una puesta en común. Una posible solución de un robot que huye cuando choca contra otro robot es la que se muestra en la Figura 17: cambia la orientación del robot girando perpendicularmente, gira el arma hacia el ángulo de choque, dispara y luego huye.

Figura 17 - Estrategia de disparo con huída perpendicular al disparo

Proponer a los estudiantes que ejecuten su robot y comprueben su estrategia. Para ello, recomendar que lo realicen en dos etapas: en la primera que elijan a SpinBot para verificar que la estrategia funciona de acuerdo a lo planificado y en la segunda que elijan a un robot de ataque como Tracker o RamFire para comprobar la efectividad de la estrategia. Advertir que no es necesario posicionar los robots en un lugar particular del campo de batalla.

Una solución de “El robot que dispara y huye” está disponible en:
http://linti.unlp.edu.ar/anexo6/Sec3act4res2

Cierre

A modo de cierre reflexionar junto con los estudiantes sobre las siguiente variaciones de la solución:

- Solución 1: girar el robot para posicionarse perpendicularmente, girar el arma al ángulo de choque, disparar y luego avanzar.
- Solución 2: girar el arma al ángulo de choque, disparar, posicionarse perpendicularmente y avanzar.

En la Solución 1, el robot antes de disparar está listo para huir, sin embargo, demora en disparar porque se posicionó, por tanto el contrincante tiene más tiempo para huir.
En la Solución 2, si bien el robot dispara inmediatamente, lo que le da más probabilidades de acertar al contrincante, demora más para posicionarse y huir, aumentando las posibilidades de ser atacado.
Las dos soluciones son correctas, cada una tiene sus pros y contras.

Ficha N°7

Ficha para el estudiante

El robot que dispara y huye

Anexo N° 4: Proyectos educativos
Secuencia didáctica 3: Estrategias de robots en RITA

Bajada

En esta actividad tu robot tomará una actitud ofensiva. En caso de choque contra otro robot, aprovechará el incidente para posicionarse apuntando al ángulo donde ocurrió el choque y disparará. Recordá que es muy probable que el otro robot tenga una estrategia preparada ante un disparo enemigo, por lo tanto, tu robot debe estar preparado para huir rápidamente una vez realizado el disparo ¿Ayudás a tu robot a construir esta estrategia?.
El siguiente gráfico puede ayudarte a analizar el escenario de choque y de huída: