Sugerencias para el dictado del curso "La programación y su didáctica. Método Program.AR"

Complemento al cuaderno "Actividades para aprender a Program.AR" de la Fundación Sadosky

Dr. Pablo E. Martínez López
Universidad Nacional de Quilmes

Versión del 13 de julio de 2016

Resumen

En este documento se ofrecen algunas recomendaciones para una manera de enmarcar y valorizar el curso "La programación y su didáctica. Método Program.AR", propuesto por la Fundación Sadosky para contribuir en la difusión de la programación y la computación en general como conocimiento de cultura general a ser impartido en todos los niveles educativos de formación básica. El material presentado aquí sirve como guía de referencia para el armado de las clases, para entender cómo cada tema aporta al conjunto y aprovechar tal cosa en el dictado del curso, así como para ofrecer opiniones y consejos respecto de las soluciones de actividades individuales con el objetivo de observar críticamente cada una de ellas y sus soluciones respecto del marco mencionado.

Este material está dirigido a los docentes que consideren dictar el curso, ya sea que lo vayan a hacer directamente a estudiantes (de primaria o secundaria) o lo hagan a otros docentes en el marco de cursos de formación docente. Es importante que cada docente adapte las sugerencias aquí ofrecidas al nivel de sus estudiantes: no es lo mismo un curso brindado a otros docentes con la idea de que en el futuro dicten este curso, que uno brindado a estudiantes de primaria.

Para leer este documento se recomienda leer completa la parte I, que contiene el contexto y el marco conceptual recomendado, dejando la parte II para consultar a medida que se va dictando el curso, y cuando se vaya a preparar la clase correspondiente a cada una de las secuencias didácticas, ya que cada una de sus secciones contiene consejos y recomendaciones para el dictado de cada una de las secuencias didácticas propuestas en el cuaderno.

Índice

Parte I
Presentación y marco conceptual

Esta primera parte presenta el propósito del documento, el contexto, y la contribución principal: un **marco conceptual** para el curso. Se recomienda leer esta parte en su totalidad antes de planificar el dictado del curso.

1 Introducción

En este documento se ofrecen algunas recomendaciones para una manera de enmarcar y valorizar el curso "La programación y su didáctica. Método Program.AR". Dicho curso se basa en contenidos y ejercitación que se encuentran explicitados en el cuaderno para el docente titulado "Actividades para aprender a Program.AR" [FSO15], que está destinado al segundo ciclo de la educación primaria y primero de la secundaria. Como resultado de haber utilizado dicho cuaderno, surge la observación de que el mismo es muy completo en contenidos pero puede verse enormemente beneficiado si los mismos se encuadran en un marco conceptual explícito, y si cada actividad es observada críticamente respecto de dicho marco. El material presentado aquí sirve como guía de referencia para el armado de las clases, para entender cómo cada tema aporta al conjunto y aprovechar tal cosa en el dictado del curso, y para ofrecer opiniones y consejos respecto de las soluciones de actividades individuales con el objetivo de observar críticamente cada actividad y sus soluciones respecto del marco mencionado. Si bien no es la única forma posible de enmarcar el curso, ha sido utilizado en varias ocasiones con excelentes resultados y críticas positivas por parte de los docentes y formadores que tomaron los cursos de formación ya dictados.

En primer lugar, en la sección 2, se hace un repaso rápido de los temas tratados por el cuaderno, con el objetivo de servir de base para las discusiones posteriores y la ubicación de las actividades en este temario.

En la sección 3 se presenta el marco conceptual sugerido para encuadrar el curso, y se discute una manera de ir presentando las actividades y los temas dentro de dicho marco. Además, se sugiere que en el caso de que el curso sea dictado por formadores de formadores (o sea, a docentes que luego apliquen el curso en estudiantes de primaria y secundaria), este marco conceptual sea presentado de manera explícita para servir de referencia a los docentes.

Una vez encuadrado el curso en el marco conceptual sugerido, puede verse que las actividades propuestas, que son muy completas y persiguen el objetivo planteado, no son consecuentes en todo momento con los consejos basados de dicho marco. Por esta razón, en la sección 4 se analizan ciertos detalles que contribuyen a la mejora de las mismas y se presentan consejos para analizar críticamente las soluciones propuestas en el cuaderno a muchos de las actividades, mostrando cómo con pequeñas correcciones que no alteran el espíritu ni propósito de la actividad original se puede conseguir mejorar aún más el valor agregado del curso desde el punto de vista de la comprensión de conceptos, y en el caso de los docentes que toman el curso en calidad de futuros formadores, pueden tener más argumentos y criterios para enfrentar situaciones de aula donde los estudiantes planteen soluciones diferentes.

Finalmente, en la sección 5 se ofrecen algunas conclusiones y se completa esta guía.

2 Análisis de "Actividades para aprender a Program.AR"

En esta sección analizaremos el cuaderno para el docente "Actividades para aprender a Program.AR" escrito Pablo Factorovich y Federico Sawady O'Connor y editado por la Fundación Sadosky [FSO15], que describe los contenidos y ejercitación del curso que nos ocupa.

El cuaderno posee 5 capítulos (además de una introducción), cubriendo cada uno un tema importante en un curso de enseñanza de programación que transmita los principios básicos de la programación de manera amena y práctica, basándose en un enfoque de aprendizaje por indagación. Los temas que se cubren en cada capítulo son:

- Comandos, procedimientos y repetición.
- Alternativa condicional.
- Repetición condicional.
- Parametrización.
- Interactividad.

En la introducción se discute el enfoque didáctico, basado totalmente en ejercitación, y se presentan los propósitos generales del cuaderno así como un desglose de los contenidos en detalle. El enfoque propuesto utiliza una técnica conocida con el nombre de aprendizaje por indagación, la cual se amplía aquí en la primera parte de la sección 3 (subsección 3.1), ya que integra el marco conceptual propuesto en este documento. Lo más importante a entender por el momento es que cada uno de los capítulos del cuaderno se presenta como una serie de actividades que permiten reflexionar sobre un determinado concepto a ser presentado, al tiempo que van sumando de a poco los conceptos presentados en ejercicios y capítulos anteriores. Una lectura rápida del cuaderno puede llevar a la consecuencia de no reconocer o soslayar el hilo conductor, por lo cual el presente documento explicita un posible marco para ello.

Corresponde hacer una **aclaración importante**: el cuaderno utiliza como herramienta para la ejercitación el entorno Scratch 2 (la versión 1 de Scratch no tiene todas las herramientas necesarias para desarrollar este curso). Sin embargo, luego de la publicación del cuaderno, la Fundación Sadosky continuó desarrollando material para el curso, y actualmente la ejercitación que aparece en este cuaderno se re-implementó utilizando el entorno Pilas Bloques, la cual dispone de una versión *online* (`http://pilasbloques.program.ar`) y también puede ser descargada para su utilización *offline* sobre diversos sistemas operativos (Huayra, Windows, Mac OS X). Además de las cuestiones de utilización del entorno Scratch, todos los contenidos propuestos en los ejercicios con excepción de los correspondientes al tema de interactividad, siguen siendo válidos en el nuevo entorno. Además, y dado que Pilas Bloques se encuentra todavía en desarrollo, es posible que aparezcan nuevos temas para completar la formación. Sin embargo, Pilas Bloques resulta complementario a Scratch, pues en este último entorno se pueden realizar muchas más cosas, y fundamentalmente permite que los estudiantes exploren sus propios ejemplos y programas. Pilas Bloques se puede utilizar para comenzar, y luego se puede agregar Scratch de considerarlo necesario. En esta guía nos basaremos fundamentalmente en el cuaderno, pero también comentaremos las diferencias y novedades que aparezcan en los ejercicios en Pilas Bloques.

En el primero de los capítulos se comienza describiendo la noción básica de programa, comandos y ejecución, se presenta la noción de procedimiento simple y luego la de repetición simple (junto con la forma más básica de expresiones: los números). Otra de las funciones de este capítulo es presentar el entorno que se utilizará para la ejercitación durante todo el libro: Scratch[1]. También se presentan, solamente para su uso dentro de este capítulo, otras dos entornos. Uno de ellos, Lightbot (`http://armorgames.com/play/2205/light-bot`), se utiliza por su efecto motivador, porque presenta de una manera interesante la necesidad de contar con procedimientos y finalmente porque es uno de los entornos alternativos de trabajo que se mostrarán durante el curso, favoreciendo la transferencia de conceptos. El segundo es una metodología para escribir y ejecutar programas sin utilizar computadoras: programando en papel

[1] Observar que, como fue mencionado, actualmente se puede utilizar otro entorno más moderno y actualizado que contiene los mismos ejercicios: Pilas Bloques.

cuadriculado; el valor de esta herramienta reside en que los estudiantes pueden entender que los programas no tienen por que ser únicamente textuales y que no solo las computadoras pueden ejecutarlos, y en simultáneo, entender las dificultades involucradas con tratar de entender un programa solo por su ejecución, sin apelar a herramientas conceptuales de mayor nivel. Al trabajar en tres entornos tan diferentes, los conceptos a transmitir se visualizan con mucha mayor claridad, y se favorece enormemente su transferencia posterior a otros entornos.

El segundo capítulo se dedica enteramente a presentar y trabajar la noción de alternativa condicional. Primero se presenta la necesidad de tener dicha herramienta (¿cómo resolver el problema de que un único programa debe funcionar en varios escenarios diferentes?), y luego se presenta la alternativa condicional (representada por la herramienta "si-entonces-sino" o su versión simplificada "si-entonces") junto con las expresiones necesarias para poder utilizarla (los sensores, p.ej. "¿Tocando?" o "¿Hay?", que permiten obtener información sobre el estado del escenario). Se plantean ejercicios de complejidad creciente, donde los últimos combinan las ideas con las presentadas en el capítulo anterior.

En el tercer capítulo se trabaja con la noción de repetición condicional. Nuevamente, la motivación para esta nueva herramienta está basada en los escenarios cambiantes, salvo que lo que cambia ahora es la cantidad de elementos a procesar (que no es conocida de antemano ni puede consultarse u obtenerse de manera simple). Las condiciones utilizadas son simples, y siguen los mismos lineamientos que las utilizadas con la alternativa condicional. Los ejercicios más avanzados, nuevamente, plantean combinaciones de la repetición condicional con las herramientas vistas en los capítulos anteriores.

El capítulo cuatro presenta la idea de parámetros y parametrización. Para empezar, lo hace a través de una analogía entre procedimientos y estribillos en canciones, y estudia lo que sucede cuando el estribillo tiene variaciones (como la canción de "Un elefante se balanceaba"'). Luego presenta la manera de definir y utilizar procedimientos parametrizados en Scratch y plantea ejercicios simples que utilizan parametrización. Además, se presenta también la idea de operaciones sobre expresiones. La operación más simple es realizar cuentas con números, lo que tiene sentido cuando uno de los números es un parámetro, ya que la cuenta no puede ser reemplazada por el resultado, al no saber de antemano qué valor adoptará el parámetro. La otra operación presentada es la de igualdad entre cadenas de caracteres, para poder determinar en qué dirección debe moverse. La ejercitación, en este caso, no profundiza en combinaciones más complejas, puesto que la noción de parámetro es complicada y no es objetivo principal del curso, más allá de entender que existe y cómo funciona.

El último capítulo, el quinto, presenta la idea de interactividad. Esta idea tiene que ver con expresar en el programa la interacción con un usuario, con el objetivo de realizar pequeños juegos. Si bien el tema de interacción hombre-máquina es muy amplio y complejo, aquí se presenta solo en su forma más simple (sensores que determinan si se está tocando cierta tecla), ya que los juegos son muy motivadores para los estudiantes, pero no es necesario presentar un modelo complejo para esto. Junto con la idea de interactividad se presentan también operaciones sobre los valores de verdad, para poder verificar condiciones complejas en los juegos. También se trata el tema de variar el aspecto visual de los escenarios (de manera independiente al programa), lo que permite tener variaciones de un juego con diferentes temáticas. Los juegos propuestos en la ejercitación son simples, pero abren la puerta a combinaciones más complejas que los estudiantes deberán explorar por sí mismos. De haber utilizado Pilas Bloques en lugar de Scratch, y puesto que en Pilas Bloques no hay actividades de interactividad, en este momento del curso debe presentarse este entorno.

El curso puede continuar con algunos temas más que no están incluidos en el cuaderno. Uno de ellos es la noción de memoria y variables, junto con el comando de asignación. Para este tema podrá encontrarse material de Scratch en `http://program.ar/material-curso-docentes`. El otro es la utilización de todos los conceptos trabajados en un entorno diferente de Scratch o Pilas Bloques. Para ello existen algunas actividades que se pueden realizar utilizando el software Alice. Las actividades propuestas se pueden encontrar en el mismo sitio mencionado

antes. Si se utiliza el entorno Alice, puede también presentarse la noción de función, que es el equivalente a los procedimientos, pero en la categoría de las expresiones. Si se utiliza el entorno Pilas Bloques, los temas de interactividad, variables y funciones no se encuentran entre las actividades propuestas, por lo que debe utilizarse Scratch y/o Alice.

3 Marco conceptual

Para poder presentar un marco conceptual para este curso, lo primero que debemos entender, y remarcar durante el dictado del mismo, es que el objetivo del curso propuesto por el cuaderno no es formar programadores profesionales, sino ***transmitir conceptos básicos y generales sobre programación*** a personas de toda índole, sin importar su profesión. En ese sentido es que esta propuesta de Program.AR se alinea con cursos de matemáticas o lengua de primaria o secundaria: se trata de formación básica que todas las personas deben tener hoy en día para enfrentarse con el mundo moderno. De esta manera, los conceptos que se tratan deben ser suficientemente generales como para abarcar este objetivo, y deben destacarse de manera adecuada para que no se pierda el foco del curso en detalles irrelevantes.

El marco conceptual que se presenta se basa en la experiencia del autor en la definición de didácticas innovadoras [ML13] y en el dictado del curso en cuestión en cuatro ocasiones diferentes, con sucesivos refinamientos de la presentación. Ha recibido críticas positivas de muchísimos de los estudiantes que participaron de las instancias del curso. Si bien no es la única posibilidad para enmarcar el curso, la experiencia muestra que es una forma de ayudar a los docentes que participan como estudiantes del curso para que puedan comprender de manera más acabada el rol de cada actividad en el total, y el énfasis que darle a cada actividad, al mismo tiempo que los provee de herramientas conceptuales que les permiten realizar observaciones críticas sobre las actividades y sus soluciones, ampliando el espectro de las mismas y las ideas con las que pueden responder a otras soluciones por parte de sus propios estudiantes.

Este marco conceptual se organiza en dos ejes, el segundo de los cuales se organiza a su vez en dos sub-ejes y cada uno de ellos en varios temas. Primero se presenta el marco conceptual completo, y luego se discute y comenta cada eje por separado. Los ítems destacados en negrita son los considerados más importantes, y a los que debe prestarse especial atención a lo largo de todo el curso.

- Didáctica propuesta
  - **Aprendizaje por indagación**
- Conceptos fundamentales
  - Herramientas conceptuales
    - Estrategia de solución y **división en subtareas**
    - Legibilidad y **elección de nombres adecuados**
    - Algorítmica básica: recorridos
  - Herramientas del lenguaje
    - Acciones (comandos – "verbos")
      - Comandos primitivos; secuencia de comandos
      - **Procedimientos** y parámetros
      - Repetición simple y repetición condicional
      - Alternativa condicional
      - Asignación de variables
    - Datos (expresiones – "sustantivos")

- Valores (literales numéricos y otros)
- Sensores y datos primitivos; sensores de interactividad
- Operadores
- Parámetros y Variables
- **Funciones**

Para cada uno de estos ejes vale la pena considerar su características, su aporte al conjunto y cómo entre ellos enmarcan y valorizan el material original. Este aporte, y la división de los subejes en temas, se trata en las subsecciones correspondientes.

3.1 Didáctica propuesta

El aprendizaje por indagación [Dos15] es una metodología de enseñanza-aprendizaje a través de la cual los estudiantes deben encontrar soluciones a un problema a partir de un proceso de investigación, usualmente poniendo énfasis en el trabajo cooperativo y en la extracción de ideas a través de la reflexión sobre las actividades realizadas para construir la solución. El enfoque por indagación facilita la participación activa de los estudiantes en la adquisición del conocimiento y ayuda a desarrollar el pensamiento crítico y la capacidad para resolver problemas [SGW11, Cha11]. La versión propuesta por el cuaderno se conoce como *indagación estructurada*, en cuanto a que el docente provee la pregunta inicial y delinea el procedimiento general de solución, siendo responsabilidad de los estudiantes encontrar la solución y formular explicaciones basados en su trabajo.

Una de las ideas fundacionales del enfoque de Program.AR es justamente poner foco en el aprendizaje por indagación. Todos los conceptos presentados deben ser motivados mediante un ejercicio que precisa el nuevo concepto, y luego de que los estudiantes intentaron solucionarlo con los elementos que disponen hasta ese momento, y a través de la reflexión, se propone el nuevo concepto, y se les permite volver a intentarlo. Esta técnica, si es utilizada correctamente, permite al estudiante adquirir el concepto con mayor firmeza, pues experimentó cómo solucionar el problema *sin* ese concepto, y por lo tanto el mismo le resulta necesario. Las experiencias realizadas muestran que los estudiantes valoran muchísimo el concepto una vez presentado y entienden claramente cómo y por qué deben utilizarlo; luego de un tiempo, ante un nuevo problema que les resulta complejo resolver preguntan "¿Cuál es el concepto que me falta?", lo que evidencia claramente que comprenden la metodología y valoran los conceptos nuevos. Es por eso que resulta tan importante esta forma de aprendizaje, y es una de las claves que diferencian el enfoque elegido por la didáctica Program.AR.

El curso posee una serie de conceptos técnicos que se busca que los estudiantes aprendan, y es usual que los docentes especializados en informática le den más importancia a estos conceptos que a la didáctica utilizada. Esto es un grave error que debe evitarse, ya que no solo importan los conceptos, sino la forma en que los mismos son aprendidos y aprehendidos por los estudiantes. O sea, estos conceptos técnicos pueden aprenderse de varias maneras, pero no todas garantizan que se pongan en juego los recursos y estructuras de pensamiento que este enfoque busca que ejerciten y cuya aprehensión aporta beneficios más allá de la programación (por ejemplo, capacidad de abstracción, estrategias de resolución de problemas, etc.). Por eso es importante remarcar con gran énfasis que en la propuesta Program.AR los conceptos técnicos están al **mismo nivel de importancia** que la didáctica. Esta guía va dirigida tanto a docentes que lo dictarán a otros, como a docentes que lo dictarán a estudiantes primarios y secundarios. Todos deben tener en cuenta equiparar la importancia de la metodología didáctica con los conceptos técnicos, pero lo deben hacer con mayor énfasis los capacitadores que dictan a otros docentes que luego a su vez lo replicarán, pues si no se enfatiza suficiente, la idea puede diluirse en futuros dictados. Por otra parte, no alcanza con decirlo de manera teórica: todo el tiempo debe hacerse uso del aprendizaje por indagación para que se valore de manera correcta. En el caso de un curso a docentes, cada utilización de la didáctica debe reforzarse con una explicitación

de su utilización, y también proponiéndoles que experimenten ellos mismos la didáctica por indagación, ya que al experimentarla pueden llegar a valorarla mucho más que si simplemente la estudian como objeto o teorizan sobre ella. Después de todo, es sabido que uno enseña de la misma forma en que aprendió. Dicho de otra forma: el curso que se dicte a formadores debe llevarse adelante considerando que los docentes son estudiantes, y proponiéndoles la indagación exactamente de la mismas forma; recién terminado el curso se les ofrecería el material didáctico completo con todas las soluciones y esta guía.

Un detalle que resultó de la experiencia de dictar efectivamente los cursos es que esta noción de que la didáctica es igualmente importante que los conceptos técnicos debe reiterarse en cada clase, y casi, debe decirse, en cada ejercicio. Esto es así porque en programación, por costumbres propias de la disciplina, es muy fácil concentrarse en los aspectos técnicos, y dejar de lado la metodología elegida. Después de todo, ¿qué importa la didáctica? Si los estudiantes consiguen resolver los ejercicios, ¿importa de qué manera lo hicieron? Como vimos, *sí* es importante, y en más de un nivel. Este consejo que se discute aquí es nada más que una instancia de la necesaria reiteración de toda idea que realmente se busque transmitir en un curso, y con la que todo docente debe estar familiarizado, con lo cual solamente debe remarcarse que este es uno de los conceptos claves a desarrollar.

3.2 Herramientas conceptuales

Además del eje de didáctica, el marco conceptual incluye dos grupos de conceptos técnicos fundamentales, que agrupamos en dos grupos de herramientas: las herramientas conceptuales, discutidas en esta sección, y las herramientas del lenguaje, discutidas en la sección siguiente.

Las herramientas conceptuales conforman, junto con la didáctica propuesta, el corazón de lo que el curso busca transmitir. Si alguien debiese resumir el curso en un par de frases, estas frases no deberían dejar de incluir estas herramientas conceptuales[2]. O sea, todo el curso fue vertebrado alrededor de estos conceptos, y por ello es muy importante entenderlos, explicitarlos y remarcarlos continuamente a lo largo de su dictado. Si se respetan estos conceptos (junto con el enfoque didáctico mencionado) se podría afirmar que el curso no sufrirá variaciones significativas. Estas herramientas se basan en una concepción moderna de la programación, que se discute en esta sección antes de entrar en detalle con cada una de las herramientas.

Las herramientas conceptuales consideradas son tres:

1. la noción de *estrategia de solución*, y la de su explicitación para aplicar la metodología de *división en subtareas*,

2. la noción de que los programas son fundamentalmente un medio de comunicación entre personas, además de servir como vehículo para hacer funcionar máquinas, capturado en la importancia de que los programas sean *legibles* (o sea, claramente entendibles por otros programadores) y esto a su vez expresado mediante la metodología de *elección de nombres adecuados* para cada una de las partes de un programa que se escribe, y

3. la noción de *algorítmica básica*, expresada en este curso en forma simplificada, por ejemplo en la noción de *recorrido*.

Cada uno de estos conceptos será desarrollado en las secciones que siguen, luego de algunas consideraciones generales más.

[2]Por ejemplo, "Este curso propone enseñar programación poniendo el foco en la didáctica por indagación y en centrar el proceso de programar en la explicitación de la estrategia de solución, dividiendo adecuadamente una tarea en subtareas, y capturando las mismas mediante herramientas del lenguaje como los procedimientos, resaltando el hecho de que un programa debe comunicar de manera legible su propósito a quién lo lea. Es de destacar que en la propuesta, la didáctica debe tener una relevancia similar a los conceptos de estrategia de solución y legibilidad, y la forma de expresarlos en el lenguaje de programación."

Las tres herramientas mencionadas son de fundamental importancia, y por esa razón, es conveniente que sean explicitadas y consideradas al solucionar cada ejercicio. Es usual que los estudiantes (y a veces también los docentes) se concentren más en completar el ejercicio o conseguir una solución operativa, que en la calidad de esa solución. Para que este curso sea realmente valioso, es importante **remarcar** a cada paso **que la calidad de las soluciones importa**, y esta calidad debe medirse en base a las herramientas aquí presentadas. Por otra parte, a la hora de determinar si una solución ofrecida por un estudiante es correcta y adecuada, es más valioso que la misma tenga en cuenta estos conceptos aunque no funcione de manera completa, a que funcione bien pero haya descuidado la aplicación de estas herramientas.

La correcta utilización de estas herramientas tiene un impacto directo en la calidad de los programas que se escriben. Esto es de fundamental importancia cuando se forman programadores profesionales, pues código de calidad significa mayor productividad. Sin embargo, esta forma de pensar y organizarse tiene valor más allá de las aplicaciones profesionales de la programación, y es por eso una de las razones por las que desde la iniciativa Program.AR, que provee el contexto de este curso, se busca generalizar la enseñanza de las Ciencias de la Computación. Las estrategias de solución, la correcta comunicación de soluciones a problemas, y mecanismos elementales de solución de problemas sencillos habilitan y fomentan la formación de capacidades de *pensamiento de alto orden* [AK01], que hoy día es aceptado como forma fundamental del pensamiento humano. Para mencionar algunos ejemplos de la vida cotidiana: cuando alguien prepara su bolso a la mañana antes de ir al colegio o al trabajo, está realizando una actividad vinculada al pensamiento de alto orden; cuando pierde algo y retrocede por los lugares que estuvo para ver en cuál lo dejó, también; cuando decide hacer un viaje y anota en su lista que debe comprar un pasaje, reservar un hotel, preparar una valija, conseguir una forma de llegar a la estación, etc., también (y en este caso, está aplicando la noción de división en subtareas, y por lo tanto considerando una estrategia de solución). Todas estas formas de pensamiento precisan del aprendizaje de habilidades que requieren tomar decisiones, resolver problemas, y aplicar pensamiento crítico, que es, justamente, el fundamento del pensamiento de alto orden. Para resumir, no da lo mismo una solución que funciona pero resuelve de cualquier manera el problema, comparada con una más elaborada que aprovecha las características del problema; y en ocasiones, la única manera de lograr una buena solución es de forma elaborada: las soluciones simples muchas veces no alcanzan para resolver ciertos problemas.

3.2.1 Estrategia de solución y división en subtareas

Toda vez que se busca solucionar algún problema, es necesario contar primero con alguna idea de cómo encarar dicha solución, o sea, qué elementos disponer para la solución y de qué manera. Esto en programación se conoce como *estrategia de solución*: ¿qué cosas considerar a la hora de realizar la solución? ¿Cuáles son los componentes que interactuarán en la solución para obtener la respuesta deseada, y de qué manera lo harán? Y de hecho cada vez que una persona intenta solucionar un problema, lo hace siguiendo cierta estrategia. Lo que no es usual es que la estrategia se considere de manera *explícita*, y se valore como una parte fundamental de la solución.

La manera más importante con la que se pueden expresar estrategias de solución es a través de considerar pequeños problemas cuyas soluciones combinadas provean la solución al problema general. Esta forma de descomponer un problema en sub-problemas (o una tarea a realizar en sub-tareas) es una de las bases conceptuales de la programación. Es un concepto tan importante, además una herramienta conceptual invalorable, que la nombramos directamente como **división en subtareas**. Si el curso debiese restringirse a un único concepto, la división en subtareas sería el concepto elegido, pues representan a la forma de pensar composicionalmente y es una herramienta invalorable en el pensamiento de alto orden. Vale la pena remarcar que mientras que la *estrategia de solución* es una idea particular de cómo resolver un problema, la *división en subtareas* es una forma específica de cómo expresar dicha estrategia a través de

soluciones a problemas más pequeños.

Todos los ejercicios en este curso se basan en la adecuada división de la tarea a realizar en subtareas apropiadas, y posteriormente la utilización de herramientas del lenguaje (ver la siguiente sección) para explicitar estas subtareas y sus combinaciones. En algunos ejemplos, especialmente los más sencillos del principio, la estrategia es muy simple y puede resultar discutible la conveniencia de explicitarla mediante subtareas. Sin embargo, es básico para el objetivo de este curso que en cada uno de los ejercicios la estrategia de solución elegida sea explicitada y luego expresada como parte del programa a través de subtareas. Esto se logra combinando las herramientas del lenguaje con las demás herramientas conceptuales de manera adecuada; pero el concepto fundamental subyacente en todo momento es el de la explicitación de la estrategia de solución a través de la adecuada división en subtareas (además de la habilidad de nombrar adecuadamente cada una y de utilizar las herramientas del lenguaje adecuadas para construir la solución).

La recomendación de este texto es que el docente fomente la explicitación de estrategias de solución y su expresión mediante subtareas, las cuales son consideradas como parte fundamental de la tarea de resolver los problemas mediante programas en cada uno de los ejercicios. Sin embargo, es importante que el docente no se restrinja a utilizar una única estrategia de solución ya probada, sino que esté abierto a otras propuestas por parte de los estudiantes, y pueda fomentar a que entre todos se analice la conveniencia o no de utilizar una u otra, o la equivalencia de diferentes variantes. Además, la misma estrategia de solución puede expresarse mediante diferentes divisiones en subtareas, por lo que este consejo de no restringir la creatividad de los estudiantes se hace extensivo también a la forma en dividir en subtareas a la hora de expresar una estrategia de solución.

Algunos de los ejercicios están pensados específicamente para mostrar la importancia de considerar estrategias de solución, especialmente el que utiliza el entorno Lightbot. Pero, no es suficiente reiteración decirlo de nuevo, es de central importancia que en cada ejercicio se busque que los estudiantes expliciten sus estrategias y las expresen adecuadamente.

La herramienta del lenguaje básica para expresar subtareas y estrategias es el **procedimiento**, y consecuentemente, es una de las primeras herramientas que se presentan y trabajan en los ejercicios. En su versión inicial, sin embargo, el cuaderno no hace uso de los procedimientos de la manera más completa posible. El autor de este texto tiene como costumbre que cada ejercicio se resuelva mediante un único procedimiento para el que se elige un nombre adecuado que exprese la tarea (y de ser posible, la estrategia); esto tiene como ventaja el hecho de que el foco inicial está puesto en identificar y nombrar adecuadamente el problema a resolver, y facilita el arrancar con la estrategia de solución, pues no se comienzan considerando detalles del problema, sino el problema como un todo. El cuaderno no sigue esta práctica metodológica, pero es recomendable que el docente considere la incorporación de esta práctica, debido a las ventajas recién descritas. Dicho esto, debe hacerse la salvedad de que esto no debe realizarse hasta que la idea de procedimiento haya sido presentada, por lo que los primeros ejercicios estarían exentos de esta recomendación.

Resulta útil considerar algunos consejos de cómo fomentar la utilización de estas ideas. Lo primero que debe hacerse al intentar solucionar un problema mediante un programa es idear y enunciar una estrategia de solución. Luego debe pensarse una división en subtareas que exprese la estrategia deseada. El siguiente paso para escribir el programa es el que recién se discutió: incluso antes de investigar cuáles son las primitivas, debe declararse y nombrar adecuadamente un procedimiento que exprese la solución al problema completo, y a continuación, declarar procedimientos para cada una de las subtareas ideadas, nombrándolos adecuadamente, y combinar estos procedimientos en el procedimiento principal para expresar la solución general. No debe utilizarse ni un solo comando primitivo antes de poder leer la estrategia de solución en el procedimiento principal y convencerse de que es adecuada; recién entonces es momento de definir cada uno de los procedimientos que expresan a las subtareas.

Otro consejo útil para fomentar la división en subtareas consiste en obligar a que cada

procedimiento pueda llevar a lo sumo una única estructura de control (ya sea una repetición, una alternativa, etc.); al no poder realizar dos repeticiones o una repetición que involucre una alternativa en el mismo procedimiento, se hace necesario definir primero un procedimiento que exprese una de las estructuras de control, y luego otro que, utilizando el definido primero, exprese la segunda.

Resumiendo, los **pasos recomendados** para solucionar un problema mediante un programa son:

1. **Idear una estrategia de solución, y explicitarla.**
2. **Expresar la estrategia mediante alguna división en subtareas.**
3. **Declarar y nombrar adecuadamente un procedimiento principal que exprese la estrategia ideada.**
4. **Declarar y nombrar adecuadamente procedimientos que expresen las subtareas.**
5. **Utilizar estos procedimientos para definir el procedimiento principal**, y validar que la estrategia de solución resulte clara al leerlo. Seguir para esto el principio de **utilizar como máximo una estructura de control por procedimiento** (para fomentar aún más la subsiguiente división en subtareas).
6. **Definir cada uno de los procedimientos que expresan subtareas**, repitiendo para cada sub-problema particular los pasos antedichos para definir la solución a un problema.

Ejemplos del uso de estos consejos pueden encontrarse en la parte II, y con más detalle en, por ejemplo, la sección 4.6, entre otras.

3.2.2 Legibilidad

Para comprender bien el valor de la siguiente herramienta conceptual a considerar, debe primero analizarse brevemente la concepción de la programación en la que esta propuesta se inscribe. La idea fundamental de esta concepción es que un programa es una ***descripción ejecutable*** de la solución a un problema computacional[3]. Al definir de esta forma la noción de programa se realza un hecho central, usualmente poco considerado, que es que los programas son *entidades duales*: ¡al mismo tiempo deben indicarle a una máquina cómo funcionar, pero también deben comunicar la solución propuesta a las personas! O sea, los programas son manipulados por **dos** entidades: máquinas, pero también personas. Las definiciones y propuestas tradicionales ponen todo el énfasis en la máquina que ejecuta el programa y en su funcionamiento, y dejan en segundo plano el valor comunicacional del programa, invisibilizando de alguna manera a las personas que los programan. En este curso ese énfasis se invierte: es **muchísimo más importante el valor comunicacional de un programa** que su capacidad de hacer funcionar una computadora; es **mucho más importante el proceso mental desarrollado por la persona que programa para pensar y escribir su programa** que el resultado final. Obviamente, el funcionamiento del programa **es** importante y no debe descuidarse, pero ello **no debe debilitar** la consideración del valor comunicacional de un programa para otras personas.

Para escribir un programa, el único medio posible es utilizar un lenguaje de programación. Entonces, el programa constituye el vehículo ideal de explicitación y comunicación de las ideas con las que se concibió la solución expresada. Es por ello que el **programa** tiene la doble función de expresar cómo debe funcionar la máquina para obtener la solución, pero **también** expresar

[3] Observar que esta forma de concebir y definir la noción de programa abarca, pero excede, a la definición tradicional de programa como "secuencia de instrucciones". En la visión tradicional, el único rol importante es el de hacer funcionar la computadora. En la visión que aquí se propone hay dos roles, siendo el nuevo rol el más importante: comunicar ideas a personas.

cuál es esa solución para las personas que lo piensan y lo leen, o sea, **ser una forma de ordenar las ideas propias, expresarlas en el lenguaje de programación y comunicarlas a otras personas** (además de ser una "receta" para hacer funcionar una máquina). A partir de esto, una consideración importante a realizar es que si bien el lenguaje de programación es un vehículo para comunicar ideas, también influye en la forma de concebir las soluciones. Dicho de otra manera, el lenguaje, además de funcionar como herramienta, tiene gran influencia en la manipulación mental de los elementos de la solución, ya que permite identificarlos y expresarlos, y establece reglas claras de qué cosas se pueden hacer con ellos y qué cosas no. Esta influencia puede ser mayor o menor según la formación de la persona que programa, pero no es algo que pueda soslayarse completamente. Y es por eso que el lenguaje de programación es al mismo tiempo poco importante y muy importante: es una herramienta para expresar las ideas, pero al mismo tiempo las influye y demarca.

La idea de que un programa es una forma de comunicación entre personas queda plasmada en una herramienta conceptual que se denomina *legibilidad*. Se dice que un programa es *legible* cuando puede ser leído con sencillez por una persona, y entendido con poca o ninguna explicación adicional. Este concepto es sutil, pues la comprensión depende mucho también de la formación y capacidad de cada persona, pero no debe descuidarse este aspecto al enseñar programación. En el curso que aquí se analiza, el concepto de legibilidad juega un rol preponderante, y se recomienda que sea otra de las ideas que se remarquen de manera continua en cada ejercicio. La legibilidad se ve expresada en la actividad de programación a través de la *elección de nombres adecuados* para las entidades que se escriban en el programa.

Al considerar las herramientas que los lenguajes de programación proveen para construir programas, se hace evidente que las hay de dos tipos: las que deben aplicarse de manera estricta (p.ej. nombres de comandos y palabras clave, coincidencia entre parámetros y argumentos, etc.) y las que requieren consideraciones de estilo por parte de la persona que escribe el programa. En el segundo grupo podemos encontrar la determinación de nombres adecuados para procedimientos, parámetros, variables y otras entidades. Si bien no existe una definición precisa de qué constituye un nombre adecuado, sí existen criterios que permiten distinguir nombres terriblemente inadecuados de otros más adecuados; la determinación final sobre la correcta adecuación de un nombre dependerá en última instancia del programador que escriba o lea el programa, pero se puede y debe trabajar sobre criterios generales para nombrar entidades. Por ejemplo, es universalmente aceptado que los nombres de una o dos letras son extremadamente inadecuados en la mayoría de los casos, y también que los nombres extremadamente largos (por ejemplo, que ocupan todo un renglón o más), también. Se suelen favorecer nombres cortos pero descriptivos, que puedan ser leídos en voz alta en lenguaje natural (castellano, en nuestro caso); en el caso de los comandos, por ejemplo, estos nombres usualmente refieren a verbos en infinitivo (ver las consideraciones de la sección 3.3). Adicionalmente, se requiere que el nombre brinde una idea correcta respecto del propósito de la entidad nombrada, pues si no usualmente conduce a confusiones que limitan o complican la legibilidad (por ejemplo, es inadecuado nombrar con el identificador "`Rojo`" a un valor que luego se utiliza para dibujar figuras verdes, o nombrar como "`Avanzar un paso adelante`" a un procedimiento que permanece en el lugar; en cambio habrían sido nombres adecuados "`Verde`" y "`Permanecer en el lugar`", respectivamente).

En los ejemplos considerados en el cuaderno la gran mayoría de los nombres elegidos son adecuados. Sin embargo, una revisión exhaustiva posterior a la publicación mostró que algunos de los nombres utilizados, tanto en el cuaderno mismo como en la ejercitación distribuida con él, tal vez no son los mejores y podrían ser mejorados, y es conveniente que el docente pueda corregirlos al dictar el curso. Por ejemplo, en la secuencia didáctica 11, "Parámetros en Scratch", que utiliza el ejemplo "El Planeta de Nano" (ver sección 4.11), el comando primitivo para pasar a la siguiente fila se nombra como `Siguiente fila`, en lugar del más adecuado `Pasar a siguiente fila`. Al presentar cada ejercicio en la sección 4, se indicará cuando, desde el punto de vista de la legibilidad, algún nombre no resulte completamente adecuado, y se sugerirán modificaciones.

Ya se mencionó que el curso está pensado para ser usado con diferentes públicos: primarios, secundarios, docentes. Y también se mencionó que la legibilidad es una cuestión subjetiva, donde hay ejemplos claros de lo que constituye un nombre totalmente inadecuado, pero donde el grado correcto de adecuación depende mucho de la persona que lee el programa. Por eso, a la hora de decidir qué nombres constituyen buenas elecciones para garantizar la legibilidad del programa se debe tener en cuenta el nivel de maduración respecto del uso del lenguaje que tengan los estudiantes del curso. Por ejemplo, al trabajar con chicos más chicos, con un grado de desarrollo del lenguaje menor, deben elegirse nombres cortos y evocativos (por ejemplo "`Avanzar`"), mientras que al trabajar con adultos puede optarse por nombres algo más extensos que garanticen mayor precisión (por ejemplo "`Avanzar un paso`"), sin que la extensión se transforme en algo que pueda volverse molesto de leer, escribir y recordar (por ejemplo "`Avanzar el ratón un paso en el laberinto hacia la salida`"). Adicionalmente, la elección de nombres puede usarse como una manera de trabajar aspectos del lenguaje natural y expresión oral y escrita, ya que finalmente un buen criterio para la elección de nombres se basa en la manera en que las personas se expresan.

Para cerrar esta sección, es importante volver a remarcar que este concepto debe ser explicitado y reforzado en cada uno de los ejercicios que se trabaje, a lo largo de todo el curso, ya que se trata de un concepto central, vinculado a la concepción de los programas como vehículo de comunicación entre personas.

3.2.3 Algorítmica básica: recorridos

La última de las herramientas conceptuales que deben vertebrar este curso está vinculada con la algorítmica. Ahora bien, el tema de algoritmos es demasiado complejo y extenso para ser tratado de manera razonable en un curso inicial de programación, y menos aún si el curso no está destinado a formar programadores profesionales, tal como fuera establecido al comienzo de la sección 3. Sin embargo, es lo suficientemente importante como para que algún tipo de algorítmica básica se haga necesaria. Por ello, en este curso, además de las formas básicas algorítmicas de alternativas y repeticiones simples, se considera una forma elemental de algoritmo para tratamiento de secuencias de elementos, expresada en una forma de división en subtareas que en esta propuesta se denomina *recorrido*.

Básicamente, un recorrido es un esquema de repetición donde se realiza una tarea para cada uno de una serie de elementos, pues recorre la secuencia de elementos de a uno, procesando de alguna forma cada uno de ellos. Este concepto está basado en ideas científicas y didácticas tales como[4] el uso de invariantes de ciclo, la idea de folds (especialmente foldr) y generalizaciones asociadas, y el enfoque de esquemas básicos de algoritmos de Scholl y Peyrin, y fue presentado junto con la didáctica innovadora que inspira este texto [ML13]. La idea de un recorrido es organizar la división en subtareas a la hora de resolver una tarea que requiere trabajar con cada uno de los elementos de una secuencia de manera tal de simplificar la confección de una solución y garantizar el correcto funcionamiento de la misma. De esta forma, habrá subtareas para iniciar el recorrido, determinar si se terminaron los elementos, procesar un elemento, pasar a considerar el siguiente elemento y para finalizar el recorrido. Estas subtareas se organizarán a través de diferentes herramientas del lenguaje (ver sección 3.3), que pueden tener diferentes variaciones según el problema bajo consideración.

La secuencia de elementos que un recorrido se encarga de procesar puede ser muy concreta (por ejemplo, una serie de elementos en fila, como ser frutas o señales) o ser más abstracta (por ejemplo, una serie de posiciones distantes a visitar en un mapa o incluso un recorrido sobre números para resolver un problema numérico tal como calcular un factorial). Sin embargo, cuanto más abstracta resulte la secuencia, más complejo será transmitir la idea adecuadamente. Por eso, se recomienda que en un curso inicial los recorridos se mantengan lo más concretos que sea posible. Los ejemplos del cuaderno fueron elegidos para garantizar esta cuestión.

[4] Puede consultarse la bibliografía para mayor detalle de los antecedentes de esta idea.

Para cerrar esta sección, se debe reiterar la importancia de considerar y explicitar adecuadamente las herramientas conceptuales a docentes que toman el curso para aprender a dictarlo, ya que son estas herramientas las que finalmente le dan sentido a la formación buscada como objetivo último.

3.3 Herramientas del lenguaje

Al pensar la tarea de programación, se debe tener en cuenta que la forma de expresar las ideas y soluciones propuestas es, invariablemente, a través de un lenguaje de programación. Un lenguaje de programación determinado establece cuáles son las maneras de escribir programas en dicho lenguaje, y brinda una serie de herramientas que se pueden utilizar para expresar diferentes conceptos. Cada lenguaje de programación específico provee diferentes herramientas, y diferentes formas de combinarlas, pero todas esas formas se pueden entender como casos particulares de una serie genérica de herramientas. En esta sección se explica el conjunto de herramientas de lenguajes de programación que se propone brindar en este curso, y la forma de clasificarlas y entenderlas. Estas herramientas fueron elegidas por ser genéricas, ya que se manifiestan de una u otra manera en cada lenguaje de programación, y sirven como base conceptual para entender cualquier lenguaje nuevo que se desee aprender, con mayor o menor dificultad según el lenguaje de que se trate y la diferencia en forma entre las herramientas de ese lenguaje y las aquí presentadas.

La clasificación propuesta es suficientemente general, y es aplicable a la mayoría de los lenguajes de programación modernos. Por esa razón, presentar esta clasificación como parte del marco conceptual es una contribución valiosa a los objetivos del curso.

Las herramientas del lenguaje se pueden clasificar según el tipo de elementos que describen: los comandos se utilizan para describir acciones y las expresiones se utilizan para describir datos. En un lenguaje natural, las acciones se describen mediante verbos y los datos se describen mediante sustantivos. Por esa razón, se puede establecer la analogía y pensar a los comandos como "verbos" y a las expresiones como "sustantivos". El programa se arma combinando "verbos" y "sustantivos" para brindar a la máquina que lo ejecutará con la descripción adecuada para que pueda llevar a cabo la solución. Los comandos le indicarán acciones que debe realizar, y las expresiones le indicarán con qué datos deben llevarse a cabo las acciones. Pensar a los elementos del lenguaje desde esta perspectiva contribuye a entender las restricciones a la hora de combinar herramientas. Por ejemplo, un comando puede usarse con expresiones como argumentos, pero no con otros comandos, de la misma forma que los verbos en un lenguaje natural se combinan con sustantivos, pero no usualmente con otros verbos[5].

Cuando se elige entender a los elementos del lenguaje desde esta perspectiva, la elección de nombres que se trató en la sección 3.2.2 implica que los comandos se designarán mediante verbos y los datos mediante sustantivos. Así, es adecuado tener, por ejemplo, un comando `Avanzar` que toma como argumento el número de pasos a dar: `Avanzar(10 pasos)`, pero no lo es si el comando se denomina `Adelante`, pues la intención es que el mismo describa una acción, y el nombre "Adelante" describe una dirección y no una acción.

3.3.1 Comandos

Habiendo explicado la división de las herramientas según el tipo de elementos que describen, es conveniente profundizar en cada una de las categorías. Es usual comenzar por los comandos ("verbos"), pues los mismos tienen efectos claramente identificables a la hora de ejecutar un programa (en especial en los modelos concretos elegidos como base de las actividades del

[5] Existen lenguajes de programación modernos (en programación funcional, por ejemplo) donde estas ideas están relajadas o extendidas, y no es tan fácil encontrar esta analogía. Pero la misma sirve como punto de partida en la comprensión de lenguajes de programación.

curso), y entonces resulta más fácil, para quién recién se inicia en programación, identificarlos y manipularlos.

Los comandos se pueden clasificar de diversas maneras. La forma que se propone en esta guía tiene que ver con la naturaleza de cada comando, y con el propósito para el que se lo utiliza. La base de esta jerarquía la proveen los comandos primitivos, que son las descripciones de las acciones elementales que la máquina que ejecutará el programa puede realizar; cada máquina considerada tendrá diferentes capacidades, lo cual se verá reflejado en el conjunto de comandos primitivos que se manejen desde el lenguaje. En los diferentes ejercicios del cuaderno los comandos primitivos van variando, y es importante entender que este conjunto constituye la base con la cual construir las soluciones, y que diferentes conjuntos de comandos primitivos (o sea, diferentes máquinas con diferentes capacidades) requerirán, posiblemente, diferentes estrategias de solución.

La primer forma de combinación de comandos primitivos (y luego, de comandos en general) es la *secuenciación*. Una secuencia de dos comandos se suele escribir poniendo un comando y otro a continuación (ya sea en una nueva línea o en la misma línea separado de alguna forma – por ejemplo un punto y coma); la acción completa que una secuencia describe corresponde a realizar la acción indicada por el primer comando y habiendo terminado esa acción, realizar la acción indicada por el segundo comando. Es la forma básica de combinación, que resulta imprescindible para construir acciones más complejas.

La siguiente forma de combinación de comandos primitivos que se propone trabajar es el ***procedimiento***. Este es uno de los puntos claves del enfoque que se propone en este cuaderno. La razón para ello es que los *procedimientos* son la herramienta que plasma de manera más precisa la noción de abstracción que se busca transmitir, y por eso se consideran una herramienta fundamental. Un procedimiento en su forma más simple no es más que una acción compleja (obtenida, por ejemplo, por secuenciación de acciones más elementales) a las que se les asigna un nombre que la identifica. Puede entenderse esta herramienta también como una forma de definición de nuevos comandos, a través del mecanismo de proveer un nombre para el nuevo comando y describir la acción que ese comando denominará mediante la combinación de acciones previas. Por ejemplo, si se cuenta con el comando primitivo `AvanzarUnPaso()`, se puede definir el procedimiento `AvanzarTresPasos()` como la secuencia `AvanzarUnPaso();AvanzarUnPaso();AvanzarUnPaso()`. Tal cual se describió en la sección 3.2.2, la elección de un nombre adecuado que describa de manera precisa el propósito del procedimiento es básica para la correcta legibilidad del programa.

Los procedimientos constituyen uno de los pilares fundamentales de este enfoque, y no deben descuidarse en ningún momento a lo largo del curso. Son la forma de explicitar en el lenguaje la división en subtareas, tal cual se estableció en la sección 3.2.1, y por ello comparten la importancia de dicha herramienta conceptual. Así como se estableció que la división en subtareas es el concepto más importante, y puesto que los procedimientos son la forma de manifestar esta herramienta en el lenguaje, los procedimientos están al mismo nivel de importancia, pero del lado del lenguaje. O sea, división en subtareas y procedimientos pueden verse como dos instancias, una del lado conceptual y la otra del lado del lenguaje, del mismo concepto fundamental: la *abstracción*.

Es importante hacer una observación sobre la denominación de *procedimiento* para este concepto, ya que la misma no es utilizada por todos de la misma forma. En algunas herramientas o enfoques lo que aquí se denomina procedimiento se puede encontrar nombrado como *funciones*, *bloques*, *subrutinas*, o alguna otra forma. Pero en cada caso, es claro que se trata de un mecanismo de abstracción sobre comandos, una forma de definir nuevos comandos. Se sugiere utilizar la denominación aquí propuesta, incluso al usar entornos que utilizan otras denominaciones. La razón para esto es que se busca transmitir el concepto de manera uniforme e independiente de la manera de denominarlo en cada entorno o lenguaje (por ejemplo, *método*, *subrutina*, *procedimiento*, etc.

En su forma más simple, los procedimientos solamente le dan nombre a alguna acción com-

pleja definida a partir de combinación de otros comandos. Sin embargo, el verdadero poder expresivo de los procedimientos se encuentra cuando se los combina con la noción de *parámetro*. Los parámetros son una herramienta que permite expresar muchos procedimientos diferentes mediante un único procedimiento parametrizado. Un parámetro es como un "agujero" que el procedimiento tiene, y que debe ser completado con un dato: el *argumento*. El argumento provisto para completar el parámetro determina cuál de las instancias específicas expresadas por el procedimiento parametrizado se busca utilizar.

Los parámetros son otra forma de conseguir abstracción: muchos casos particulares son expresados de una sola vez a través de un procedimiento parametrizado. Sin embargo, esta forma de abstracción es mucho más difícil de transmitir que la de procedimiento, especialmente a estudiantes que recién se inician en la programación. Por esa razón, en este curso los parámetros aparecen de manera tardía, y no se hace un fuerte hincapié en ellos. La noción de parámetro y parametrización es crítica en la formación de un programador profesional y no tanto al tratar la programación como un elemento de cultura general; sin embargo, a pesar de que puede tratarse de una manera menos profunda en un curso que solo transmita nociones generales, no debe faltar.

Como se estableció al comienzo de esta sección, la didáctica es una de las herramientas más importantes de este curso. Esto es especialmente importante a la hora de transmitir las nociones de procedimiento y parámetros, por lo que se recomienda prestar mucha atención a la forma en que el cuaderno propone hacerlo. Para procedimientos, se trata de comenzar pensando la estrategia de solución, y capturar las grandes subtareas que constituyen dicha estrategia, y luego expresar las mismas a través de procedimientos. Para el caso de parámetros, se trata de escribir varios procedimientos simples similares (por ejemplo, dibujando cuadrados de distinto tamaño), y luego expresarlos en un único procedimiento parametrizado que reciba el tamaño como parámetro. Y en ambos casos, no olvidarse que la indagación es un elemento crucial. Si se lo realiza de manera adecuada y correcta, las ideas de procedimiento y parámetros son muy accesibles y rápidamente incorporadas por los estudiantes.

La clasificación de comandos continúa con diversas formas de combinación de comandos: dos formas de repetición (simple y condicional) y una forma de alternativa (condicional). Estas herramientas permiten aumentar el poder expresivo de la persona que programa al habilitar o facilitar comportamientos complejos, que son o bien difíciles o bien imposibles de expresar utilizando solamente las herramientas previas. Nuevamente, la didáctica por indagación debe ser tenida en cuenta al presentar las herramientas, para que de esa forma la utilización de las mismas no sea el resultado de una receta, sino el fruto de la necesidad. Los ejercicios propuestos guían en este enfoque.

A la hora de presentar repeticiones es conceptualmente más simple comenzar con una forma de repetición que repite un número fijo de veces. Esta forma es denominada aquí *repetición simple*, y si bien es expresable mediante formas más complejas (al punto que en muchos lenguajes de uso profesional o comercial no existe), es parte de esta didáctica el presentarla por separado y primero. Habiendo dominado esta parte, se puede pasar a dominar las formas más avanzadas.

La forma avanzada de repetición que se presenta en el curso es la que se denomina *repetición condicional*, pues la cantidad de repeticiones no se conoce a priori, y se utiliza una condición como mecanismo para determinar cuándo dejar de repetir. Esta forma de repetición es conocida por muchos con nombres menos descriptivos, como *"while"* o *"do while"* (mientras) o *"repeat until"* (repetir hasta); sin embargo es conveniente utilizar la denominación de repetición condicional, pues es más descriptiva y adecuada a la hora de ubicarla en el marco conceptual. También importa la dificultad didáctica de cada una de las variaciones de la repetición condicional. En este curso se propone la utilización de la forma "repetir hasta" por ser más sencilla desde el punto de vista didáctico que su equivalente "repetir mientras". Esto se refleja en los ejercicios propuestos, y no debe ser pasado por alto.

La misma cuestión con respecto a la denominación adecuada de la repetición sucede con la *alternativa condicional*: resulta conveniente denominarla de esta forma descriptiva y no con

formas más familiares. La alternativa condicional es una forma de elegir entre dos posibles cursos de acción, basándose en una condición. Es comúnmente conocida como *"if-then-else"* o simplemente *"if"* o a veces *"condicional"*. Pero estas formas familiares de denominarla ignoran el hecho fundamental de que lo que se está expresando es una **alternativa** entre dos posibles acciones. Además, al presentar alternativa condicional (lo que es conveniente hacer antes de presentar la repetición condicional, como se explica en la sección 3.4), deben presentarse las formas de establecer condiciones (ver la sección 3.3.2).

La última de las herramientas conceptuales vinculadas a los comandos tiene que ver con la necesidad de recordar información durante la ejecución de un programa. Desde el punto de vista de los comandos, esta herramienta es la *asignación de variables*. La idea es utilizar una variable (ver la sección 3.3.2 que explica la idea de variable) para recordar información, y la *asignación* es el comando que describe la acción de recordar dicha información. En los enfoques tradicionales, las variables y la asignación son presentados casi al inicio del curso, como un comando primitivo y el énfasis está puesto en las variables, y no tanto en la asignación. Esto es así porque estos enfoques se concentran exclusivamente en computadoras clásicas, preocupándose más por el funcionamiento de dichas máquinas que por la forma de pensar y entender la programación. En el enfoque que describimos aquí la asignación juega un rol menor; es una herramienta necesaria, pero no es considerada dentro del conjunto de herramientas fundamentales, y por esa razón es dejada para el final. El cuaderno no incluye actividades con variables y asignación, pero el curso se complementa con una serie de actividades adicionales (ver sección 4.15) para tratar este tema como última herramienta del curso, y solamente si los tiempos lo permiten. La razón para este enfoque, donde la asignación es relegada al final, es que la correcta utilización mediante asignación de variables y memoria es extremadamente compleja, y suele ser fuente de numerosísimos errores y problemas a la hora de confeccionar programas, además de que resulta complejo aprenderlo y, por lo tanto, inconveniente para un curso introductorio.

El estilo de programación que se utiliza en las actividades, y que no incluye variables como algo imprescindible, expresa adecuadamente el enfoque conceptual y las herramientas conceptuales sobre las que la iniciativa Program.AR propone trabajar. En cambio, en el estilo de programación basado en variables es mucho más complejo y sutil percibir este enfoque. Por ello, la postergación (o incluso eliminación) de la herramienta de asignación es otro de los aportes importantes de este enfoque, y no debe ser modificado sin un análisis profundo de las consecuencias.

3.3.2 Expresiones

La siguiente categoría de elementos de un lenguaje de programación que consideramos es la de *expresiones*, que son la manera de describir información, y por ello equiparables a los "sustantivos" de un lenguaje natural. En su forma más básica, las expresiones aparecen en forma literal, por ejemplo para representar números (`2`, `17`, `42`, etc.) o cadenas de caracteres (`"hola"`, `"Este es un string"`, etc.) o valores de verdad (`verdadero` y `falso` que en lenguajes con base en inglés serían `True` y `False`) o, en algunos lenguajes, algunas otras variaciones de formas literales. Estas expresiones literales constituyen la base, junto con un par más de formas primitivas, sobre la que construir datos o información más compleja, a partir de diversas formas de combinación. Al principio de un curso inicial es conveniente no profundizar demasiado en la naturaleza de las expresiones, o en su combinación, puesto que estos valores literales son vistos con naturalidad por los estudiantes. No es si no hasta promediado el curso que resulta útil presentar algunas formas más de combinación y construcción de expresiones, y siempre como fruto de la necesidad.

Antes de continuar con las formas de combinación, y si bien se presentan más adelante en la secuencia didáctica (ver la sección 3.4) vale la pena detenerse en las otras formas primitivas utilizadas:

- por un lado, nombres que representan información de manera indirecta, a saber, *paráme-*

tros y *variables*, y

- por el otro, el conjunto de sensores y datos primitivos que representan la información que la máquina puede recolectar del medio ambiente en el que ejecuta.

Es poco usual en los enfoques tradicionales visibilizar a los nombres (parámetros y variables) y a los sensores primitivos como expresiones básicas, puesto que en dichos enfoques el énfasis está puesto en las acciones que las máquinas realizan. En el enfoque Program.AR, si bien las expresiones no reciben tanta atención como los comandos, sí resulta conveniente remarcar el hecho de que son una categoría por sí misma y que las formas en que se piensan y combinan expresiones sigue principios similares a las encontradas en comandos. Esto, además de por su valor didáctico, es especialmente útil porque habilita de manera óptima la continuación posterior del estudio de estructuras de datos, que en este curso no aparecen.

En el caso de los parámetros y variables, es interesante ver que ambos son un caso particular de formas de nombrar valores de manera indirecta. Tanto un parámetro como una variable describen un dato fijo. Lo que distingue a parámetros de variables es la forma en que ambos toman valor, la forma en que pueden variar el valor descrito, y la validez del nombre durante la ejecución del programa. Un parámetro toma su valor al momento de invocación de un procedimiento (o función), e idealmente, no debería variar durante la ejecución de dicho procedimiento. Además, su validez es solamente durante la ejecución del procedimiento que lo define. En cambio una variable toma su valor en la primer asignación que la menciona, y puede variar a voluntad de la persona que programa en futuras asignaciones, teniendo diferentes formas de validez en diferentes lenguajes (lo que da lugar a diversas denominaciones, como *variables locales*, *variables globales*, *variables de instancia*, etc.). En lenguajes de uso profesional o comercial, donde la didáctica no es una prioridad, es usual mezclar estas nociones, y así los parámetros funcionan también como variables. Sin embargo, es importante desde el punto de vista didáctico distinguir estas dos nociones, y considerarlas como formas particulares de una noción más general, que es la de nombrar datos de manera indirecta. De ahí que se haya utilizado la designación de *nombres* para hablar en forma común de parámetros y variables.

En el caso de los sensores primitivos, y al igual que sucede con los comandos primitivos, el conjunto disponible de los mismos en el lenguaje queda determinado por las habilidades de la máquina que ejecuta el programa. Y de la misma manera, la disponibilidad o no de algún sensor específico implicará que algunas estrategias de solución sean factibles o no resulten posibles. Nuevamente, esta noción no es remarcada por los enfoques de aprendizaje tradicionales, pues los lenguajes profesionales más difundidos utilizan formas combinadas que son al mismo tiempo comandos y expresiones, y entonces los sensores primitivos quedan subsumidos dentro del grupo de los comandos primitivos. Pero, una vez más, la necesidad didáctica lleva a que se haga necesario y útil el considerarlos como un concepto separado.

Las expresiones literales para describir valores, los nombres, y los sensores y datos primitivos se pueden utilizar para construir expresiones más complejas, combinando estos elementos (y otras expresiones previamente construidas) mediante ciertas formas de combinación. Para el caso de los números es fácil de ver: dados dos literales numéricos, se puede sumarlos, multiplicarlos, o combinarlos mediante alguna otra operación aritmética. Estas operaciones no solo se realizan sobre literales numéricos, sino sobre otras expresiones que describan números. Así, es posible escribir `((2+3)*8)+2`, y otro montón de "cuentas"; lo mismo sucede con los nombres (sean parámetros o variables) que describen números, y así es posible escribir `edad+1`, `(base*altura)/2`, etc. Estas formas de combinación existen para cada uno de los tipos de datos que se consideren. Las más sencillas de ejemplificar son los números, puesto que los mismos se aprenden desde el inicio de la escolarización. Para otros tipos las formas de combinación adoptan otras formas: por ejemplo los valores de verdad (también conocidos como *booleanos* en honor del matemático George Boole, que los estudió) pueden combinarse mediante negación, conjunción y disyunción. En el caso de los booleanos, este enfoque propone no ir más allá de lo que sea necesario para resolver los ejercicios planteados y pensarlo de manera intuitiva e

informal, en contraposición con el enfoque tradicional que consiste en explicar todas las formas de combinación posible y analizarlas formalmente (por ejemplo, mediante tablas de verdad). Esto es consecuencia de la utilización de la didáctica por indagación. Solo en cursos avanzados, donde el objetivo es formar profesionales de la programación, tiene sentido tratar formalmente a los valores de verdad.

Así como en la categoría de los comandos existen los procedimientos como una forma de definir nuevos comandos no primitivos (o sea, de *abstraer* comportamiento), en la categoría de las expresiones existe una herramienta equivalente: las *funciones*. Una *función* es una expresión compleja a la que se le asigna un nombre, y de esa forma define un nuevo sensor o dato no primitivo. Por ejemplo, si el lenguaje provee un sensor que indica la cantidad de bolitas de cierto color, `cantidadDeBolitasDeColor(Rojo)` y lo mismo para otros colores, y suponiendo que `Rojo`, `Verde` y `Azul` son los únicos colores disponibles, por ejemplo, podría definirse una nueva función `cantidadTotalDeBolitas()` como `cantidadDeBolitasDeColor(Rojo) + cantidadDeBolitasDeColor(Verde) + cantidadDeBolitasDeColor(Azul)`. Es importante notar que, en sintonía con la idea de legibilidad discutida en la sección 3.2.2, los nombres de los sensores primitivos y de las funciones deberían ser sustantivos (`cantidadDeBolitasDeColor`, `cantidadTotalDeBolitas`, etc.).

En el cuaderno esta herramienta no se presenta de manera explícita, pero la misma aparece en algunas de las actividades complementarias en otros entornos (como Alice). Además, la correspondencia de este concepto con el de procedimiento, y consecuentemente con el concepto de división en subtareas, junto a que el concepto aparece en algunos de los entornos propuestos, hace que valga la pena incluirlo en el mapa conceptual y considerarlo de manera propia. Sin embargo, puesto que se trata de un concepto sutil y complejo, puede resultar necesario quitarlo de los temas a dictar en algunos cursos, especialmente si no se dispone de tiempo o la audiencia puede beneficiarse más de profundizar conceptos más importantes.

El verdadero poder de las funciones aparece cuando se las combina con parámetros, de manera similar a lo que ya fue explicado para los procedimientos. Sin embargo, y en virtud de lo explicado en el párrafo anterior acerca de las funciones, y en secciones anteriores acerca de los parámetros, es decisión no profundizar mayormente este aspecto. Los parámetros son el concepto más complicado de transmitir, y puesto que habiendo trabajado el concepto de procedimiento, las funciones son similares, el tratamiento del tema de funciones parametrizadas solo aporta en tanto profundización de conceptos y completitud del marco conceptual.

Un elemento que vale la pena mencionar por separado, aunque desde un punto de vista técnico es similar a los sensores primitivos es el de los sensores de interactividad. Para poder realizar programas donde la máquina puede consultar al usuario que la utiliza (por ejemplo, en el caso de juegos interactivos) es necesario contar con alguna herramienta que permita, desde el lenguaje, describir tal interacción. Existen numerosos modelos para expresar interacción en lenguajes de programación, pero su análisis y comprensión es tarea de profesionales. Sin embargo, la posibilidad de escribir y utilizar programas interactivos posee un alto valor didáctico, y por esa razón en este enfoque se decidió proveer una forma básica de interactividad. Los sensores de interactividad funcionan de manera similar a los sensores primitivos, pero en lugar de sensar datos fijos, sensan las respuestas del usuario (por ejemplo, a teclas presionadas). En la clasificación de los elementos del lenguaje realizada fueron ubicados de manera separada simplemente porque constituyen una forma específica de proveer interactividad, y al separarlos, habilita a que se los comprenda de manera independiente, permitiendo posteriormente su evolución a formas más complejas de interactividad en caso de ser necesario.

3.4 Presentación cronológica del marco conceptual

La presentación del marco conceptual analizada en las secciones previas lo muestra de manera completa, utilizando como idea rectora la clasificación de los elementos desde un punto de vista conceptual, sin tener en cuenta el factor del orden en que los mismos deberían ser presentados

para maximizar los beneficios didácticos. En esta sección se considera el mismo marco, pero desde el punto de vista de su presentación cronológica, intercalando elementos y conceptos de las diferentes categorías, e incluso en ocasiones volviendo a reforzar algunos de ellos. Una de las *contribuciones importantes* desde el punto de vista didáctico que posee este enfoque es, justamente, la *adecuada secuenciación de los conceptos, teniendo en cuenta consideraciones respecto de su complejidad conceptual, de su importancia y de la necesidad de su utilización en virtud del aprendizaje por indagación*. Por ello, es recomendación de esta guía no alterar el orden propuesto sin una comprensión cabal de las razones por las cuales ese orden fue diseñado, y de las consecuencias de dicha alteración al aprendizaje propuesto.

El primer tema a considerar es el de la didáctica, ya que todas las actividades estarán orientadas a dicho enfoque, y es necesario que el docente que dicta el curso no se desvíe de la misma. El eje del marco conceptual concerniente a la didáctica permea todo el curso.

Los siguientes temas a trabajar son la noción de programa como solución a un problema computacional, la composición de un programa a partir de comandos primitivos y secuenciación, y la idea de estrategia de solución al analizar el problema a ser resuelto. Como se mencionó, la estrategia de solución conduce naturalmente a la técnica de división en subtareas, que es la herramienta fundamental a trabajar en el curso. De forma consecuente con esta importancia, y como también fue mencionado, la siguiente herramienta del lenguaje a presentar es los procedimientos, ya que los mismos son la manera de plasmar desde el lenguaje la idea de división en subtareas. Esta herramienta es fundamental y debe fomentarse su utilización a lo largo de todo el curso. Junto con la noción de procedimiento debe incorporarse la idea de legibilidad, expresada fundamentalmente a través de la adecuada elección de nombres para los mismos.

Habiendo presentado las herramientas más importantes (didáctica por indagación, división en subtareas, procedimientos, elección de nombres), es posible continuar presentando herramientas que facilitan la programación o habilitan soluciones más complejas y poderosas. Así, la siguiente herramienta a presentar es la repetición simple, motivándola en la necesidad de no repetir la escritura de comandos primitivos o definidos. Como fue explicado, es mejor comenzar con la repetición simple por su pureza y simplicidad conceptual, postergando la presentación de formas más complejas. Junto con la noción de repetición simple aparecen las expresiones que describen valores de forma literal, en la forma de números utilizados para indicar la cantidad de repeticiones.

Luego de esto, la siguiente herramienta propuesta es la alternativa condicional, junto con la noción de sensores primitivos. Acá es conveniente trabajar con cierta delicadeza estos conceptos, pues son algo menos obvios que los anteriores, y a muchos estudiantes les suele resultar más complejo entender la noción de valor de verdad, la noción de sensor primitivo y la noción de alternativa. Se recomienda hacer fuerte uso de las fortalezas de la ejercitación propuesta. La motivación para el uso de esta herramienta viene dada por la necesidad de considerar escenarios que pueden variar de una ejecución a otra, al principio solo de elementos presentes o ausentes, y consecuentemente el programa debe contemplar alternativas para las posibles variaciones.

Recién luego de manejar con cierta fluidez la noción de alternativa y todas las anteriores, es razonable presentar la forma más compleja que es la repetición condicional. Como fue mencionado, la variante propuesta de repetición condicional es la de "repetir hasta", por resultar usualmente más intuitiva para personas sin experiencia previa. Y puesto que la repetición condicional es una herramienta muy poderosa, que requiere mucha habilidad para su dominio, junto con esta herramienta debe presentarse la idea de recorridos, como forma de estructurar la utilización de la repetición y también la de combinaciones básicas de expresiones booleanas (negación y conjunción). La motivación para la repetición condicional viene también de la necesidad de considerar escenarios con variaciones, en esta ocasión variaciones de cantidad y no solo de elementos presentes o ausentes. Por otra parte, la motivación para los recorridos está dada por la complejidad propia de la repetición condicional y la necesidad de controlarla.

Siguiendo con la idea de aumentar gradualmente la complejidad conceptual, la siguiente herramienta a presentar es parámetros y parametrización. Tal cual se explicó, la idea no es hacer

una exposición detallada de todos los usos y variaciones, sino la de presentar la idea de manera simple (como "agujeros" que al ser llenados permiten obtener diferentes procedimientos similares, pero diferentes). En simultáneo con la idea de parámetro (que requiere volver a trabajar la elección de nombres y la legibilidad), se pueden presentar formas de combinación más complicadas para los números, como ser utilizar cuentas sobre los parámetros (p.ej. `360/numeroDeLados` para los grados de un ángulo).

El último tema tratado por el cuaderno es el de interactividad, a través de la presentación de sensores de interactividad. Las actividades de esta parte del curso son útiles para reforzar además todos los conceptos vistos con anterioridad.

Fuera del cuaderno, pero como posibles extras disponibles desde la página de Program.AR, pueden verse variables y asignación y uso de variables. Este tema resulta complejo, y es conveniente considerarlo con cuidado a la hora de agregarlo.

Finalmente, el tema de las funciones solo puede trabajarse si se cambia de entorno y se utiliza Alice en lugar de Scratch, algo que puede resultar provechoso como ejemplo de que los conceptos trabajados no se limitan a un único entorno o a un único lenguaje.

Para cerrar la discusión sobre el marco conceptual, queda considerar la conveniencia o no de explicitarlo durante el curso. La decisión de trabajar el marco conceptual de manera explícita está vinculada fuertemente al tipo de público al que va dirigido el curso, y a los objetivos de formación buscados. En un curso donde el público son estudiantes de primaria o secundaria, y el objetivo solo es tratar a la programación como un tema de cultura general, no es necesario explicitar el marco conceptual: los estudiantes viven la ejercitación como un juego, y adquieren naturalmente los conceptos trabajados. En cambio, cuando el público esté compuesto por docentes, y el objetivo sea formar futuros formadores en temáticas de programación, resulta casi imprescindible explicitar el marco conceptual y la secuenciación del mismo, puesto que eso provee a los docentes que toman el curso de un punto de referencia desde el cual enmarcar y juzgar la ejercitación, y les permite planificar sus propios dictados con mucha mayor confianza. Todo docente debe conocer en qué marco conceptual se inscriben las ideas que busca transmitir, y este curso no es la excepción.

Parte II
Actividades

Esta parte presenta el análisis de las actividades propuestas por el cuaderno en referencia al marco conceptual explicado en la primera parte, y provee consejos a posibles situaciones que surjan durante la preparación y el dictado del curso. No es imprescindible su lectura secuencial, sino que cada subsección puede abordarse como parte de la preparación de la clase que incluya la secuencia didáctica correspondiente.

4 Valorización de la ejercitación

En esta sección se revisan las soluciones provistas en el cuaderno para cada uno de los ejercicios propuestos, y se discuten sus características y posibles alternativas, junto con algunas recomendaciones respecto a qué dificultades pueden encontrarse al trabajar con ellos junto a los estudiantes y cómo manejarlas y a cómo medir la calidad de las soluciones ofrecidas por los estudiantes. Para estas cuestiones se utilizará el marco conceptual presentado previamente como referencia en la sección 3, guiando de esta forma la valorización de la ejercitación propuesta.

El curso propuesto está pensado originalmente para ser dictado a estudiantes de entre 11 y 14 años. Esto se refleja en la elección de los personajes y temas de ejercitación y el nivel de los actividades que se proponen. Pero como se mencionó antes, es posible utilizar el material para aplicarse con otros públicos, tanto con chicos más chicos o más grandes, como con docentes que vayan a dictar el curso u otros adultos. En el caso de chicos de edades mayores es necesario tener en cuenta que estos estudiantes tienden a aburrirse con las temáticas por resultarles demasiado infantiles y sienten cierto rechazo, que es mayor entre adolescentes de 15 a 18 años que en personas adultas. Por otra parte, los estudiantes menores tienen dificultades para comprender las cuestiones técnicas más complicadas.

Otro aspecto para remarcar es que varias de las actividades que propone el cuaderno son para realizar sin contar con computadoras. Esta característica es importante por varias razones.

- Los estudiantes, especialmente los más jóvenes, tienen una tendencia a apropiarse mejor de los conceptos cuando los mismos son más concretos, y consecuentemente las actividades sin computadoras les atraen mucho más que a los más grandes, ya que son menos abstractas.
- Puede suceder que, por diversas causas, en algunas clases o en todas no haya computadoras (la escuela puede no tener computadoras, los chicos pueden haberlas olvidado, puede no haber luz, etc.).
- El desarrollo de nuevas actividades es complejo o imposible en los entornos propuestos (Scratch, Alice, Pilas Bloques), pero es posible que los docentes quieran desarrollar algunas actividades adicionales o variaciones de las propuestas, y esto siempre es posible si las actividades se realizan sin computadoras.

Cualquiera de las actividades con computadoras podrían, en caso de ser necesario, ser adaptadas para realizarse sin computadoras, siguiendo el ejemplo de la primera actividad, *"El robot humano"*. Esto puede resultar extremadamente útil para aquellas ocasiones en que los estudiantes olvidan las computadoras, o si se corta la luz y no tienen suficiente batería, o situaciones similares, y también porque el docente decide que alguna de las actividades es mejor hacerla sin computadora. Pero para que resulte una experiencia exitosa, debe planificarse con tiempo cómo llevarla a cabo, pues no hay que olvidar que es importante que los estudiantes realicen el proceso de indagación, y que utilicen solamente las herramientas propuestas para cada actividad.

A continuación se dedica una subsección a cada una de las secuencias didácticas del cuaderno.

4.1 Secuencia didáctica 1: Las computadoras hacen todo al pie de la letra

En esta secuencia didáctica se proponen dos actividades: "El robot humano" y "Seamos autómatas". La primera es fundamentalmente realizada por el docente, mientras que la segunda tiene mayor participación de los estudiantes. Estos primeros ejercicios tiene como objetivo comprender algunos puntos centrales de la actividad de programación. Son ejercicios para realizar sin computadoras, de manera de concentrarse en las ideas, y ver que la programación y las computadoras no son necesariamente lo mismo. Los objetivos fundamentales son:

- Entender que la tarea de programar involucra resolver problemas mecánicamente.
- Entender que un programa es un texto que describe una forma de solucionar el problema.
- Entender que una máquina ejecuta el programa para producir la solución.
- Descubrir los comandos primitivos que la máquina puede ejecutar y su combinación para construir el programa.

4.1.1 El robot humano.

Lo mejor para este ejercicio es contar con dos personas: una de ellas oficiará de "máquina" y la otra de "escritor del programa". El docente "máquina" se debe ubicar en una posición fija y pedirles a los estudiantes que le vayan dando instrucciones para salir del aula usando una puerta que se encuentra cerrada. Es importantísimo plantear el problema con precisión; por ejemplo, "deben darme instrucciones para salir del aula por la puerta partiendo desde esta posición".

En un primer momento los estudiantes no saben cuáles instrucciones pueden dar y cuáles no, y suelen empezar con cosas muy generales como "salí del aula". Se recomiendo elegir previamente un conjunto fijo de instrucciones (por ejemplo, "`Avanzar un paso`", "`Avanzar 2 pasos`","`Avanzar 3 pasos`", etc., "`Girar 90 grados a la derecha`", "`Girar 90 grados a la izquierda`", "`Extender el brazo`" y "`Empujar la manija`" suelen ser las más adecuadas), y permitir que por prueba y error los estudiantes las descubran. En caso de que den una instrucción que no está en el conjunto, el docente "máquina" debe decir "No entiendo", y nada más (puede ser que el docente "escriba" decida escribir una o dos de las instrucciones incorrectas, para luego *tacharlas* al comprobarse inválidas, para trabajar la idea de errores al codificar). Si la instrucción es válida (o sea, una de las del conjunto), el docente "máquina" la llevará a cabo (incluso en caso de que NO contribuya a la solución, o no sea posible realizarla – por ejemplo, de tener que avanzar frente a una pared o un obstáculo, intentar avanzar de todas maneras), y en simultáneo el docente "escriba" anotará dicha instrucción en el pizarrón. Al concluir la tarea saliendo efectivamente del aula, se reflexionará sobre el programa que quedó plasmado en el pizarrón. Se puede reforzar la idea de programa colocando a otra persona en la posición inicial y pedirle que realice las instrucciones del programa una por una (o sea, que *ejecute* el programa).

Una de las primeras dificultades del ejercicio tiene que ver con el hecho de que los estudiantes pueden demorar en descubrir las instrucciones válidas, y esto puede producir desmotivación. La intención de esta forma de llevarlo adelante es promover la indagación (recordemos que la didáctica por indagación es uno de los fundamentos conceptuales del curso), y descubrir que hay ciertas instrucciones que la máquina entiende, y que las mismas son un conjunto limitado que debe conocerse de antemano. Si los estudiantes no hacen la pregunta correcta en un tiempo razonable (por ejemplo "¿Cuáles instrucciones se pueden usar?"), orientarlos a hacerse tal pregunta, y darles como respuesta el conjunto predefinido. No debe permitirse que

esta etapa se dilate produciendo mayor desmotivación o frustración, pero sí permitir el suficiente tiempo para que la idea de que si no se conoce el conjunto de instrucciones de antemano, puede resultar muy frustrante intentar programar una computadora.

La siguiente dificultad que puede surgir tiene que ver con la intención de producir un programa general, que pueda sacar al docente desde cualquier lugar del aula, en lugar de hacerlo desde la posición inicial fija. Las herramientas necesarias para escribir un programa general se irán presentando a lo largo del curso, y por lo tanto es importante que en este primer ejercicio la posición inicial sea fija, y el propósito del programa sea "lograr que el docente salga del aula si se encuentra parado en la posición dada, mirando para la dirección dada". Al elegir la posición, intentar que el programa resultante no sea trivial (por ejemplo, que deba realizar un par de combinaciones de girar y avanzar antes de llegar a la puerta), pero que tampoco resulte excesivamente complejo (por ejemplo, tener que recorrer un laberinto de sillas con múltiples giros y avances). Si los estudiantes observan que el programa producido solo sirve para sacar al docente de ESA posición y ninguna más, explicar que por el momento es así, pero más adelante se verán herramientas para lograr mayor generalidad.

La tercera dificultad está vinculada al tema de dar instrucciones de más, imposibles de realizar o que alejan de la solución. En este caso debe permitirse brindar dicha instrucción, y que la misma quede en el programa. Si la instrucción es imposible de realizar, se sugiere intentarlo de todas maneras, para mostrar que las máquinas no eligen si van a realizar o no la instrucción: simplemente lo intentan. Entonces, por ejemplo, si reciben la instrucción de avanzar y hay un pozo, se caerán dentro, y si hay una pared, intentarán avanzar de todas maneras. Esto es parte del proceso de indagación necesario para descubrir el cuidado que debe tener la persona que escribe el programa para que al dar una instrucción la misma sea realizable. Luego de trabajar esta noción, puede permitirse a los estudiantes borrar la instrucción y continuar desde el estado anterior a la misma, para no complicar innecesariamente el resultado final. Si esto no se les ocurriese, puede sugerirse que está permitido hacer correcciones al programa (y en ese caso, el docente "máquina" podría volver a su posición inicial y volver a ejecutar lo escrito hasta el momento, reforzando la idea de cómo se usa el programa).

Si todo resulta de la manera esperada, se concluirá con un programa en el pizarrón. Por ejemplo

```
Girar 90 grados a la derecha
Girar 90 grados a la derecha
Avanzar 3 pasos
Girar 90 grados a la izquierda
Avanzar 7 pasos
Extender el brazo
Empujar la manija
Avanzar 1 paso
```

Una vez concluida esta parte del ejercicio, puede trabajarse un aspecto más dentro del marco conceptual: la idea de estrategia de solución. Al presentarlo de la forma recién mencionada, queda la impresión de que un programa no es más que una secuencia de instrucciones, y que eso es lo único o lo más importante. Sin embargo, como se trabajó en la sección 3.2.2 , un programa es más que una simple secuencia de instrucciones: es una *descripción ejecutable* de una solución a un problema computacional, y por lo tanto es una *entidad dual*, que debe ser utilizada por una máquina para guiar su ejecución, pero también leída por personas para comprender la solución en cuestión. Por ello, la legibilidad del programa es importante, y brindar adecuadamente una guía sobre la estrategia de solución al problema planteado, también. Incluso en un programa tan simple como este puede considerarse la noción de estrategia de solución. Como se mencionó en la sección 3, es recomendación de esta guía que esta idea sea trabajada desde el primer ejercicio y reiterada en cada uno de los ejercicios subsiguientes. En este caso particular, la estrategia de solución podría resumirse en

```
Ubicarse frente a la puerta
Abrir la puerta
Salir
```

pero dado que la máquina no entiende dichas instrucciones, cada una de ellas debe expandirse en los comandos previos vistos con anterioridad

Programa	Estrategia de solución
`Girar 90 grados a la derecha`	
`Girar 90 grados a la derecha`	
`Avanzar 3 pasos`	`Ubicarse frente a la puerta`
`Girar 90 grados a la izquierda`	
`Avanzar 7 pasos`	
`Extender el brazo`	
`Empujar la manija`	`Abrir la puerta`
`Avanzar 1 paso`	`Salir`

o si se elije el otro punto de vista, las instrucciones pueden agruparse según el propósito que cumplen dentro de la estrategia de solución propuesta.

Lo más interesante de pensar la estrategia de solución es que si el punto inicial fuese diferente, la estrategia podría seguir siendo la misma, y solo habría que cambiar la solución de la subtarea `Ubicarse frente a la puerta` para conseguir la solución. Además, como el programa es mucho más simple de entender para una persona que lo lea, resulta consecuentemente más sencillo de modificar. Estos son valores importantes a destacar, puesto que hacen a la naturaleza misma de la tarea de programar.

Para concluir con este ejercicio, es importante destacar que a pesar de parecer sencillo y ser el primero, las sutilezas y complejidades que pueden surgir merecen atención anticipada, y resulta fundamental manejarlas adecuadamente para cumplir el propósito de contribuir a los objetivos del curso.

4.1.2 Seamos autómatas

El segundo ejercicio dentro de esta actividad también se lleva adelante sin utilizar computadoras. Es una actividad para realizar en dos o tres etapas (según el tiempo disponible). Para esta actividad los estudiantes deben conformar grupos de dos personas.

En la primera etapa, cada grupo debe diseñar tres cosas:

- un conjunto de comandos simples que puedan ser realizados por una persona,
- una tarea que pueda realizarse mecánicamente con esos comandos,
- un contexto inicial para realizar la tarea.

Para tener una idea de lo que se espera, es interesante considerar un ejemplo (que puede usarse en clase luego de que hayan realizado la actividad, complementando a los que se presenten como interesantes). Se provee entonces:

- conjunto de comandos:
  - `Cruzar los brazos`
  - `Estirar los brazos`
  - `Decir "¡HEY!"`
  - `Levantar rodilla izq/der`
  - `Estirar pierna izq/der`

  - `Bajar pierna izq/der`

- tarea: escribir un programa que indique cómo bailar polka.
- contexto inicial: parado con los brazos al lado del cuerpo.

Es importante recordarles que identifiquen a su grupo en la hoja donde escriben la solución a la primera etapa, con el objetivo de que luego la segunda etapa sea más fácil de completar.

En la segunda etapa, cada grupo intercambia con otro los elementos elegidos, y confecciona un programa que solucione la tarea desde el contexto inicial dado, utilizando los comandos primitivos provistos. Es importante que cada grupo utilice solamente lo provisto, y no modifique ninguna de los elementos dados. En caso que falten o sea imposible realizarlo, deben informar dicha situación y explicar por qué sucede. Por ejemplo, continuando con el ejercicio anterior, la solución de un segundo grupo podría ser:

Programa	Estrategia de solución

`Cruzar los brazos`
`Levantar rodilla izq`
`Estirar pierna izq`
`Estirar los brazos`
`Decir "¡HEY!"`
	`Paso de polka, pierna izq`

`Cruzar los brazos`
`Levantar rodilla der`
`Estirar pierna der`
`Estirar los brazos`
`Decir "¡HEY!"`
	`Paso de polka, pierna der`

...	...

Una tercera etapa opcional es que un tercer grupo (distinto de los dos que ya trabajaron sobre el ejemplo) tome la solución provista por otro de los grupos e intente ejecutar el programa. Esto puede hacerse en el frente como puesta en común, y de esa manera servir para discutir los problemas o situaciones que se hayan encontrado. Esta etapa puede reemplazarse por una puesta en común donde es el docente el que elije algunas de las soluciones y las ejecuta.

En esta actividad pueden surgir un montón de problemas y dudas, lo cual es extremadamente valioso como parte de la didáctica de indagación.

El primero de los problemas está asociado con la elección de la tarea a realizar. Si la tarea es demasiado compleja, requiere pasos demasiado minuciosos o tiene requerimientos de obtener información del entorno, el grupo tendrá dificultades para completar la tarea. Es usual que muchos de los grupos tomen por imitación algún desplazamiento dentro de la habitación, al que le agregan alguna dificultad adicional (prender una luz, tomar un vaso de agua, escribir una línea en el pizarrón); sin embargo, otros grupos suelen ser muy creativos (una propuesta muy interesante que dieron en uno de los cursos fue "desde una posición dada –muy simple– del cubo mágico, completar el armado de todas las caras; observar que si la tarea hubiese sido armar el cubo desde cualquier posición, el trabajo de programación habría sido terriblemente complejo). Una manera de aliviar este problema es explicarles que las tareas deben ser sencillas, deben ser realizables con instrucciones simples y debe estar claro cuál es la posición inicial de todos los elementos **antes** de comenzar. Sin embargo, debe balancearse la cantidad de ayuda para que el proceso de indagación pueda realmente tener lugar.

El siguiente problema está asociado con la elección de los comandos. Una primera cuestión que suele aparecer, y que debe dejarse que suceda para luego explicarlo en la puesta en común, es que aparecen comandos de diferente nivel de complejidad en el conjunto básico. Por ejemplo, al mismo tiempo hay comandos de movimiento preciso de las manos ("`Estirar los`

`dedos`", "`Separar los dedos`")y otros mucho más complejos ("`Agarrar el vaso`", "`Escribir 'Hola'`") en el mismo conjunto. Si bien esto no está mal *per se*, puede reflexionarse sobre en qué nivel de detalle trabaja la máquina que debe programarse, y explicar que es usual que los comandos primitivos sean todos más o menos del mismo nivel de complejidad. Una segunda cuestión muy común es que los conjuntos de comandos primitivos sean incompletos (por ejemplo, hay un comando para "`Estirar el brazo`", pero no hay uno para "`Doblar el brazo`", a pesar de que la solución requiera de ambos). En este caso pueden tomarse dos caminos: la mayoría de los grupos decidirán completar el conjunto de comandos agregando el comando que falta; sin embargo, un enfoque más realista que algunos grupos pueden tomar (sobre todo si hay estudiantes con alguna experiencia de programación) es simplemente decir que con esos comandos es imposible realizar la tarea. Esta segunda forma es claramente más realista, porque si la máquina que debe programarse carece de la capacidad de ejecutar una instrucción (por no tenerla en su conjunto base), no resulta sencillo modificarla para que sí lo haga. Si este problema aparece, puede dar lugar a reflexionar sobre la naturaleza de las máquinas, y sobre la importancia de que el conjunto de comandos primitivos sea suficientemente flexible y completo como para llevar adelante las tareas necesarias. Una tercera cuestión mucho más sutil tiene que ver con trivializar tanto el conjunto de comandos como la tarea a realizar. Es común ver algunas soluciones donde la tarea es extremadamente sencilla, y el conjunto de comandos es prácticamente, la solución del problema. Por ejemplo, si la tarea fuera "Colocarse un par de anteojos, suponiendo que uno está sentado delante de la mesa, con los anteojos sobre ella", y los comandos fueran "`Estirar el brazo`", "`Tomar los anteojos`", "`Doblar el brazo`", "`Colocar los anteojos sobre la nariz`", el programa a realizar es demasiado obvio y trivial. Nuevamente, esto no está mal *per se*, pero trivializa demasiado el problema y dificulta la comprensión buscada sobre la naturaleza de los programas. A partir de esta situación se puede reflexionar sobre el hecho usual en programación donde cada comando suele ser necesario más de una vez, y con diferentes objetivos, y donde la combinación de comandos es usualmente no trivial. La última cuestión a mencionar vinculada con los comandos tiene que ver con tener comandos redundantes, que podrían expresarse por combinación de comandos más simples. Por ejemplo, tener al mismo tiempo "`Subir la rodilla`", "`Adelantar el pie`" y también "`Estirar la pierna hacia adelante`": el último de los comandos es expresable como combinación de los dos primeros, volviéndolo redundante.

Un tercer grupo de problemas se presenta con los programas realizados por los grupos en la segunda etapa, y también con la ejecución realizada en la tercera etapa. Es extremadamente común que al ejecutar el programa las personas 'completen' detalles faltantes, omisiones de comandos, etc. para conseguir resolver la tarea pedida a pesar de que el programa no esté bien confeccionado. Esto da lugar a reflexionar sobre la naturaleza dual de los programas: cuando el programa es procesado por la máquina, la misma lo hace con gran precisión y sin completar ningún detalle, pues *no existe* una *comprensión* sobre la tarea a realizar; en cambio cuando las personas leen el programa, completan los detalles necesarios a pesar de que no existan, pues sí comprenden el propósito del programa pero no son buenos mecanismos de ejecución, por su falta de precisión. Es sugerencia de esta guía que las personas eviten ejecutar los programas paso a paso, comprendiendo mejor los propósitos y herramientas utilizadas para la construcción de programas, dejando la ejecución para las máquinas. Históricamente la enseñanza de la programación ha tomado el camino contrario, enseñando a ejecutar los programas paso a paso, suponiendo que de esta forma se comprende mejor el funcionamiento de los mismos. Sin embargo, en la actualidad la tecnología de compilación y ejecución es tan sofisticada que esta forma de enseñanza limita en el mediano plazo en lugar de ayudar.

El último tema a considerar antes de terminar con el ejercicio tiene que ver con la legibilidad de los programas producidos. Es poco usual que los grupos expliciten su estrategia de solución al confeccionar el programa. Así, al llevar adelante la tercera etapa, puede darse el caso de que el grupo que deba ejecutar no sepa qué problema está resolviendo y consecuentemente no pueda completar detalles o malinterprete los comandos primitivos (esto vinculado al punto del

párrafo anterior). En este caso debe presentarse la idea de *legibilidad*, y destacar la importancia de contar con una buena *estrategia de solución* al confeccionar un programa, así como la de expresarla en el código resultante.

Esta primera secuencia didáctica puede parecer sencilla en una lectura superficial, pero si se la lleva adelante de manera precisa y completa, es extremadamente enriquecedora, pues los estudiantes realmente empiezan a sentir el poder de la didáctica por indagación, y se plantean un montón de cuestiones que hacen a la naturaleza de los programas y de la actividad de programación. Es importante observar que ya desde esta primera actividad debe trabajarse dentro del marco conceptual presentado en la sección 3, en los primeros temas de la presentación cronológica En caso de haber decidido explicitar el marco conceptual (por ejemplo, por tratarse de un curso para formadores), este es un buen momento para comenzar a hacerlo, agregando las categorías de *didáctica* por indagación (3.1) y *herramientas conceptuales* y dentro de esta última la noción de *estrategia de solución* (3.2.1), remarcando el hecho de que son los conceptos más importantes del curso, y también la categoría de *herramientas del lenguaje* (3.3), junto con la subcategoría *comandos* y dentro de ella, *comandos primitivos* y *secuencia de comandos* (3.3.1). Es buena idea utilizar este marco conceptual, y los puntos conceptuales mencionados, para enmarcar el ejercicio dentro del resto del curso.

4.2 Secuencia didáctica 2: Presentación de Scratch/Pilas Bloques

La segunda secuencia didáctica está destinada a presentar el entorno que se utilizará en la mayoría de las restantes actividades, y también a reforzar los conceptos presentados en la secuencia anterior. En el cuaderno, la propuesta es utilizar el entorno Scratch; sin embargo, con posterioridad, la Fundación Sadosky creó el entorno Pilas Bloques, que puede ser usado con el mismo propósito aunque con menos elementos disruptivos y más focalizada en los objetivos didácticos y en los conceptos a trabajar.

4.2.1 El entorno: Scratch o Pilas Bloques

En esta sección se discuten las características y dificultades de utilizar Scratch, para aquellos docentes que elijan utilizarla, aunque se recomienda fuertemente la utilización de Pilas Bloques en su lugar. En cada caso, se contrastarán las características de ambos entornos para clarificar sus similaridades y diferencias.

La primera observación al respecto de Scratch es que se trata de un entorno general producido por el MIT, cuyo objetivo es servir para que un público amplio sin experiencia en programación se interese y aprenda programación. Sin embargo, también tiene como objetivo que se puedan realizar todo tipo de programas, y que se puedan realizar tareas complejas en él[6] y por ello contiene muchísimos elementos y combinaciones que pueden confundir o distraer del objetivo de comenzar a entender las ideas básicas de programación. La propuesta del curso que aquí se discute es concentrarse en una serie de actividades que promuevan la indagación de ciertos conceptos fundamentales, englobados dentro del marco conceptual presentado. Es por ello que no es correcto afirmar que en este curso se enseña Scratch, sino que debería afirmarse que se enseñan ciertos conceptos de programación usando Scratch. El entorno Pilas Bloques, por otra parte, fue pensado para contener exclusivamente las actividades del curso, sin objetivos complementarios y por ende, sin elementos distractivos y conteniendo varias mejoras que permiten orientar la didáctica en el sentido deseado.

Tanto Scratch como Pilas Bloques pueden ser utilizados desde la web o sin conexión. Sin embargo, la instalación de Scratch para su uso sin conexión es bastante más compleja que la instalación de Pilas Bloques. En caso de planificar su utilización *offline*, se recomienda dedicar

[6] Según fuera establecido por un miembro del Equipo de Scratch en sus foros de discusión, ver `https://scratch.mit.edu/discuss/topic/245/`.

tiempo específico de la clase a que todos tengan los entornos instaladas. Los errores más comunes que encontramos son los siguientes:

- Las actividades solamente funcionan con la versión 2.0. Si los docentes tienen la versión 1.0 de Scratch ya instaladas en sus computadoras, no van a poder ejecutar ninguna de las actividades. Se recomienda, entonces, verificar con anticipación que todos tienen la versión correcta y operativa del entorno.

- Para poder instalar Scratch, se precisa contar primero con un software de soporte llamado Adobe Air. Es importante que al distribuir los instaladores esté la versión adecuada de este software. Con la versión 18.0 funciona; otras deberían probarse con anticipación para mayor seguridad.

- Las actividades se distribuyen de manera independiente al entorno Scratch, por no ser parte integral de la misma. Es importante asegurarse que todos los docentes cuentan con versiones originales en funcionamiento, ya que las actividades pueden ser modificadas por los estudiantes, y si utilizan una versión diferente pueden producirse resultados inesperados. Además, es conveniente que construyan un *atajo* a la carpeta que contiene las actividades en el directorio por defecto, para no tener que buscarlo cada vez que se quiere abrir una actividad.

Dos temas que pueden resultar una traba importante son el idioma y el tamaño de la pantalla. Al comenzar, Scratch puede que empiece a funcionar en inglés. En ese caso el cuaderno da instrucciones de cómo cambiar al castellano. En cambio Pilas Bloques arranca directamente en castellano. Con respecto al tamaño del escenario, si es muy grande deja prácticamente sin espacio para la confección de programas; para ello, se puede utilizar la opción `Escenario pequeño` del menú `Editar` (o de manera equivalente cliquear en la flechita que separa el escenario del sector de programa). En Pilas Bloques también puede resulta inconveniente el tamaño, haciendo que ciertos textos no sean visibles; en ese caso, en el menú de `Opciones` (el engranaje de arriba a la derecha) puede cambiarse el porcentaje de visualización de los gráficos.

La mayor dificultad de Scratch proviene del hecho de que cuenta con diez secciones diferentes con herramientas de programación, de las cuales se utilizarán en cada actividad entre una y cinco. Al comenzar, siempre arranca en la sección `Movimiento` que no se utilizará nunca. Debe explicarse con cuidado que lo primero que debe realizarse al iniciar una actividad es cambiar a la sección `Más Bloques`, que es donde residen los comandos primitivos. Además, en cada actividad habrá que aclarar cuáles son las herramientas que están permitidas y cuáles no lo están (por ejemplo, en la actividad de motivación de la repetición se espera que no se utilice repetición, justamente para motivar su utilización). Esto resulta molesto y confuso, pues los estudiantes, especialmente los más jóvenes, sienten una curiosidad natural por explorar todas las secciones, y si bien es interesante hacerlo, los aleja de los objetivos del curso. En cambio en Pilas Bloques solo las secciones que se utilizan en cada actividad aparecen activas (además de tener una mejor clasificación, acorde con el marco conceptual aquí propuesto).

Otra dificultad, aunque no tan importante, proviene del hecho de que en las actividades de Scratch no hay ninguna guía respecto de qué tarea debe resolverse, y el entorno tampoco contempla ningún criterio de éxito. Por eso es necesario que la tarea se explique de manera verbal en el pizarrón (suele ser muy útil contar con alguna transparencia que posea el enunciado de la actividad) y que luego se revise que todos están cumpliendo el objetivo de manera adecuada. Hasta que los estudiantes se acostumbran a esta forma, pueden surgir situaciones de confusión donde ellos no entienden qué se espera que hagan provocando desazón y desmotivación, por lo que se recomienda ser extremadamente cuidadosos y claros con la consigna de cada actividad.

Una dificultad más, que puede resultar molesta en ocasiones, tiene que ver con la capacidad de revisar soluciones previamente dadas. Para que esto sea posible se recomienda una de dos alternativas. La primera es realizar una copia de la carpeta `ActividadesScratch`, y grabar en la copia, lo que resulta en una opción es más sencilla, aunque es complicado hacer más de una

variante por ejercicio; se recomienda esta forma especialmente en cursos con chicos chicos, o con poco manejo de carpetas y archivos. La otra alternativa consiste en salvar cada una de las soluciones que realizan mediante el menú `Archivo`→`GuardarComo`, y que lo hagan en una carpeta específica de su elección creada especialmente para guardar sus soluciones. Hay dos cuestiones a tener en cuenta para que esto no se transforme en un problema.

- En Scratch si están utilizando la carpeta original de actividades y no una copia, no deben utilizar la opción `Guardar`, ya que si lo hacen estarán sobrescribiendo la actividad original, y no podrán volver a comenzarla desde cero (por ejemplo, para probar alternativas de solución).

- Se recomienda, si el curso tiene suficiente manejo de archivos, que creen una carpeta específica en su zona de trabajo para guardar las soluciones y que las mismas sean nombradas adecuadamente, porque al aumentar el número de actividades realizadas y de variantes hechas, puede resultar complejo encontrar una solución específica que se quiere revisar. Si el curso, en cambio, es de chicos pequeños o con poco manejo de archivos, se recomienda crear una copia de la carpeta de actividades, así las originales se preservan en otro lado.

La última de las dificultades, presente en ambos entornos, tiene que ver con la forma de escribir el programa. Para que el mismo funcione, los comandos (bloques, en la terminología utilizada en estos entornos) deben agregarse al bloque inicial, fijo, que ya se encuentra en el área de programa: `Al recibir "empezar"` en Scratch, y `Al empezar a ejecutar` en Pilas Bloques. Suele suceder que los estudiantes no enganchan adecuadamente los bloques a este bloque inicial, y consecuentemente el programa no hace nada. Algunos estudiantes piensan que por colocar bloques en el área de programa, aún sin engancharlos, los mismos deberían funcionar; es bueno si indagan hasta entender cómo funciona, pero debe tenerse cuidado con la desmotivación que puede surgir si no encuentran la solución luego de un tiempo.

4.2.2 El gato en la calle

Habiendo considerado las características y dificultades del entorno a utilizar, es tiempo de dedicar atención a la actividad en sí. El cuaderno propone la actividad denominada *"El gato en la calle"*, la cual consiste en escribir un programa para lograr que el gato realice algunas acciones sencillas. En ambos entornos encontramos la misma actividad; además, en Pilas Bloques hay una actividad adicional (previa): *"El alien toca el botón"*, que es mucho más sencilla por tener un único objetivo específico y solo contar con comandos primitivos como herramientas.

En *"El gato en la calle"*, luego de experimentar con los comandos primitivos, se puede establecer la primera consigna: lograr que el gato avance, se acueste a dormir, se levante, salude y vuelva al inicio. Este ejercicio busca presentar la idea de procedimientos (o en la terminología de Scratch, bloques definidos), según explica el cuaderno. Al explicar la solución es posible que algunos de los estudiantes manifiesten que no ven la necesidad de contar con un procedimiento `Dormirse`, puesto que el mismo resulta extremadamente sencillo. Aquí es donde puede trabajarse la idea de programa como forma de comunicar una idea, y evaluar la diferencia entre utilizar un procedimiento denominado `Dormirse` versus la secuencia de comandos individuales que lo implementan. Claramente en el segundo caso es más complicado observar la idea de la persona que escribió el programa de enviar el gato a dormir; además, puede pedirse una variante del ejercicio donde el gato deba dormirse y despertarse varias veces, observando así la utilidad de los procedimientos para evitar repetir la confección de soluciones previamente desarrolladas. Es importante presentar la idea de procedimiento en este ejercicio como preparación para la próxima secuencia didáctica. En Pilas Bloques existe además el comando primitivo `Soñar`, que puede utilizare para reforzar la idea de que el gato está durmiendo.

Con respecto al marco conceptual, además de reforzar los conceptos de *didáctica por indagación* y de *estrategia de solución* (3.1 y 3.2.1, lo que se hará en todas y cada una de las

actividades) y el de *comando primitivo* (3.3.1), se puede comenzar a trabajar con el concepto de legibilidad y de elección de nombres (agregando la herramienta *legibilidad* a la categoría de *herramientas conceptuales* si es está explicitando el marco, 3.2.2) y el de procedimientos (agregando la herramienta *procedimientos* a la categoría de *herramientas del lenguaje* dentro de *comandos*).

4.3 Secuencia didáctica 3: Presentación de Lightbot

En esta secuencia didáctica se propone el uso de un entorno especial, llamado Lightbot, que se utilizará por única vez en el curso. El objetivo al utilizar este entorno es múltiple: por un lado, provee una excelente motivación en los estudiantes, por su planteo en formato de juego y su progresión en complejidad de tipo tutorial; por otro lado, las actividades de la versión propuesta están organizadas de manera tal que los conceptos de estrategia de solución y de procedimientos son imprescindibles para resolver algunos de los niveles, sirviendo como claro ejemplo de su importancia; finalmente, porque manejar diferentes entornos de programación permite ver que los conceptos y herramientas que se trabajan en el curso (esto es, estrategia de solución, división en subtareas, etc.) son transversales a diferentes lenguajes y entornos de programación, y permite también que los mismos sean aprehendidos con mayor solidez.

Sin embargo, este entorno no fue pensada para servir como vehículo de aprendizaje, y por ello tiene varias falencias que deben ser tenidas en cuenta por el docente. La primera dificultad, especialmente con estudiantes muy jóvenes, es que el entorno está en inglés, y no existe versión en castellano. Sin embargo, es mayormente visual y gráfico, por lo que esto no es tan grave como podría pensarse; los estudiantes se adaptan rápidamente al mismo. La siguiente dificultad es que no se puede elegir en qué nivel comenzar: siempre debe comenzarse desde el nivel 1, y resolver todos los niveles intermedios hasta llegar al nivel que se quiere resolver. Por eso es que se recomienda utilizarla una única vez, resolviendo todos los niveles en secuencia. La tercera dificultad es que una vez terminado un nivel, se pasa automáticamente al siguiente, y no puede volverse atrás. Esto es poco útil si se trata de revisar la solución del estudiante, considerar alternativas, etc. Por ello, se recomienda pedirles a los estudiantes que siempre dejen la última luz sin prender, para poder mostrarle al docente la solución, y considerar alternativas. Otra dificultad tiene que ver con la cantidad limitada de procedimientos (2) y la imposibilidad de darles nombre a los mismos, lo que impide trabajar el concepto de legibilidad del programa. La última dificultad digna de mencionarse es que en algunos niveles es necesario utilizar comandos "de más" (por ejemplo, indicar prender una luz donde no hay luz, o saltar donde no hay nada para saltar), para poder lograr encontrar una estrategia de solución que uniformice todos los casos. Esto puede evitarse si no se trabaja más allá del nivel 10 (que es el último que plantea un verdadero desafío desde el punto de vista de este curso). Resumiendo, los niveles 7, 8 y 9 son los verdaderamente interesantes, puesto que en el 7 se presenta la idea de estrategia de solución expresada con procedimientos, y en los niveles 8 y 9 se practica esa idea; el nivel 10 también practica esa idea pero es mucho más difícil de resolver, y los niveles siguientes requieren trucos que no aportan desde el punto vista conceptual.

Existen otras versiones de Lightbot, y algunas incluso solucionan algunos de los problemas aquí mencionados. Sin embargo, en este curso preferimos utilizar esta versión de Lightbot (la versión 1.0), porque sus niveles presentan los conceptos que se busca transmitir en este curso con mucha mayor precisión que en los niveles de las otras versiones. Además, las otras versiones también incorporan comandos adicionales y otros conceptos que este curso maneja con posterioridad. Es por ello que, a pesar de las dificultades mencionadas, el curso sugiere trabajar con la versión 1.0, y quizás sugerirles a los estudiantes que existen otras versiones luego de haber terminado esta secuencia didáctica, para que los interesados puedan explorarlo con posterioridad.

El entorno puede utilizarse desde un navegador conectado a Internet, o (en Windows) como aplicación de consola. En caso de utilizarlo como aplicación de consola, debe tenerse en cuenta

que la pantalla inicial muestra una opción de *"Cargando..."* (*"Loading..."*) que nunca progresa; en realidad esa opción es innecesaria y puede hacerse click inmediatamente sobre el botón de *"Jugar"* (*"Play"*). Muchos estudiantes se quedan esperando a que termine de cargar, y esto puede demorar un rato la clase, por lo que es conveniente alertarlos para evitar este tipo de demoras.

Los niveles del 1 al 5 son muy sencillos, y están pensados para familiarizar al estudiante con el entorno (los 3 primeros, por ejemplo, iluminan los comandos necesarios que son nuevos, a modo de tutorial). Recién en el nivel 6 se plantea la necesidad de utilizar lo que en este entorno se denomina "*funciones*", pero que en realidad son procedimientos, según la denominación utilizada en esta guía y en el cuaderno. En este nivel todavía no es imprescindible utilizar una estrategia de solución para la división en subtareas, aunque siguiendo el marco conceptual propuesto, puede resultar interesante discutir sobre una buena estrategia. En los niveles 7, 8 y 9, tener alguna estrategia de solución es fundamental para poder resolver el nivel. Y en el nivel 10, se precisa una estrategia de solución óptima, por lo que este nivel es que el plantea mayores desafíos.

El mayor problema que los estudiantes encuentran es que les cuesta visualizar su estrategia de solución plasmada en procedimientos cuyos nombres son simplemente `f1` y `f2`. Esto puede resultar frustrante para algunos, y es conveniente que los docentes estén atentos para ayudar cuando esto sucede. En las páginas 20 a 29 del cuaderno se ofrece una guía precisa de aquellos puntos que han sido comprobados como los más problemáticos, y se sugiere cómo ayudar a los estudiantes. Debe recordarse que la propuesta del curso es utilizar el gran valor de la dinámica interactiva de los ejercicios, siguiendo lo que dicta el aprendizaje por indagación, en lugar de una forma más tradicional donde se plantea una pregunta que el alumno ya está en condiciones de resolver en función de lo explicado. Resulta extremadamente satisfactorio plantear (en forma controlada) cuestiones que los alumnos no puedan resolver, como forma de estimular su curiosidad, evitando la noción de ejercicio como estructura estanca posterior a la clase magistral y que solo sirve para ejercitar lo visto allí.

En el caso del nivel 10, el cuaderno sugiere la estrategia de solución óptima: utilizar un procedimiento para avanzar hasta la "entrada" de la estructura, luego encender la luz de la izquierda, volviendo fácilmente a la situación inicial y entonces reutilizando el procedimiento para volver a la "entrada" y desde ahí prender la luz derecha. El cuaderno no lo sugiere, pero puede resultar valioso discutir otras estrategias posibles, y por qué las mismas no servirían para solucionar el problema en este juego (por la falta de recursos). Por ejemplo, una estrategia alternativa es avanzar a la "entrada" y prender primero la luz de la derecha, y recién luego la de la izquierda; sin embargo en este caso no es sencillo volver a la situación inicial para reutilizar la tarea de ir hasta la "entrada", y entonces se precisan más recursos de los disponibles.

Como cierre de la actividad el cuaderno sugiere motivar a los estudiantes a que comparen Scratch con Lightbot (y aquí se agrega también con la primera actividad) desde el punto de vista de los conceptos vistos: programa como solución a un problema, comandos primitivos, secuencia, procedimientos, estrategia de solución, legibilidad, etc. Es importante para el trabajo que esta parte no sea salteada, pues es el momento donde los conceptos discutidos en el marco conceptual empiezan a ser visibilizados e incorporados.

Con respecto al marco conceptual, esta actividad contribuye a profundizar los conceptos de didáctica por indagación (3.1) y estrategia de solución (3.2.1), al tiempo que muestra la importancia de la legibilidad (3.2.2) cuando la misma no puede conseguirse.

4.4 Secuencia didáctica 4: Repetición simple

En esta secuencia de didáctica se trabaja nuevamente con Scratch, a través de la actividad *"No me canso de saltar"*, en la cual se trata de lograr que el gato salte 30 veces. La idea es primero probar solamente con las herramientas vistas hasta el momento: comandos primitivos (que en esta actividad es solamente `Saltar`) y procedimientos. Existen dos soluciones posibles utilizando estas herramientas: la que usa solamente comandos primitivos y usa 30 comandos

`Saltar` en secuencia (la más sencilla), y otra que utiliza procedimientos para, por ejemplo, `SaltarDiezVeces`, y luego utiliza 3 llamados a este procedimiento. Esta última forma es muy interesante, y muestra que los estudiantes que la hayan pensado entendieron muy bien la idea de procedimiento, por lo que no debe desalentarse.

Una vez que los estudiantes probaron alguna de estas 2 soluciones, y especialmente si no encontraron otra solución más que la secuenciación de 30 comandos, se procede a presentar la *repetición*, que es una forma de combinar comandos para describir acciones más complejas. A este tipo de combinaciones se las conoce con el nombre de *estructuras de control*, pues permiten controlar la ejecución del programa de maneras diferentes a la secuencia; por ello, tanto en Scratch como en Pilas Bloques se pueden encontrar en la sección `Control`.

Utilizando repetición, la solución más sencilla consiste en pedir la repetición 30 veces de un único comando `Saltar`, y este bloque colocarlo directamente en el bloque inicial (`Al recibir "empezar"`, en Scratch). Sin embargo, con la intención de reforzar la idea de estrategia de solución y también la idea de programa como medio de comunicación de ideas, como ya se adelantó en la sección 3.2.1, puede resultar interesante que en cada ejercicio se defina un procedimiento inicial, que sirva para resolver la tarea, y nombrarlo adecuadamente. En este ejercicio sería `SaltarTreintaVeces`, que haría una repetición de 30 veces de `Saltar`, o podría hacer una repetición de 3 veces de `SaltarDiezVeces`, el cual a su vez haría una repetición de 10 veces de `Saltar`. El cuaderno ofrece la solución donde la repetición se coloca directamente en el bloque inicial, pero esta guía sugiere utilizar siempre un procedimiento como forma de reforzar tanto la idea de legibilidad del programa como la idea de procedimientos.

La repetición simple requiere utilizar un número para indicar la cantidad de repetición. Este hecho puede tomarse como algo natural, o en los cursos que lo ameriten, puede explicitarse que los números pertenecen a una subcategoría diferente a los comandos dentro de las herramientas del lenguaje: las expresiones. También puede decirse que los números utilizados son las expresiones más simples: expresiones que describen valores de forma literal.

Con respecto al marco conceptual, debe agregarse a la categoría *herramientas del lenguaje*, subcategoría *comandos*, la herramienta *repetición simple* (3.3.1), y también agregar la subcategoría *expresiones* con el concepto de *valores*. Al mismo tiempo, continuar reforzando los conceptos fundamentales de didáctica por indagación, estrategia de solución, legibilidad y definición y uso de procedimientos (3.1, 3.2.1, 3.2.2 y 3.3.1).

4.5 Secuencia didáctica 5: Programación en papel cuadriculado

Esta secuencia didáctica propone un modelo de ejecución de programas que combina el espíritu de la primera secuencia didáctica con las características de Lightbot, y permite profundizar otra manera de formular instrucciones y programas, comprender las dificultades asociadas a ejecutar programas, y trabajar mucho más con la idea de legibilidad de un programa. Es importante que para esta actividad todos los estudiantes cuenten con un par de hojas de papel cuadriculado; se recomienda a los docentes contar con un block de estas hojas para que incluso los estudiantes que no las tengan puedan realizar la actividad.

La dificultad más grande de la actividad radica en que toma mucho tiempo que los estudiantes entiendan la manera precisa de dar instrucciones, especialmente la definición y uso de procedimientos junto con una adecuada elección de nombres para los mismos. Es importante detenerse en este aspecto, porque los procedimientos y la legibilidad son conceptos fundamentales que se busca trabajar durante el curso. Luego de haber explicado el modelo de programación y ejecución, de haberles hecho practicar realizando algunos dibujos propuestos, y de haber explicado la solución de uno de ellos comparando una versión que solo tenga comandos básicos contra una que utilice adecuadamente procedimientos, se puede pasar a la parte más enriquecedora de la secuencia: cada estudiante pensará un dibujo, lo programará, e intercambiará su programa con otro, para ejecutarlo y ver si el programa es entendible y si al ejecutarlo efectivamente hace lo esperado.

Un error común que comenten los estudiantes en esta parte, y que debe remarcarse como inadecuado, es tratar de que su programa no se entienda. O sea, intentan que el programa se transforme en un acertijo, en lugar de tratar de que el mismo sea fácilmente entendible aún sin ejecutarlo. Puesto que uno de los objetivos primordiales del curso es transmitir la idea de que un programa es una forma de comunicación, esto debe ser explicado con detalle, y valorar especialmente aquellas soluciones que son entendibles aún sin ejecutar. Por ejemplo, comparar este programa que no utiliza procedimientos

(↯←)7 (↯→↑)7 (↯↓)7 →→ (↯↑)6 →↑ (↯→↓↓)3 (↯←)4
↓↓ (↯→)7 (↯↓←)2 (←↯)13 (↑←↯)2 (→↯)9

contra este otro que sí los utiliza

```
[Dibujar velero]

"Dibujar velero"
    [Dibujar vela izq]
    →→
    [Dibujar vela der]
    ↓↓
    [Dibujar casco]

"Dibujar vela der"
    (↯↑)6 →↑ (↯→↓↓)3 (↯←)4

"Dibujar casco"
    (↯→)7 (↯↓←)2
    (←↯)13 (↑←↯)2 (→↯)9

"Dibujar vela izq"
    (↯←)7 (↯→↑)7 (↯↓)7
```

Resulta imposible determinar qué dibujo realiza el primer programa sin ejecutarlo, mientras que en el 2do caso podemos predecir que el programa fue pensado para dibujar un velero. Esta capacidad de leer el programa tiene una ventaja adicional que puede discutirse al realizar la puesta en común: si el programa contuviese algún error (por ejemplo, por haber omitido un movimiento, o haberlo realizado en un sentido incorrecto), en el 1er caso resulta imposible entender cuál es la corrección a realizar, mientras que en el 2do caso resulta muy simple de encontrar el error. Lo mismo sucedería si queremos modificar alguna de las partes del dibujo, por ejemplo, haciendo las velas más altas, o el casco más largo o profundo.

Este ejercicio es extremadamente importante si se lo utiliza de la manera aquí expuesta, pues permite reforzar aún más los conceptos de estrategia de solución (3.2.1), legibilidad (3.2.2) y el uso de comandos primitivos, secuencia y repetición simple (3.3.1). Además permite profundizar en la diferencia que existe entre la ejecución mecánica de un programa y la idea que busca expresar, y en la diferente adecuación de las máquinas y las personas para realizar la ejecución o entender la idea, respectivamente.

4.6 Secuencia didáctica 6: Repetición simple (II)

La secuencia didáctica número 6 plantea una actividad principal, *"El marciano en el desierto"*, y luego una serie de actividades de ejercitación, a saber *"Lightbot en Scratch"* (llamado *"Tito*

enciende las luces" en Pilas Bloques), "*El recolector de estrellas*", "*María, la come sandías*", "*Alimentando a los peces*", "*Instalando juegos*", "*La gran aventura del mar encantado*" y finalmente, "*Reparando la nave*"; además, en Pilas Bloques hay una actividad más, "*El alien y las tuercas*".

Las actividades que se sugiere sean realizadas intercalando puestas en común son las 3 primeras, puesto que cada una de ellas ofrece alguna innovación desde el punto de vista de estrategias de solución: la primera sirve para ver variaciones a la hora de elegir los procedimientos, y cómo decidir cuál es la mejor, la segunda permite varias estrategias de solución diferentes, y la tercera, debido a la limitación impuesta por los comandos primitivos, obliga a utilizar una única estrategia. Luego las demás actividades se pueden dejar como ejercitación libre, supervisando a los estudiantes de manera individual, y trabajando con los que tienen mayores dificultades para expresar su estrategia de solución o elegir nombres para sus procedimientos.

En el cuaderno, en todas las soluciones ofrecidas para cada una de las actividades, se expresa la solución mediante varios comandos a partir del bloque inicial (`Al recibir 'empezar'`). Si bien esto es correcto, hay una manera mucho más acorde con el objetivo de transmitir los conceptos de estrategia de solución, procedimientos y el de legibilidad de un programa. Esta forma consiste en definir, como primera tarea, un procedimiento cuyo nombre exprese la solución a definir, y donde se coloquen los comandos que expresan la solución. De esta forma, en el bloque inicial solo haría falta utilizar este procedimiento. Por ejemplo, en el caso de "*El marciano en el desierto*", se definiría un bloque `Comer todas en sentido antihorario`, donde se colocarían los demás procedimientos que expresan la solución (`Comer fila inferior`, `Comer columna derecha`, etcétera). El hecho de tener que definir como primera tarea un procedimiento que exprese en su nombre la solución buscada motiva a varias cosas importantes:

- pensar una estrategia de solución,
- pensar un nombre adecuado para esa estrategia, y
- practicar el uso de procedimientos como medio de expresar ideas y aportar legibilidad al programa.

Otro tema a tener en cuenta es que algunos de los nombres de procedimientos elegidos en el cuaderno pueden mejorarse para hacerlos más descriptivos y adecuados al objetivo de hacer que el programa resulte legible y comunicativo. Es importante tener en cuenta el consejo de que los nombres de comandos y procedimientos son mucho más adecuados si utilizan un verbo, y también si describen de manera breve la estrategia de solución elegida. Por ejemplo, comparar el nombre `Comer 2 derecha` con `Comer fila inferior`, para una de las subtareas de "*El marciano en el desierto*"; el primero es que el se utiliza en el cuaderno, y el segundo es el que se sugiere en esta guía como mejora, teniendo en cuenta los conceptos a transmitir. Otras sugerencias de mejoras se ofrecen en la siguiente tabla

Actividad	**Nombre utilizado**	**Nombre sugerido**
"*El marciano en el desierto*"	`Comer 3 arriba`	`Comer columna derecha`
"*El marciano en el desierto*"	`Comer 2 izquierda`	`Comer fila superior`
"*Lightbot en Scratch*"	`Diagonal de 4 luces`	`Prender diagonal hacia arriba`
"*Lightbot en Scratch*"	`Bajar`	`Ir a la base de la 2da diagonal`
"*El recolector de estrellas*"	`Siguiente fila`	`Pasar a siguiente fila`
"*María la come sandías*"	`Siguiente fila`	`Pasar a siguiente fila`
"*Instalando juegos*"	`Siguiente compu`	`Pasar a la siguiente compu`
"*La gran aventura...*"	`Buscar llave`	`Buscar la llave`
"*La gran aventura...*"	`Buscar sombrero`	`Buscar el sombrero en el cofre`
"*La gran aventura...*"	`Buscar espada`	`Intercambiar sombrero por espada`
"*La gran aventura...*"	`Rescatar princesa`	`Escapar con la princesa`

Cuando la estrategia resulta clara, se puede juzgar mejor una solución ofrecida. Considerar, por ejemplo, el procedimiento `Mover 2 derecha` en la *"El marciano en el desierto"*. ¿Resulta adecuado en la solución final? Comprar este programa, sugerido en el cuaderno

```
Al recibir 'empezar'
  Comer 2 derecha
  Mover 2 derecha
  Comer 3 arriba
  Comer 2 izquierda
```

con este otro, sugerido en esta guía

```
Al recibir 'empezar'
  Comer todas en sentido antihorario

definir Comer todas en sentido antihorario
  Comer fila inferior
  Comer columna derecha
  Comer fila superior
  Volver a la posición inicial
```

¿Cuál de ellos resulta más claro? Utilizar la estrategia de solución como criterio para decidir las subtareas, y al propósito que cada subtarea tiene como criterio para elegir su nombre, resulta un criterio mucho más claro para juzgar una solución.[7] Y así, se puede ver que `Mover 2 derecha` es solo una parte de la tarea de `Comer fila inferior` en la estrategia general. Puesto que ambos son programas completos, ejecutables, y que cumplen con la solución, el criterio que se propone aquí que debe usarse para juzgarlos debería ser su capacidad para transmitir con claridad la estrategia de solución, algo que se puede decir que el 2do programa hace de manera más adecuada. Observar que no hace falta conocer las definiciones precisas de los procedimientos utilizados en el procedimiento principal (`Comer todas en sentido antihorario`) para convencerse de que, si cada uno de los procedimientos involucrados cumple con el propósito expresado, la solución es correcta. Este es un claro ejemplo del consejo brindado en la sección 3.2.1 al respecto de comenzar siempre definiendo este "esqueleto", aún antes de haber considerado el uso de un solo comando primitivo. Se sugiere fuertemente que se proponga a los estudiantes seguir esta metodología, como forma de fomentar la explicitación de la estrategia de solución, y su expresión mediante procedimientos. Al principio no es sencillo para muchos estudiantes, ya que preferirán la indagación más simple de acumular comandos primitivos para tratar de entender el funcionamiento. Pero si, al realizar las puestas en común y explicaciones, se trabajan las soluciones de esta manera, y se les recuerda al comenzar cada ejercicio que lo intenten, la gran mayoría adquirirá rápidamente esta costumbre, mejorando notablemente la capacidad de analizar y comprender cómo solucionar problemas. Otras actividades de esta secuencia cuyas soluciones ofrecidas en el cuaderno pueden mejorarse utilizando estas observaciones son *"El recolector de estrellas"*, *"Alimentando a los peces"* y *"Reparando la nave"*.

Un tema importante a tener en cuenta a la hora de valorar las soluciones ofrecidas por los estudiantes es considerar con equidad las diferentes estrategias de solución que los estudiantes puedan ofrecer, sin buscar imponer una como la mejor. En este nivel de programación cualquier programa que ofrezca una solución razonable desde el punto de vista de la estrategia es válido, incluso si el mismo no es todo lo eficiente que podría ser. O sea, es mucho más valioso el criterio de la claridad en la elección de la estrategia de solución y la facilidad con la que el

[7] Una aclaración valiosa es que los nombres deben ser cortos si se trata de chicos con poco manejo del lenguaje (como es el caso de chicos más chicos) y un poco más expresivos en caso de chicos más grandes, pero sin que eso lleve a tener nombres extremadamente largos.

programa escrito transmite esta idea (o sea, es legible, en la terminología aquí sugerida), que utilizar exclusivamente el criterio de la eficiencia del programa. Históricamente la eficiencia de los programas fue vista como un criterio básico para juzgar la calidad de una solución pues las máquinas tenían recursos limitados y la tecnología existente no permitía mejoras sustanciales, por lo que era tarea del programador encontrar la solución más eficiente. Sin embargo, en la actualidad las computadoras cuentan con muchísimos más recursos, y la tecnología permite que al ejecutar un programa se produzcan numerosísimas transformaciones que lo hacen más eficiente, por lo que el criterio que debe privilegiarse en un curso inicial (y más en uno que no busca formar programadores profesionales) debe ser la claridad de la solución y no la eficiencia. Si bien la noción de eficiencia sigue siendo importante, se considera que este es un concepto avanzado, más adecuado para impartirse en cursos posteriores.

En cada una de las actividades son posibles varias estrategias de solución. Las únicas excepciones son las actividades *"El recolector de estrellas"*, *"Instalando juegos"*, y *"La gran aventura del mar encantado"*, donde, por la disponibilidad de comandos primitivos, solo una estrategia es posible; esto hace a dicha actividad interesante para discutir cómo los comandos primitivos pueden ayudar o forzar a tener ciertas estrategias. Pero en las demás actividades es bueno permitir que los estudiantes exploren sus propias estrategias de solución, y en caso de que la mayoría vaya por una, de todas maneras mostrar alguna otra o discutir las posibles variaciones. Como ayuda, se ofrecen aquí algunas de las estrategias que fueron consideradas por estudiantes a la hora de realizar cada una de las actividades en los cursos ya dictados.

"El marciano en el desierto" La estrategia de solución que elige la gran mayoría, quizás por la proximidad de las primeras manzanas, es recorrer el borde del escenario en sentido antihorario (o sea, primero la fila inferior de izquierda a derecha, luego la columna de la derecha de abajo a arriba, y finalmente la fila superior de derecha a izquierda). Otras soluciones posibles, menos obvias pero igualmente válidas son realizar el giro en sentido horario (o sea, primero la fila superior de izquierda a derecha, etc.) y también una que coma las manzanas por columna o por fila – como sucederá después en *"El recolector de estrellas"*).

"Lightbot en Scratch" La estrategia de solución usualmente más elegida es la que enciende primero las luces de la diagonal de la izquierda, de abajo hacia arriba, y luego las de la diagonal de la derecha de arriba hacia abajo. Esta solución, sin embargo, requiere 2 procedimientos para encender diagonales. Una solución más concisa, que varios estudiantes proponen, es utilizar un único procedimiento para prender cualquiera de las diagonales de abajo hacia arriba, y hacerlo dos veces; esto requiere posicionarse en la base de la diagonal de la derecha luego de prender la diagonal de la izquierda (o vice-versa si se hace en el otro orden). Una solución más que es posible, mucho menos obvia, es prender ambas diagonales al mismo tiempo, por "filas": primero las dos luces de más abajo, luego las dos luces siguientes, etc. Esta estrategia fue descrita por un estudiante como "prender las vías del tren", ya que el no veía dos diagonales, sino unas vías del tren.

Esta última observación es interesante para entender que no existe una única mirada sobre un problema, aunque haya visiones que son más usuales; en muchas ocasiones, la utilización de lo que se denomina "pensamiento lateral" (o también "pensar fuera de la caja", por su forma inglesa *thinking out of the box*) es importante para encontrar mejores soluciones.

"El alien y las tuercas" En esta actividad puede optarse por recorrer la diagonal de arriba hacia abajo, de abajo hacia arriba, o realizar el recorrido por filas, moviéndose progresivamente más cerca hasta la tuerca (pero esto es poco común, pues con las herramientas disponibles se precisan demasiados procedimientos similares, aunque diferentes).

"María, la come sandías" En esta actividad la estrategia más obvia es la de comer las sandías de cada fila de izquierda a derecha, subiendo por filas completas, y luego comer las de la columna de la izquierda. Otras soluciones menos obvias son comer las sandías por columnas, o hacerlo comenzando por las filas superiores, o recorriendo las filas de derecha a izquierda.

"Alimentando a los peces" Todas las estrategias deben comenzar por recoger el alimento para los peces. Luego se pueden obtener variaciones recorriendo primero la fila superior y luego la inferior, o vice-versa, recorrer las filas de izquierda a derecha o de derecha a izquierda (e incluso una en cada sentido), y la menos obvia, recorrer los peces por columna.

"Reparando la nave" Aquí puede comenzarse primero por el hierro y luego por el carbón o vice-versa, completando cada uno de ellos antes de pasar al siguiente, o realizar alguna alternancia entre hierro y carbón, típica, pero no exclusivamente, uno y uno.

El hecho de utilizar el consejo de primero expresar la misma usando procedimientos, antes de pensar en las primitivas, contribuye a pensar en dividir en subtareas, pero no garantiza que esta idea sea utilizada en todo su poder. Considerar, por ejemplo, la siguiente solución de la actividad *"Reparando la nave"*

```
Al recibir 'empezar'
  Reparar la nave y escapar

definir Reparar la nave y escapar
  Buscar 3 hierros
  Buscar 3 carbones
  Escapar
```

donde los procedimientos `Buscar 3 hierros` y `Buscar 3 carbones` aún no han sido definidos (si se sigue el principio de comenzar la resolución expresando la estrategia de solución mediante procedimientos antes de considerar utilizar primitivas, este es un posible estado del programa en construcción). ¿Qué puede decirse de esta propuesta de definición al primero de estos procedimientos?

```
definir Buscar 3 hierros
  Repetir 3
    Repetir 3
      Moverse arriba
    Tomar hierro
    Repetir 3
      Moverse abajo
    Depositar
```

Ahora considerar esta otra. ¿Qué diferencias observa?

```
definir Buscar 3 hierros
  Repetir 3
    Ir al hierro
    Tomar hierro
    Volver del hierro
    Depositar

definir Ir al hierro
  Repetir 3
    Moverse arriba

definir Volver del hierro
  Repetir 3
    Moverse abajo
```

Aquí, el procedimiento `Buscar 3 hierros` está definido en base a 4 tareas: ir, tomar, volver y depositar. A su vez, dos de estas tareas, ir y volver, están definidas en base a repetición y primitivas. En la primera de las propuestas, en cambio, todo está definido en el mismo cuerpo del procedimiento. ¿Cuál solución resulta más clara? Si el estudiante aún no domina completamente la división en subtareas y su expresión mediante procedimientos, es usual que ofrezca la primera de las soluciones, en lugar de la segunda. Si se siguen los principios conceptuales que esta guía busca transmitir, es más factible proponer la segunda solución, que es mucho más legible y entendible que la primera. La solución que utiliza 3 repetir en el mismo procedimiento es la forma en que se enseñaba usualmente con los métodos tradicionales, y obedece a una anticuada concepción del excesivo costo de utilizar procedimientos. Pero en la programación moderna el uso de procedimientos no resulta tan costoso, y por otra parte se ha comprobado que el aspecto comunicacional del programa es extremadamente importante. Por otro lado, la forma sugerida aquí (la segunda), es mucho más cercana a la forma de pensar que tienen las personas; se puede decir, entonces, que sigue lineamientos asociados con el pensamiento de alto orden, y es por ello mucho más adecuada como forma de pensar y como guía para construir programas.

La regla de trabajo que se brinda aquí para desarrollar este tipo de pensamiento es simple: nunca utilizar más de una estructura de control dentro de un procedimiento. Este es un ejemplo de lo que se explicó en la sección 3.2.1. En este ejemplo, la utilización de 3 repeticiones en el procedimiento `Buscar 3 hierros` indica que no se ha completado satisfactoriamente la división en subtareas, por lo que deben descubrirse aún algunas subtareas más. Por supuesto, esta regla de trabajo no es inviolable, pero en general funciona en la gran mayoría de los casos. Sin embargo, esta regla de trabajo no fue aplicada a todas las soluciones ofrecidas en el libro, por lo que las mismas deberían ser revisadas antes de ofrecer las soluciones finales durante el curso. Las actividades cuyas soluciones ofrecidas en el cuaderno no siguen esta regla son *"El recolector de estrellas"*, *"Alimentando a los peces"* y *"Reparando la nave"*.

Hay una última cuestión que debe considerarse al diseñar estrategias de solución que involucran procesar una serie de elementos iguales utilizando repetición, referida a algunas variaciones necesarias. Algunas de estas actividades, como *"Prendiendo las compus"*, son sencillas, pues todo el procesamiento y movimiento ocurre dentro de la repetición. En otras en cambio, como por ejemplo *"Lightbot en Scratch"* o *"El recolector de estrellas"*, hace falta considerar una variación. El problema es que en estas dos actividades la cantidad de procesamientos y la cantidad de movimientos entre elementos no es la misma, sino que difieren en uno. Por ello, uno de los procesamientos debe hacerse fuera de la repetición. En el caso de *"El recolector de estrellas"*, por ejemplo, hay 4 filas pero solo hay que cambiar de fila 3 veces; entonces al escribir la repetición, y dado que la misma involucra repetir procesar la fila y moverse a la siguiente, solo puede hacerse 3 veces – la última fila (o la primera) deben procesarse fuera de la repetición. Este fenómeno es recurrente, y debe ser considerado al analizar la estrategia de solución, la división en subtareas

para expresarla y el uso de una repetición para codificar la solución. Sin embargo, y dependiendo del grupo de estudiantes al que se dicta el curso, esto puede explicitarse en forma genérica o simplemente tratarlo cono cada estudiante que lo requiera de manera particular. Es usual que todos los estudiantes intenten primero una repetición de 4 veces, antes de darse cuenta que eso falla porque no se puede mover una cuarta vez; en ese caso sirve preguntarles *"¿Cuántas veces debe moverse el extraterrestre?"* para que deduzcan por sí solos que la repetición debe ser tres veces (y el procesamiento afuera de la repetición surge prácticamente como necesario). Este fenómeno volverá a aparecer al tratar la repetición condicional en la sección 4.9.

El objetivo principal de esta secuencia didáctica es practicar la estructura de control para repetición, en diversas combinaciones. Sin embargo, si se utiliza el marco conceptual propuesto aquí, estas actividades son una excelente oportunidad de practicar los conceptos de estrategia de solución (3.2.1), legibilidad y elección de nombres (3.2.2) y uso adecuado de procedimientos (3.3.1), además de hacer uso extensivo de la didáctica por indagación (3.1).

4.7 Secuencia didáctica 7: Escenarios cambiantes

La secuencia didáctica 7 propone volver al entorno de la secuencia 1, con *"El robot humano"* (ver sección 4.1), pero donde ahora el escenario puede sufrir variaciones y por lo tanto hace falta una nueva herramienta del lenguaje, no considerada hasta el momento: la alternativa condicional.

La actividad propuesta para esta secuencia ofrece un desafío interesante a la hora de realizarla. El objetivo fundamental de esta secuencia es entender que un *mismo programa* tiene que funcionar en *diversos escenarios*, cada uno de los cuales puede presentar variaciones respecto de los otros. Las variaciones que se van a considerar empezarán siendo muy simples y concretas para luego irse complejizando lentamente. De todas maneras, como el curso pretende ser solo un curso inicial, las variaciones siempre serán controladas (o sea, no se considerarán escenarios totalmente arbitrarios durante el curso).

Si no se enuncia con cuidado y correctamente, esta actividad puede resultar un obstáculo para la didáctica por indagación, puesto que requiere que los estudiantes realicen un proceso de abstracción sutil y complejo con respecto al aula. Es usual que los estudiantes se pierdan considerando detalles que para el docente resultan irrelevantes, y no logren hacer foco en aquellos que constituyen la variación que se busca resaltar. Puede resultar más sencillo en algunos casos comenzar con la secuencia didáctica 8 (ver la sección siguiente), donde la variación del escenario es muy obvia (la presencia o no de una banana, o la presencia de una fruta que puede ser banana o manzana), y recién luego volver a considerar esta secuencia. Se recomienda especialmente poner mucho cuidado en la enunciación y presentación de la actividad propuesta en esta secuencia, y especialmente en **guiar cuidadosamente** a los estudiantes para que los resultados sean los esperados.

En caso de realizarla primero, hay varias cosas a tener en cuenta. La primera es que como el objetivo es que comprendan que *un único programa* debe funcionar en *situaciones cambiantes*, debe plantearse con claridad cuál es la variación de escenario que se debe considerar, preferentemente que haya solo 2 variaciones posibles, y también destacar que el mismo programa debe funcionar en ambos. La segunda cuestión a tener en cuenta es que a pesar de que los estudiantes a esta altura del curso ya están manejando adecuadamente estrategias de solución y su expresión mediante procedimientos, al volver a esta actividad muchos vuelven a caer en la tentación de dar una solución instrucción a instrucción. Por ello debe prestarse especial atención en guiar a los estudiantes a pensar primero la estrategia de solución (e incluso no es necesario llegar a plantear cada una de las subtareas instrucción a instrucción).

Por ejemplo, puede trabajarse con la siguiente variación del escenario: el salón tiene 2 puertas, y cada día solo una de ellas estará abierta (y la otra estará cerrada, sin posibilidades de abrirla, por ejemplo, por estar con llave). Desde la posición del docente, es sencillo determinar cuál de las puertas está abierta. El objetivo debe ser escribir un programa que permita que el docente, al ejecutarlo mecánicamente, pueda salir del aula, sin importar cuál de las 2 puertas es

la que está abierta. Enunciar esto adecuadamente hasta que se entienda es parte fundamental de esta secuencia. Entonces la estrategia buscada, a la que debe guiarse a los estudiantes, sería

```
Si la puerta de adelante está abierta
  Ir a la puerta de adelante
  Salir
y si no (la puerta de atrás está abierta)
  Ir a la puerta de atrás
  Salir
```

Observar que no hay necesidad de escribir cada una de las subtareas en términos primitivos, ya que alcanza con considerar la estrategia de solución para cumplir el objetivo buscado de entender las variaciones de escenarios. Es importante dejar que los estudiantes propongan soluciones, hasta que alguno establezca la necesidad de usar la alternativa condicional, donde la condición permite determinar cuál de las puertas está abierta. Si luego de un tiempo razonable esta idea no aparece, debe tratar de sugerirse la necesidad de determinar por cuál de las puertas salir a través de una condición, y la construcción si-entonces debería ocurrírseles.

Estas mismas consideraciones al respecto de la importancia de transmitir que *el mismo programa* debe funcionar en *diversos escenarios posibles* debe repetirse en la secuencia siguiente.

4.8 Secuencia didáctica 8: Alternativas condicionales en Scratch

Esta secuencia propone continuar considerando variaciones en el escenario, para utilizar la herramienta del lenguaje llamada alternativa condicional (`if-then-else`) sobre comandos, pero hacerlo dentro del entorno de trabajo (Scratch en el cuaderno, Pilas Bloques si se está trabajando con el nuevo entorno). Si se sigue el enfoque conceptual propuesto en esta guía, la idea es que los estudiantes realicen primero un proceso de indagación para descubrir que el escenario es variable, y de esa manera el comando de alternativa condicional se vuelve necesario. Como se discutió en la sección 4.7, puede comenzarse con esta secuencia y luego volver a la secuencia número 7, o seguir el orden propuesto en el cuaderno. La posibilidad de realizar primero las actividades de esta secuencia, antes que las de la sección anterior, tiene que ver con que las variaciones del escenario propuestas aquí son bien concretas (si la banana está o no está, si la fruta que aparece es una manzana o una banana), y por ello sencillas de descubrir. Para motivar aún más la necesidad de tener alternativa condicional, y de entender por qué es necesaria, se puede sugerir que solamente tienen permiso de utilizar las herramientas vistas hasta el momento (comandos primitivos, secuencia, procedimientos y repetición simple), pero no otras. Como en Scratch están disponibles todas las herramientas, vistas y no vistas, esta es una práctica que ya se viene haciendo desde el principio, pero si se utiliza Pilas Bloques resultará una novedad: hay herramientas disponibles que no deben ser consideradas hasta después. Si se sigue esta propuesta, cuando los estudiantes ofrecen su primera solución, la misma fallará invariablemente en algunos casos: si se come la banana, fallará cuando no haya banana, por ejemplo. Para que este hecho se destaque, el estudiante debe probar su programa varias veces, y poder entender que el mismo debe funcionar **siempre**, en todo escenario que se presente dentro del desafío. O sea, hay que entender que un programa no es correcto si funciona la mitad de las veces. Para algunos estudiantes esto resulta fácil de entender, pero a otros les cuesta más. Y dado que es una característica fundamental para entender que la alternativa condicional es una herramienta imprescindible, es importante trabajarlo hasta que todos lograron dicha comprensión.

En las dos actividades principales de esta secuencia se trabajan dos variaciones de la alternativa condicional. En la más completa de ellas, la que se presenta en la actividad *"La elección del mono"*, se trata de elegir entre una de dos posibilidades, en ese caso, entre comer una banana o comer una manzana. En esa actividad se observa con claridad que se trata de una alternativa entre dos posibilidades. En la forma más simple, presentada en *"El mono y las bananas"*, la

alternativa se utiliza para decidir si hacer una acción o no hacerla. Es también una alternativa, porque se elige entre hacer o no hacer la acción, pero entenderla como alternativa es más sutil que en el otro caso. En cursos tradicionales se suele nombrar a la alternativa como *condicional* destacando el hecho de que se usa para esta segunda forma, donde la ejecución de una acción es condicionada a que suceda cierta situación, postergando y muchas veces invisibilizando el hecho de que realmente se trata de una alternativa. La correcta denominación, *alternativa*, permite destacar su verdadera naturaleza, mostrando su valor conceptual dentro del marco.

Para poder utilizar la alternativa condicional, hace falta ser capaces de expresar condiciones sobre el estado del escenario. Para ello se utiliza una categoría de elementos del lenguaje denominada *sensores*. Los sensores son expresiones que expresan información sobre el estado del escenario, y que como se mencionó en la sección 3.3, constituyen los "sustantivos" de un lenguaje de programación. En este caso, la información que los sensores ofrecen es establecer si cierta condición sobre el escenario es verdadera o falsa. El sensor que más se utilizará en las actividades de este curso es el denominado `¿Tocando objeto?` (que en Pilas Bloques aparece como `¿Hay objeto?` o `¿Llegué al lugar?`) que es verdadero cuando el personaje se encuentra en la misma ubicación que el objeto por el que se pregunta. Así es como puede determinarse, por ejemplo, si hay o no una banana (`¿Tocando banana?` o `¿Hay banana?`). En los cursos donde se explicita el marco conceptual es útil detenerse un momento y agregar los sensores en la sección de expresiones, junto con los datos primitivos.

Luego de las dos primeras actividades de presentación, el cuaderno propone varias actividades de ejercitación: *"Laberinto corto"*, *"Tres naranjas"*, *"Lightbot recargado"* (llamado *"Tito recargado"* en Pilas Bloques), y *"Laberinto largo"*. Al proponer y realizar estas actividades, es importante volver a reforzar los consejos metodológicos que se empezaron a trabajar en la sección 4.6.

- Comenzar a intentar la solución expresando la tarea completa en un único procedimiento al que se le elige nombre adecuado.
- Procurar no utilizar más de una misma estructura de control en cada procedimiento. Esto vale para varias repeticiones, o para combinaciones de repeticiones y alternativas. El caso de varias alternativas se discute en los próximos párrafos.
- Elegir nombres de procedimientos que expliciten el propósito de un procedimiento, y contribuyan a la legibilidad del programa.

Idealmente cada procedimiento debería expresar la solución de una única tarea de manera simple, y su nombre debería expresar con claridad tal tarea. Como se explicó, esto es consecuencia de utilizar el marco conceptual aquí planteado, y contribuye a que los programas sean más entendibles y más modificables, además de fomentar y facilitar el pensamiento de alto orden.

Con respecto a cada actividad en sí, es conveniente repasar algunas de sus características. En cada caso, resulta bueno sugerir a los estudiantes explorar antes las posibles variaciones del escenario, como metodología previa a empezar a pensar una solución. En caso de que luego de un rato de análisis, los estudiantes no descubran por sí solos alguna de las características importantes de cada escenario (las cosas que nunca cambian, y las que sí lo hacen), pueden darse sugerencias para que se enfoquen en ellas. En todos los casos no hay margen de variación con respecto a la estrategia de solución, que es bastante sencilla y es similar en todos.

"Laberinto corto" Esta actividad es muy simple y similar a la de la elección del mono, aunque la condición es levemente menos concreta (la explicitación de la flecha la vuelve más concreta, pero en un caso más realista puede resultar más complejo determinar hacia donde avanzar). La variación del escenario está en el sentido en el que se puede avanzar, lo cual es indicado siempre por la flecha (o flechas) en el piso. Si no lo descubren por sí mismos, puede explicitarse que siempre hay una flecha en el piso y que la misma apunta en la dirección correcta.

En el cuaderno explica cómo se pueden obtener dos soluciones, una a través del uso de la alternativa en su forma completa (`si-entonces-sino` o `if-then-else`), y otra combinando dos formas simples (`si-entonces` o `if-then`). Las soluciones finales no son completamente equivalentes en cualquier escenario, pero en los que se presentan aquí cualquiera de las dos funciona bien.

"Tres naranjas" En esta actividad se debe combinar una repetición (3 veces) con la decisión de si se debe comer o no en cada casillero (dependiendo de si hay naranja o no la hay). Siempre aparecen 3 casilleros, y la variación consiste en que en cada uno puede aparecer o no una naranja.

En el cuaderno se explica con claridad que puede darse una solución sin utilizar procedimientos, pero esto no es conveniente, lo cual sigue uno de los principios metodológicos sugeridos antes. Sin embargo, el cuaderno no sigue el otro, y no ofrece un procedimiento principal que exprese la totalidad de la solución.

"Lightbot recargado" Esta actividad es similar a la anterior, y aplican las mismas observaciones. En cuanto a las características del escenario, siempre habrá 11 celdas, pero solo las 10 primeras pueden o no tener una luz; la última nunca tiene luz. Esto permite que la única diferencia con la solución anterior, tal y como se explica en el cuaderno, es que la acción de prender la luz debe realizarse antes de avanzar.

"Laberinto largo" Este problema es una combinación de todos los anteriores, y si bien puede resultarles complejo a algunos estudiantes, orientándolos a analizar primero las variaciones del escenario (21 celdas, 20 de ellas con flechas y una de llegada, donde las flechas pueden ser hacia abajo o la derecha) y pensar y expresar la estrategia de solución antes de empezar a poner código puede ayudarlos a ver esta similaridad, y facilitarles la solución.

Justamente, la idea es que a esta altura del curso la metodología de trabajo los vaya llevando a la solución de manera sencilla. El pensamiento de alto orden necesario se va formando de a poco, y es el verdadero objetivo del curso. Por eso es tan importante que el docente esté enfocado en el marco conceptual adecuado.

Corresponde realizar una consideración a realizar con respecto a la metodología que indica que no es conveniente utilizar más de una estructura de control por procedimiento, como forma de fomentar el uso de procedimientos y la división en subtareas. Dado que la herramienta usual de alternativa condicional de los lenguajes solo permite elegir entre dos alternativas, si el problema a expresar tiene que elegir entre más de dos se hará necesario utilizar varias construcciones para ir distinguiendo los casos (por ejemplo, si en lugar de 2 frutas fuesen 3, habría que expresarlo como[8]

```
definir Comer fruta
  si ¿Tocando banana?
   entonces Comer banana
   sino si ¿Tocando manzana?
         entonces Comer manzana
         sino     Comer uva
```

Técnicamente esto viola la recomendación de no utilizar más de una estructura de control, pero es necesario para reemplazar la falta de una alternativa que considere más de dos posibles caminos para elegir. En algunos lenguajes hay formas de alternativa que permiten este tipo de

[8] Recordar que en Pilas Bloques sería `¿Hay banana acá?` y `¿Hay manzana acá?`.

multi-alternativa, pero es un concepto sutil que cae fuera de los objetivos de este curso. Solo se lo menciona aquí por su conflicto con la recomendación metodológica ofrecida.

En esta secuencia se presentaron dos nuevas herramientas del lenguaje: la alternativa condicional y los sensores, comenzando a ampliar el abanico de herramientas del lenguaje. De explicitar el marco conceptual, debe agregarse *alternativa condicional* a la subcategoría *comandos* de *herramientas del lenguaje* (3.3.1), y también agregar *sensores* a la categoría de *expresiones* (3.3.2). Los primeros ejemplos trabajan con escenarios que requieren solo el uso de una alternativa simple, pero luego se requieren combinar las alternativas con los otros conceptos trabajados previamente, mostrando así la riqueza del lenguaje de programación para expresar soluciones. Es importante destacar siempre la necesidad y utilidad de los conceptos de didáctica por indagación (3.1), estrategia de solución y división en subtareas (3.2.1) y su expresión mediante procedimientos (3.3.1), y la legibilidad del código (3.2.2).

4.9 Secuencia didáctica 9: Escenarios con secuencias de tamaño variable

Al igual que en la secuencia didáctica anterior, en esta se presenta una nueva herramienta del lenguaje: la repetición condicional. Y al igual que con la alternativa condicional, la idea es que la herramienta está motivada en una variación en el escenario. En este caso, la variación que debe considerarse tiene que ver con secuencias cuya cantidad de elementos no es conocida de antemano: cada escenario podrá tener diferente cantidad. Nuevamente, es recomendable sugerir a los estudiantes que se familiaricen con las variaciones del escenario antes de comenzar a pensar la solución, y que traten de solucionarlo con los elementos que ya poseen (procedimientos, repetición simple y alternativa condicional) hasta convencerse de que eso no es posible en una situación general.

Al intentar soluciones que no utilicen la nueva herramienta, los estudiantes en ocasiones ofrecen dos variaciones que merecen mayor discusión: una que no es correcta y otra que sí lo es.

La solución que no es correcta consiste en realizar una repetición simple un número grande de veces, donde se utiliza la alternativa para encender una luz y moverse solo cuando existe.[9]

```
definir Prender todas las luces
  repetir 200
   Prender luz si hay lamparita

definir Prender luz si hay lamparita
  si ¿Tocando luz?
   entonces Prender luz
            Avanzar
```

¿Por qué esta solución no es correcta? Si se consideran como escenarios posibles solamente aquellos que pueden ser generados por el entorno, y dado que el espacio para dibujar el escenario es pequeño, el programa dado funcionará en todos ellos. Sin embargo, la solución ofrecida no es conceptualmente correcta, por dos razones. Primero, si se supone que el programa debe funcionar en escenarios donde realmente no se conoce el número de luces a encender, ¿qué va a suceder cuando haya 201 luces? Claramente el programa no cumplirá su propósito, y por eso es incorrecto (y también el nombre elegido para él, ya que habría sido más adecuado llamarlo `Prender hasta 200 luces`). Y esto sería igual si en lugar de 200 hubiese cualquier otro número fijo; el punto crucial es ese: en este problema no se conoce de antemano ni el número de luces, ni el máximo de las que pueden haber. En segundo lugar, ¿qué pasa cuando solo hay unas pocas luces, por ejemplo 2? El programa establece que deben hacerse 200 repeticiones, 198 de

[9] En Pilas Bloques en lugar del sensor `¿Tocando luz?` aparece el equivalente `¿Hay lamparita acá?`.

las cuales simplemente serán para preguntar (de manera redundante) si hay o no una luz para prender. En una solución verdaderamente conceptual, lo ideal sería que si hubiese 2 luces, el procedimiento se repitiese 2 veces, y si hubiera 2000, se repitiese 2000 veces. Se trata de un par de nociones sutiles, vinculadas con la forma conceptual de pensar, y debe prestarse especial atención, porque en algunos cursos suelen suscitarse discusiones, ya que no todos perciben el caso general a primera vista. Por ello se recomienda ser cuidadosos a la hora de enunciar esta actividad, y de discutir una solución como la recién trabajada. En última instancia, se puede establecer como regla de trabajo que idealmente una repetición no debe trabajar más de la cuenta, y entonces si no se conoce el número de veces a repetir (como es el caso aquí), la repetición simple no es suficiente.

La otra solución posible sin utilizar la herramienta que se busca motivar (repetición condicional), utiliza procedimientos de una forma extremadamente creativa y adecuada desde el marco conceptual de esta guía: recursión. Si bien es muy poco probable que a un estudiante se le ocurra esta solución, si llega a darse el caso de que alguno la proponga, debe tenerse mucho cuidado de no desalentar su creatividad. La recursión consiste en que el comando nuevo que un procedimiento define puede utilizarse en la misma definición del procedimiento.

```
definir Prender todas las luces
  Prender las luces que faltan

definir Prender las luces que faltan
  si ¿Tocando luz?
   entonces Prender luz
            Avanzar
            Prender las luces que faltan
```

Ahora bien, si la solución es correcta y es deseable, ¿por qué no se la propone como la manera correcta de hacerlo? El tema es que el concepto de recursión es muy abstracto, bastante complejo y posee muchísimas sutilezas y problemas, por lo que resulta más apropiado su tratamiento en forma profesional. Y dado que el curso propuesto pretende ser un curso introductorio y no uno profesional, es preferible evitar en lo posible el tema de la recursión. A pesar de haber decidido no incluir este tema en un primer curso, es especialmente deseable que no se inhiba al estudiante que propone una solución recursiva, pues que se le haya ocurrido tal solución es indicio de que está entendiendo perfectamente los conceptos que se busca transmitir, y los puede utilizar de formas muy creativas. La sugerencia es que se le recomiende a este estudiante profundizar el tema de recursión luego de finalizado el curso, alentándolo a continuar estudiando programación.

En Pilas Bloques, dado que se restringe el uso a ciertas operaciones, para expresar cualquiera de las dos soluciones deben utilizar un `si-entonces-sino` donde la alternativa en caso verdadero se deje vacío. Por esa razón es menos probable que estas soluciones sucedan si se utiliza dicho entorno de trabajo.

Una vez que los estudiantes intentaron solucionar el problema con las herramientas existentes, y habiendo discutido las propuestas de solución, es el momento de presentar la nueva herramienta: la repetición condicional. En diferentes lenguajes de programación existen diferentes variantes de esta herramienta, siendo la más popular la conocida como `repetir-mientras` o `while-do`. En este curso, en cambio, se propone la utilización de la variante `repetir-hasta-que` (menos común en los lenguajes de uso industrial, pero conocida como `repeat-until` en ellos), pues se ha comprobado en la práctica que la misma resulta más accesible para estudiantes que recién se inician. Una posible razón para esto es que la condición que debe utilizarse en la variante `repetir-hasta-que` establece claramente el momento de finalizar la repetición, mientras que la condición del `repetir-mientras` establece la condición con la que hay que continuar repitiendo. Si bien la equivalencia es sencilla (negar la condición para cambiar de variante), agre-

gar el tratamiento de condiciones negadas incrementa la complejidad conceptual, y es preferible postergar el tratamiento de condiciones complejas para una secuencia didáctica posterior.

4.9.1 Recorridos y repetición condicional

En esta secuencia didáctica se presenta la última de las herramientas conceptuales, vinculada con la algorítmica: la noción de *recorrido*. Como explica el cuaderno, un *recorrido* es una forma de procesar una secuencia de elementos de manera estructurada. Básicamente, es un esquema de cómo dividir en subtareas una tarea que implica procesar una secuencia, garantizando que el tratamiento cubrirá todos los elementos deseados, y terminar con éxito. Puesto que la repetición condicional introduce la posibilidad de que los programas no terminen, la estructura de recorrido permite utilizar repetición condicional garantizando que siempre se terminará de procesar. Un tratamiento completo de la no-terminación y cómo evitarla es mucho más complejo y debe dejarse para cursos avanzados. Un recorrido tiene las siguientes partes:

- `Iniciar recorrido`: usualmente, posicionarse en el primer elemento, y realizar las inicializaciones que correspondan.
- `Condición de fin de recorrido`: condición que indique cuando se acabaron los elementos a procesar.
- `Procesar elemento actual`: el procesamiento propiamente dicho de un elemento por vez.
- `Pasar al siguiente elemento`: la forma de pasar al siguiente elemento de la secuencia, si existe.
- `Finalizar recorrido`: en los casos en que fuera necesario, retornar la información pedida, volver a una posición dada, etc.

Una o más de estas partes pueden omitirse. Por ejemplo, en la actividad *"Super Lightbot 1"*, el robot ya comienza en el elemento inicial, por lo que no hace falta tener `Iniciar recorrido`, y al terminar tampoco tiene que volver al principio, ni devolver nada, y por lo tanto tampoco hace falta `Finalizar recorrido`.

Es importante, al momento de presentar la idea de recorrido, que se vuelva a hacer énfasis en las buenas prácticas que se vienen trabajando: representar cada subtarea con un procedimiento, nombrar adecuadamente las subtareas, utilizar un procedimiento inicial que capture la solución completa (esta última no fue utilizada en las soluciones provistas en el cuaderno, pero se recomienda continuar con esta buena práctica). En todas las actividades propuestas en esta secuencia, la condición de fin de recorrido es sencilla de descubrir, e involucra el uso de un único sensor. Las otras partes, en cambio, pueden requerir separar en una subtarea. Se analizan los ejemplos según cada actividad.

"Super Lightbot 2" En esta actividad para la subtarea de `Pasar al siguiente elemento` alcanza con el comando primitivo `Avanzar`, por lo que no tendría sentido hacer un procedimiento para ella, pero para la tarea de `Procesar elemento actual` es conveniente hacer un procedimiento (llamado adecuadamente `Prender si es luz` en el cuaderno).

"Laberinto con queso" En esta actividad tanto `Procesar elemento actual` como `Pasar al siguiente elemento` son representadas en el cuaderno mediante procedimientos, `Comer queso si hay` y `Mover`, respectivamente. Observar que, según el principio de elección de nombres, sería más adecuado que `Mover` se llamase `Avanzar un lugar en el laberinto` o algo similar.

"El detective Chaparro" Esta actividad tiene un par de dificultades que las previas no tenían. En primer lugar, la búsqueda no consiste en procesar *todos* los elementos, sino que debe detenerse al encontrar al sospechoso. Por esa razón, el orden entre las subtareas `Procesar elemento actual` y `Pasar al siguiente elemento` deben estar invertidas (así si el sospechoso es el último, no se cae del escenario). Por esto, también debe procesarse el primer elemento fuera de la repetición, ya que si se lo encuentra al principio, ya no es necesario realizar más repeticiones. En segundo lugar, se hace necesaria la utilización de la subtarea `Iniciar recorrido`, pues el detective no arranca siempre desde el primero (esto debe ser observado por los estudiantes). En este caso, alcanza con utilizar el comando primitivo `Primer sospechoso`, que sería mejor si se llamase `Ir al primer sospechoso`.

Es conveniente que las soluciones de esta actividad se discutan entre todos, pues puede ser que muchos tengan dificultades con encontrar la solución adecuada, debido a las sutilezas mencionadas.

"Fútbol para robots" Esta actividad tiene dos particularidades propias: deben realizarse la actividad de patear pelota y volver para 7 filas, y en cada una se debe usar una repetición condicional para ir, y otra para volver al inicio de la fila luego de patear la pelota. Es conveniente que cada una de estas actividades se exprese en un procedimiento propio. En el cuaderno, el procedimiento de `Volver al inicio` se utiliza por separado del procedimiento `Hacer gol`, por lo que la repetición simple general tiene 3 partes en lugar de 2. Si se sigue la idea de recorrido presentada, sería conveniente que estas dos tareas fuesen agrupadas en un procedimiento `Hacer gol y volver` que simplemente las englobe, ya que éste sería el que representa a la subtarea de `Procesar fila actual`.

"Prendiendo las compus" Esta actividad es similar a la anterior, en cuanto a que combina 4 repeticiones de una tarea que requiere repetición condicional. La diferencia distintiva es que se requieren 4 procedimientos para procesar cada una de las filas (preparando el terreno para la incorporación de la idea de *parámetro* en la secuencia didáctica siguiente), y que cada uno de ellos debe comenzar haciendo un movimiento como `Iniciar recorrido`, pues todas las esquinas son iguales.

"El mono sabe contar" Esta actividad tiene gran similaridad con *"Fútbol de robots"*, y aplican todas las observaciones realizadas en la misma.

Al discutir la forma de expresar la división en subtareas propuesta por el recorrido en algunas de las actividades, puede surgir una cuestión acerca de la ubicación de ciertos procedimientos. En particular, en *"Fútbol de robots"* y *"El mono sabe contar"*, luego de procesar una fila, debe volverse al inicio de la misma para poder pasar a la siguiente. Si se siguen las recomendaciones de esta guía, existirá un procedimiento `Volver al inicio de fila`, que debe ser utilizado luego de haber procesado todos los elementos de la fila. Pero, justamente, este procedimiento puede ubicarse en diferentes partes del programa (siempre después de haber procesado la fila): puede estar dentro del procedimiento `Procesar fila`, puede estar dentro de `Pasar a la siguiente fila`, o puede estar de forma independiente, entre ambos. Cualquiera de estas variaciones puede considerarse correcta, si bien la última no respeta de manera fiel el esquema de recorrido sugerido. La razón para mencionarlo aquí es alertar a los docentes que no deben corregir las soluciones de los estudiantes si ellos proponen una ubicación diferente a la ofrecida en el cuaderno, o a la pensada por otros compañeros, pues no es objetivo del curso discutir esta sutileza. Es mucho más importante fomentar la capacidad de los estudiantes de encontrar y juzgar sus propias soluciones, que la de discutir este tipo de variaciones menores. Si el curso lo amerita (por su manejo de conceptos, por su interés en temáticas específicas de programación, o por tratarse de estudiantes avanzados o adultos), puede discutirse las alternativas y que cada

uno intente justificar por qué lo ubicarían en uno u otro lado; pero esto solamente constituiría un adicional no imprescindible.

La variación discutida al final de la sección 4.6 acerca de que el último (o primer) elemento debe ser tratado de manera separada a la repetición vuelve a aparecer en *"El detective Chaparro"*, *"Fútbol de robots"* y *"El mono sabe contar"*.

Si se está explicitando el marco conceptual, luego de realizar la actividad de presentación de la repetición condicional y de la idea de recorrido (habiendo permitido primero explorar las soluciones, como dicta la didáctica propuesta), es momento de agregar la *repetición condicional* (3.3.1) a las *herramientas del lenguaje*, y la noción de *recorrido* (3.2.3) a las *herramientas conceptuales*.

4.9.2 Datos primitivos

Las dos actividades finales de ejercitación de esta secuencia didáctica, *"El mono cuenta de nuevo"* y *"El super viaje"* no son acerca de repetición condicional ni de recorridos. El tema que las mismas presentan es el uso de una nueva forma de expresiones: los datos primitivos. Al igual que los sensores son expresiones que devuelven información inmediata sobre el escenario, los datos primitivos también devuelven información sobre el mismo, con la diferencia de que esa información suele ser más compleja de percibir. En los ejemplos, la longitud de la fila sobre la que está parado el mono, o la cantidad de kilómetros que le falta recorrer a Superman.

En el caso de la actividad *"El mono cuenta de nuevo"*, puede reutilizarse la estrategia de solución y la división en subtareas vista en *"El mono sabe contar"*, pero cambiando la repetición condicional por una repetición indexada. Es interesante que esto sea deducido por los estudiantes como parte de su proceso de indagación.

En el marco conceptual se puede agregar *Datos primitivos* dentro de la subcategoría *Expresiones* (3.3.2) de *Herramientas del lenguaje*. En ningún momento debe dejar de resaltarse la necesidad de pensar primero la estrategia de solución (3.2.1), la división en subtareas expresadas mediante procedimientos (3.3.1), y la legibilidad obtenida de elegir buenos nombres para los mismos (3.2.2). También es conveniente cerrar la secuencia didáctica discutiendo las diferencias al escribir un recorrido (3.2.3) utilizando repetición condicional o repetición simple.

4.10 Secuencia didáctica 10: Canciones y estribillos

En la secuencia didáctica 10 se trata de presentar la noción de parámetros relacionándola con un tema completamente separado, en principio, de la programación: canciones. Para ello, primero se trata de entender que los estribillos de las canciones ofrecen muchísima similaridad con la noción de procedimiento en programación. La idea para conseguir esto es hacer escuchar a los estudiantes una canción que conozcan (que posea un estribillo fijo) y pedirle que la escriban en papel. Una vez hecho esto, preguntarles quiénes de ellos copiaron el estribillo más de una vez, y quiénes en cambio utilizaron alguna forma de abreviarlo (por ejemplo, usando la palabra ESTRIBILLO). Al realizar la puesta en común de la actividad, se recomienda utilizar las mismas convenciones que en la secuencia didáctica "Programación en papel cuadriculado": en la definición del estribillo se escribe `"ESTRIBILLO"` usando comillas, y al utilizarlo nuevamente se escribe `[ESTRIBILLO]`, entre corchetes (en el cuaderno no se realiza esta distinción). De esta forma, se aprovecha las notación y las ideas ya trabajadas sobre procedimientos escritos en papel y resulta más sencillo establecer el vínculo entre el concepto de procedimiento y el de estribillo. Dado que la actividad puede resultar tediosa para los estudiantes (tener que escribir toda una canción), muchas veces existe la tentación de saltear la parte de la indagación y contarles directamente el resultado. Esto usualmente es más cómodo, pero hacer que los estudiantes más lentos demoren en adquirir los conceptos buscados, por lo que se recomienda tratar de hacer la actividad favoreciendo la indagación (si se quejan sobre tener que escribir mucho, es un buen momento para mencionar los conceptos que se vienen trabajando y preguntarles si conocen la

noción de estribillo). Un detalle que es importante para que la actividad resulte atractiva es la elección de la canción a escuchar y transcribir. En el cuaderno se ejemplifica con la canción "La Tarara", de un poema de García Lorca, pero a muchos jóvenes no les resulta tan atractiva. En algunos cursos hemos utilizado "El niño caníbal" de Luis Pescetti, con gran éxito. Cada docente conoce mejor que nadie a sus estudiantes, y debe elegir una canción que los motive y entusiasme.

Luego de que se trabajó la similaridad entre estribillo y procedimientos, se puede pasar a la siguiente etapa de esta actividad. La misma consiste en proponer una canción que posea pequeñas variaciones en el estribillo, como "Un elefante se balanceaba", que es suficientemente conocida y motivadora. En caso de elegir cambiar la canción, debe procurarse que la misma posea una variación sencilla, ya que el objetivo es descubrir la idea de parámetro como un hueco que varía en cada uso, y no representar todas las variaciones posibles de cualquier canción.

La noción de parámetro, tal cual se explicó en la sección 3.3.1, es crítica en la formación de un programador profesional, pero no tanto en la programación como cultura general. Sin embargo, explicar con cuidado esta noción, y lograr una comprensión básica de su funcionamiento es importante, pues es un ejemplo clarísimo del poder expresivo que permiten lograr las herramientas de los lenguajes de programación, y favorece formas de pensamiento de orden superior a la hora de realizar divisiones en subtareas. La forma didáctica que se propone en el cuaderno para explicar este concepto es comenzar con varios ejemplos similares pero con una variación, "recortar" esta variación para obtener un *"estribillo con agujero"* y luego usar un nombre para designar a dicho "agujero". Claramente este *nombre* no es parte del estribillo original, sino que debe ser reemplazado por un valor específico; este punto es el más complejo de entender para alguien sin conocimientos de programación: que el parámetro no se usa de forma literal, sino reemplazándolo por un valor específico.

Al valor específico que se usa para reemplazar cada uso del parámetro se lo conoce con el nombre de *argumento*. Muchas veces los programadores profesionales y los docentes de programación tienen la tendencia a mezclar estas dos nociones, la de parámetro y argumento, y utilizan una u otra denominación de manera indistinta. Sin embargo, en un curso inicial es conveniente establecer con claridad la diferencia. Por otra parte, el uso de canciones populares, más cercanas a la experiencia cotidiana de casi todos, permite que la noción de parámetro se explique sin tanta carga técnica, facilitando su comprensión. Es importante volver a destacar el valor de la didáctica propuesto en este curso, y su importancia.

Luego de esta actividad puede agregarse la herramienta de *parámetros* junto a la de *procedimientos* en la subcategoría *comandos* (3.3.1) de *herramientas del lenguaje*, y al mismo tiempo en la subcategoría de *expresiones* (3.3.2) se puede agregar el *uso de parámetros* como una expresión válida. Como ya fue mencionado, esta idea de "utilizar" el nombre del parámetro como si fuese un dato es la que produce mayor dificultad a la hora de ser comprendida.

4.11 Secuencia didáctica 11: Parámetros en Scratch

En la secuencia didática 11 el propósito es continuar el trabajo iniciado en la secuencia anterior con parámetros y parametrización, pero ahora utilizando el entorno Scratch. Se propone una única actividad, *"El planeta de Nano"*, para que los estudiantes realicen el proceso de indagación que es parte integral de este curso.

La parte más compleja de trabajar con parámetros en Scratch está en la manera de definirlos a través de la interfaz del entorno. En el cuaderno esto está muy bien explicado, pero es importante destacar que puede constituir un obstáculo, e interferir con la adquisición del concepto de parámetro. En la versión disponible al momento de la confección de esta guía del entorno Pilas Bloques, la interfaz de parámetros es similar a la de Scratch, pero es posible que en versiones posteriores este aspecto sea mejorado.

Como ya se explicó al tratar parámetros en las secciones 3.3.1 y 4.10, la noción de parámetro es compleja de transmitir, y si bien no hace falta profundizar en todos los aspectos técnicos

del tema, es muy importante conocerla por su valor al pensar y construir soluciones mediante programas, ya que favorece formas de pensamiento de orden superior a la hora de realizar divisiones en subtareas. Entonces, ¿qué nociones sobre parámetros pueden ayudar a entender el concepto de la manera que este curso propone? Las ideas importantes, que no necesariamente deben explicitarse sino utilizarse como ayuda cuando , son las siguientes.

- Que un procedimiento parametrizado tiene un "agujero" o hueco que debe llenarse antes de poder utilizarlo.
- Que la forma de llenar dicho agujero es mediante un valor al que se denomina *argumento*.
- Que un procedimiento parametrizado expresa de una sola vez a muchos procedimientos similares (pero diferentes), uniformizando sus diferencias a través del parámetro.
- Que los parámetros de cada procedimiento se identifican mendiante un nombre (y que idealmente ese nombre *no debería ser el mismo* que el de parámetros de otros procedimientos).
- Que el parámetro, al ser utilizado dentro del procedimiento que lo define, describe un valor específico que varía cada vez.
- Que un parámetro no puede utilizarse fuera del procedimiento que lo define.

El caso de la elección de nombres de parámetros es un claro ejemplo de sutilezas en las que no debería entrarse. Si bien parámetros diferentes en procedimientos diferentes pueden utilizar el mismo nombre, esta **no es recomendable**, pues induce a confusiones, y lleva a la necesidad de explicar con detalle la noción de alcance de parámetros, lo que resulta en demasiada complejidad técnica. Todas estas cuestiones no aparecerán necesariamente en esta primera actividad, pero es conveniente tenerlas en cuenta desde el primer momento que se empiezan a trabajar parámetros dentro de un programa.

La actividad de *"El planeta de Nano"* puede resolverse sin parámetros, utilizando 4 procedimientos diferentes, pero cuya única variación es la cantidad de veces que debe repetirse el proceso de moverse y comer una banana. Es usual que los estudiantes manifiesten la molestia que implica realizar cuatro procedimientos tan parecidos (*"¿Los tengo que hacer a los 4? ¡Si son todos iguales!"*), y es ahí cuando debe surgir la relación con la actividad anterior y la idea de parámetro, aplicando la didáctica. Si ningún estudiante trae la cuestión de que "sean todos iguales", puede preguntárseles qué pasaría si hubiese más filas con diferentes cantidades de bananas, y cuando surge, discutir que no es cierto que sean todos iguales pues solamente son parecidos, aunque tienen algo en común y eso es lo que se busca aprovechar al utilizar parámetros. Habiendo descubierto entre todos que los procedimientos son parecidos, aplicar la idea vista en la secuencia anterior (4.10) de crear un "procedimiento con un hueco" o agujero. Una vez que entendieron que la idea es que el procedimiento tiene un agujero, puede preguntarse "¿Cómo podemos mencionar al agujero dentro del programa?", y deducir entre todos que debe usarse un nombre (similar en forma a los que se usan con los procedimientos). Recién ahí es conveniente pasar a ver cómo se hace esto en el entorno.

El punto que las experiencias en el dictado de este tema muestran como más complejo es *vincular mentalmente* el *nombre de un parámetro* con un *dato ausente*, que tomará valor al utilizar el procedimiento con un argumento. Es por eso que es conveniente comenzar con el recurso didáctico de mostrar un procedimiento con agujero *antes* de pasar a definir el procedimiento con el nombre del parámetro. Este procedimiento está bien explicado en el cuaderno, pero se repite aquí por ser una de las claves en la didáctica de parámetros. En el caso de *"El planeta de Nano"*, esto sería escribir en el pizarrón la siguiente secuencia de pasos. Comenzando con varios procedimientos similares que contienen un dato fijo que varía entre ellos

```
definir Comer [4] bananas
  repetir [4]
    Comer banana
```

```
definir Comer [3] bananas
  repetir [3]
    Comer banana
```

eliminar el dato fijo, dejando un agujero (y observar que todos las variaciones producen el mismo código con agujero)

```
definir Comer [ ] bananas
  repetir [ ]
    Comer banana
```

luego agregarle el nombre al agujero, resultando en

```
definir Comer [                    ] bananas
                    cantidadDeBananas
  repetir [                        ]
                  cantidadDeBananas
    Comer banana
```

para terminar con

```
definir Comer [cantidadDeBananas] bananas
  repetir [cantidadDeBananas]
    Comer banana
```

y recién entonces pasar a ver cómo obtenerlo en Scratch. Esta forma de hacerlo apunta justamente a transmitir las ideas que se identificaron anteriormente como importantes.

Una diferencia sutil entre la actividad en Scratch según la presenta el libro y la actividad en Pilas Bloques está en el orden en que aparecen las filas de bananas. En Scratch aparecen en orden decreciente (4,3,2,1) lo que puede llevar a tratar de usar algún tipo de repetición. Pero la repetición necesaria para expresar esa idea está más allá del propósito del curso, y por lo tanto se sugiere evitarla. En Pilas Bloques la actividad usa las filas en desorden (3,1,4,2) y así no surge tan fácilmente la idea de usar algún tipo de repetición, pues la secuencia no es uniforme.

Al decidir que debe ponerse un parámetro, es importante que los estudiantes piensen qué papel juega el dato que desean parametrizar en la solución, y elijan un nombre adecuado que se corresponda con ese propósito. Esto es una aplicación más de la herramienta conceptual de *elección de nombres*, lo cual, como se estableció, contribuye a la *legibilidad*. Es usual que los docentes de programación (y muchas de las personas que programan) elijan, por comodidad, nombres de una sola letra o con nombres poco significativos. ¿Cuál de las siguientes definiciones resultan más entendibles?

- `definir Comer cantidadDeBananas bananas`
- `definir Comer x bananas`
- `definir Comer param bananas`
- `definir Comer p1 bananas`

Y esto es todavía más notable al utilizarlo dentro del procedimiento.

- `repetir cantidadDeBananas { Comer banana }`
- `repetir x { Comer banana }`

- `repetir param { Comer banana }`
- `repetir p1 { Comer banana }`

Como puede observarse, la elección del nombre del parámetro es determinante para aportar legibilidad y contribuir a que el programa resulte fácilmente comprensible. Es por ello que el énfasis que se venía poniendo en la correcta elección de nombres de procedimientos debe hacerse extensible también a los nombres de los parámetros. Es una tentación grande utilizar nombres cortos para acelerar la confección del programa, pero esto es algo que debe buscar evitarse: no es bueno hacerlo ni siquiera con la justificación de "tengo poco tiempo para escribirlo", que es la que suelen dar los docentes al tener que copiarlo al pizarrón.

Al trabajar con parámetros, algunos estudiantes pueden llegar a experimentar problemas sutiles, especialmente si no siguen la regla de utilizar un nombre diferente en cada procedimiento que utilice parámetros. En particular, en Scratch la interfaz de parámetros no es óptima, y pueden producirse errores extraños. Por ejemplo, uno error típico es utilizar un parámetro en un procedimiento diferentes al que lo define. En algunos casos esto es fácil de descubrir, por ejemplo, porque el procedimiento que usa no define ningún parámetro, pero en otros puede llegar a resultar complicado darse cuenta cuál es el error. Esto sucede, por ejemplo, si los dos procedimientos definen parámetros de nombres sutilmente diferentes (por ejemplo, `cantidadDeBananas` y `candidaddeBananas` – observar que uno es con 'D' mayúscula y el otro con 'd' minúscula) pero el que trata de utilizar el parámetro utiliza el parámetro del otro. Esto hará que el programa se comporte de maneras no esperadas, y puede resultar difícil de detectar. La interfaz actual de Pilas Bloques para parámetros es un poco más amigable que la de Scratch, pero también puede resultar compleja para los estudiantes. Se espera que en futuras versiones esto resulte mejorado. El consejo general que se brinda es evitar este tipo de sutilezas reforzando la necesidad de elegir buenos nombres para los parámetros. En caso de que algún estudiante interesado en la programación plantee cuestiones sutiles como la mencionada, se recomienda alentarlo a profundizar el tema por su cuenta, o a estudiar programación de manera profesional, donde podrá entender mejor tales cuestiones.

Con respecto al marco conceptual, debe cambiarse la herramienta *procedimientos* de la subcategoría *comandos* (3.3.1) para completarla a *procedimientos con y sin parámetros*, y también debe agregarse el uso de *parámetros* en la sub-categoría *expresiones* (3.3.2). Además debe mencionarse nuevamente la importancia de la *legibilidad* (3.2.2) a través de la adecuada *elección de nombres*, ahora también aplicable a los nombres de los parámetros además de los procedimientos.

4.12 Secuencia didáctica 12: Dibujando figuras

En esta secuencia didáctica se propone una única actividad principal, *"Dibujando figuras"*, en la que un robot puede dibujar figuras geométricas, y dentro de ella se proponen varios problemas diferentes a ser abordados de a uno por vez. El propósito principal de la actividad es trabajar la utilización de diferentes parámetros numéricos, tales como el largo de un lado y el número de lados. Los primeros problemas presentan el funcionamiento del robot, y plantean problemas simples que pueden ser resueltos sin parámetros, con la intención de que los siguientes vayan complejizando los problemas hasta hacer sentir la necesidad de utilizar parámetros. Una característica interesante de esta actividad es que las primitivas están parametrizadas: la de dibujar y mover se parametrizan con una longitud, y la de girar con los grados del ángulo a girar. Esto constituye una novedad respecto de actividades anteriores, y puede merecer cierta explicación, aunque se ha observado que luego de trabajar parámetros en las primeras actividades, los estudiantes lo aceptan con bastante naturalidad.

En esta actividad Scratch y Pilas Bloques difieren bastante. En Scratch se presenta como una única actividad, y es tarea del docente ir proponiendo los diferentes problemas, mientras que en Pilas Bloques la actividad se divide en varias sub-actividades mucho más guiadas. Así, el

entorno de Pilas Bloques focaliza los conceptos a trabajar, mientras que el entorno de Scratch, al no tener un problema definido y planteado, permite mucho más la exploración por parte de los estudiantes que pueden jugar con diferentes figuras y formas, aunque a riesgo de olvidar que el propósito de la actividad es trabajar el concepto de parámetro. Tanto si se utiliza Scratch como Pilas Bloques, la sugerencia es seguir la secuencia de problemas propuesta en Pilas Bloques (que tiene muchísimas coincidencias con las que se presentan en el cuaderno, aunque hay algunas más, y en este entorno están mejor delimitadas y fueron nombradas de manera individual) antes de pasar a la etapa de exploración. En caso de haber decidido utilizar Scratch durante el curso, mantener el foco en los problemas planteados puede resultar más complejo, pues los estudiantes tenderán naturalmente a intentar explorar, por lo que es conveniente insistir con el hecho de que primero realicen los ejercicios planteados que luego se les dará tiempo para explorar. En caso de haber decidido utilizar Pilas Bloques, puede resultar interesante que luego de haber realizado todas las sub-actividades de parámetros, se presente el entorno Scratch y se les permita realizar la exploración con la actividad *"Dibujando figuras"*.

Las sub-actividades, según los nombres y el orden que se propone en Pilas Bloques, son las siguientes

- *"Dibujando: Al cuadrado"*: el dibujo de un cuadrado solo, sin parámetros.
- *"Dibujando: Rayuela robótica"*: el dibujo de varios cuadrados del mismo tamaño, sin parámetros, para practicar movimiento.
- *"Dibujando: Corto por la diagonal"*: similar al anterior, pero con mayor trabajo en los giros.
- *"Dibujando: Mamushka cuadrada"*: el dibujo de varios cuadrados de distintos tamaños, para presentar la parametrización del tamaño del lado del cuadrado.
- *"Dibujando: Escalera cuadrada"*: (no aparece en el cuaderno) combinación de los anteriores, con varios tamaños y movimiento.
- *"Dibujando: Hexágono"*: el dibujo de un hexágono, sin parámetros, para avanzar hacia el dibujo de figuras genéricas.
- *"Dibujando: Pirámide invertida"*: (no aparece en el cuaderno) el dibujo de un triángulo, sin parámetros, para continuar hacia figuras genéricas.
- *"Dibujando: Figuras dentro de figuras"* (sugerida en el cuaderno, pero no explícita) el dibujo de varias figuras, para practicar la parametrización del número de lados.
- *"Dibujando: La cueva de las estalagtitas"* (no aparece en el cuaderno) un dibujo que combina varias figuras de diferentes tamaños y movimientos, para parametrizar el número de lados y el tamaño del lado.

Como se observa, hay dos sub-actividades donde aparecen los parámetros, *"Dibujando: Mamushka cuadrada"* y *"Dibujando: Figuras dentro de figuras"*, hay otras dos donde se utilizan en combinación con otras características, *"Dibujando: Escalera cuadrada"* y *"Dibujando: La cueva de las estalagtitas"*, y las demás son para presentar las características del robot y la forma de manejar dibujos y movimientos.

Una de las dificultades que se presentan en esta actividad es el tema de moverse de una figura a otra. Si bien en algunos casos, como *"Dibujando: Corto por la diagonal"*, puede usarse una estrategia que no precisa movimientos sin dibujar, la idea es utilizar estrategias que involucren figuras y movimientos, por lo que los movimientos se hacen necesarios. En el entorno Scratch la actividad provee primitivas que permiten moverse sin dibujar, pero en el entorno de Pilas Bloques, no. En ambos casos se presentan dificultades diferentes al respecto. En el caso de Scratch, la dificultad reside en que solo hay primitivas para moverse hacia la derecha y abajo y

en que al moverse, el robot puede quedar en una posición que implique que al dibujar se sale de la zona de dibujo. La primera dificultad implica que para moverse hacia la izquierda o arriba debe usarse un parámetro negativo; esto puede resultar más o menos complejo o poco intuitivo, según la edad y el nivel de manejo matemático que tengan los estudiantes. La segunda dificultad puede llevar a muchas pruebas y errores para posicionar adecuadamente el robot antes de poder completar alguno de los dibujos. En Pilas Bloques no se proveen primitivas de movimiento sin dibujar, y entonces la complejidad pasa por llegar a otro punto del dibujo moviéndose por líneas que ya fueron dibujadas, siendo lo más complejo reposicionar el robot en el ángulo adecuado para que pueda dibujar la siguiente figura en la posición correcta. Aquí se requiere bastante trabajo con ángulos, y según el nivel matemático de los estudiantes puede requerir ayuda del docente, incluso hasta llegar a proveerles la solución correcta. Por ejemplo, en *"Dibujando: Corto por la diagonal"*, el último comando del procedimiento para moverse al siguiente cuadrado debe realizar un giro de 270 grados.

El manejo de ángulos al dibujar figuras constituye una dificultad a abordar, de manera temprana en Pilas Bloques por el tema de los movimientos, o al dibujar figuras de diferente cantidad de lados en Scratch. Si el nivel de madurez en matemáticas y geometría del curso lo permiten, es mucho mejor permitir que los estudiantes exploren esta complicación. Pero si esto se puede transforma en un obstáculo que complique el tratamiento del tema de parámetros, se podría escribir una tabla de correspondencia entre el número de lados y el ángulo (cuadrado, 90 grados; hexágono, 60 grados; triángulo, 120 grados, etc.), e incluso en la misma tabla se puede mostrar que en todos los casos la cantidad de grados es el resultado de dividir 360 por la cantidad de lados, en preparación del ejercicio posterior. Además, para poder realizar la cuenta del ángulo una vez que se parametrizó el número de lados, debe presentarse la noción de *operación entre números*, como lo indica el cuaderno. Esto requiere además agregar la herramienta de *operadores* al marco conceptual, en caso de explicitarlo.

Para cerrar esta actividad se puede utilizar el procedimiento genérico de dibujar figuras geométricas para trabajar la idea de que una circunferencia puede verse como un polígono de infinitos lados. Para esto, puede sugerirse que prueben dibujar figuras de más y más lados (incrementando de a 5 lados es bastante rápido y útil, por ejemplo, 5, 10, 15, 20 lados y luego pasar a 60 y 120 lados; debe tenerse en cuenta achicar el tamaño del lado para que la figura se mantenga de un tamaño razonable dentro de la pantalla), hasta llegar a una figura que en la pantalla se vea como una circunferencia. En este punto se puede discutir al respecto de cómo la visualización del polígono de 120 lados luce como un círculo, y qué sucedería si se continuase incrementando el número de lados. Como se puede observar, la programación puede servir como vehículo para trabajar otros temas no vinculados necesariamente a ella.

Luego de haber terminado la actividad de *"Dibujando figuras"*, el cuaderno propone alguna ejercitación adicional para practicar el uso de parámetros. Estas actividades son *"La fiesta de Drácula"*, *"Salvando la navidad"*, *"Prendiendo las compus (parametrizado)"*, *"Lightbot cuadrado"* y *"El cangrejo aguafiestas"*. En ninguna de las cinco soluciones discutidas en el cuaderno se define un procedimiento principal para solucionar el problema, pero en todas se divide adecuadamente en subtareas con sus procedimientos (parametrizados) correspondientes. En las dos primeras el parámetro es un número, por lo que la solución es similar a actividades anteriores. Pero en los tres siguientes se presenta una nueva dificultad: el valor a parametrizar debe ser una dirección. Puesto que Scratch solo posee valores literales numéricos y de cadenas de caracteres (o *strings* por su denominación en inglés), la única alternativa es codificar la dirección utilizando o bien un número o bien un *string*. Adicionalmente, y puesto que hay cuatro primitivas diferentes para moverse en cada una de las direcciones, es necesario "decodificar" el parámetro para seleccionar la primitiva correspondiente. Esto se realiza mediante alternativas condicionales, pero precisa poder comparar el parámetro con los valores de codificación, a través del operador de igualdad. Esta decodificación es una dificultad adicional que no aporta ningún valor a la parametrización. Por esa razón, con posterioridad a la publicación del cuaderno, estas tres actividades fueron modificadas para que la primitiva de mover reciba un parámetro

de tipo string y realice ella misma la decodificación; las actividades modificadas en Scratch se denominaron *"Lightbot cuadrado (el verdadero)"*, *"Prendiendo las compus (reloaded)"* y *"El cangrejo aguafiestas (el verdadero)"*. En Pilas Bloques las actividades se presentan ya con esta modificación aunque con su nombre original.

Con respecto al marco conceptual, en esta secuencia didáctica no se agregaron herramientas al mismo. Sin embargo, no debe olvidarse remarcar la importancia de las herramientas principales: *estrategia de solución* y *división en subtareas* (3.2.1), *legibilidad* (3.2.2) y la utilización de *procedimientos* para expresar subtareas, con la incorporación de *parámetros* para aumentar la expresividad (3.3.1).

4.13 Secuencia didáctica 13: Juegos y escenarios cambiantes

La secuencia didáctica 13 sirve como presentación de la idea de operadores lógicos, los cuales son necesarios para poder establecer condiciones más complejas que las provistas por un simple sensor. Consiste en 2 actividades, *"La música del loro"* y *"Dino come chatarra"*. Ambas comparten la característica de que debe llegarse hasta cierto elemento para poder llevar a cabo cierta acción, pero la primera es mucho más sencilla, con el propósito de servir de introducción a la nueva herramienta, y la segunda permite combinar todo lo aprendido hasta el momento. Como siempre, es deseable que los estudiantes indaguen por sí mismos cómo hacer que el loro toque el instrumento, y comprueben que les hace falta alguna forma de preguntar si llegaron al instrumento, o sea, si llegaron al tambor *o* llegaron al micrófono. En ese punto es cuando es conveniente presentar la idea de operadores lógicos y el operador lógico de *disyunción*.

Las condiciones a establecer para cumplimentar estas actividades son más complejas que las que se venían utilizando (por ejemplo, "`¿Tocando micrófono o tocando tambor?`")[10], lo cual puede complicar la lectura del programa, y así, su legibilidad. Esto es un signo de que hace falta alguna herramienta más (las *funciones*), pero puesto que Scratch no posee funciones, su presentación debe diferirse hasta utilizar otro entorno, o simplemente evitarse por completo (ver la sección 4.16.2).

Al observar las soluciones que el cuaderno propone y contrastarlas contra el marco teórico y los principios de trabajo propuestos en esta guía, puede observarse que las mismas no cumplen la totalidad de las pautas que aquí se proponen. En el caso de *"La música del loro"*, la solución final no utiliza procedimientos de ningún tipo: ni uno para expresar la totalidad del problema (`Ir a tocar el instrumento`), ni uno para cada una de las subtareas (`Avanzar hasta el instrumento` y `Tocar el instrumento`)). En el caso de *"Dino come chatarra"*, la solución sí utiliza procedimientos para las subtareas, pero, al igual que todas las soluciones propuestas en el cuaderno, no define un procedimiento para expresar la totalidad del problema (`Comer toda la comida chatarra`). Además, en esta actividad puede observarse cierta sutileza al respecto de la ubicación del procedimiento `Volver`: en el cuaderno la misma se ubica como última acción del procedimiento `Comer chatarra`. Sin embargo, ¿por qué debería este procedimiento ser el que vuelva? Sería muchísimo más adecuado que el procedimiento que en el cuaderno se llama `Ir y comer`, se llamase `Ir, comer y volver`, y que luego de `Comer chatarra`, fuese *este* procedimiento el que invocase a `Volver`. Esto es una sutileza que puede omitirse en un curso de cultura general, pero que no debe dejar de tenerse en cuenta en caso de que algún estudiante con mayor interés en la programación lo pregunte.

Esta secuencia didáctica lleva por título "Juegos y escenarios cambiantes", pero en realidad no trabaja con juegos. Surge naturalmente la pregunta de por qué llevaría ese título si no hay juegos involucrados. La razón es que las herramientas aprendidas en esta secuencia son

[10] En Pilas Bloques no están estas actividades, pues el entorno no cubre el tema de interactividad. Pero si se incluyese el tema, el nombre de los sensores sería "`¿Hay micrófono acá?`" y "`¿Hay tambor acá?`" con lo cual la pregunta a realizar sería "`¿Hay micrófono acá o hay tambor acá?`". ¿Cuál de las preguntas resulta más fácil de entender, la de Scratch o la que estaría en Pilas Bloques? Se puede volver a reflexionar en la importancia de la elección de nombres para proveer legibilidad.

necesarias como paso previo para la construcción de juegos, tema que se aborda en la siguiente secuencia.

Con respecto al marco conceptual, debe agregarse la herramienta *operadores* en la subcategoría de *expresiones* (3.3.2), si es que no fue agregada durante la actividad *"Dibujando figuras"*. Además, no deben dejar de trabajarse los conceptos fundamentales propuestos en esta guía: *división en subtareas* (3.2.1) y su expresión mediante *procedimientos* (3.3.1), los cuales deben nombrarse adecuadamente para obtener *legibilidad* (3.2.2).

4.14 Secuencia didáctica 14: Programación de juegos

En esta secuencia didáctica se propone integrar los conceptos trabajados en las secuencias previas mediante la programación de pequeños juegos de computadora muy sencillos. Estas actividades solamente están disponibles en el entorno Scratch; el entorno Pilas Bloques no provee actividades para interacción.

Para programa pequeños juegos hace falta contar con una herramienta adicional que permita que el jugador interactúe con el juego. A un programa que interactúa con un usuario se denomina *programa interactivo*, y a la característica de interacción, *interactividad*. Existen muchas formas de expresar interactividad en los lenguajes de programación, pero las más realistas conllevan normalmente alta complejidad conceptual y requieren el manejo de muchas herramientas adicionales y sutilezas conceptuales que escapan el alcance de este curso. Por eso, la propuesta que realiza el cuaderno consiste en brindar una forma elemental de interactividad mediante un tipo especial de sensores, a los que se denominan *sensores de interactividad*.

Para programar utilizando *sensores de interactividad* alcanza con las herramientas trabajadas durante el curso. Sin embargo se requiere entender un par de detalles respecto de la estrategia de solución para poder hacer que los programas funcionen de la manera esperada. Esto se explica en el cuaderno de manera adecuada aunque sin mencionar la estrategia de solución: cada sensor de interactividad indica si una cierta tecla está presionada *en el momento en que el programa ejecuta la instrucción que utiliza al sensor*, y entonces puede utilizarse como parte de una alternativa condicional para elegir una acción en base a una tecla. Pero dado que el sensor solo indica si la tecla está presionada al momento de ser ejecutado, para que el programa responda a teclas de manera continua debe ubicarse su uso dentro de una repetición condicional. Así, la estrategia de solución para un juego interactivo con este modelo sería

```
definir Jugar el juego
  Preparar el juego (si corresponde)
  Jugar una partida
  Finalizar el juego

definir Jugar una partida
  repetir hasta que gane o pierda
    Analizar teclas y realizar acciones correspondientes
    Modificar estado (si corresponde)

definir Finalizar el juego
  si ganó
   entonces Acciones al ganar
   sino Acciones al perder
```

Observar que la interacción solo se produce mientras se está ejecutando la acción de `Jugar una partida`; luego de que la misma termina ya no se analizan más las teclas. Una vez decidida la estrategia de solución, el trabajo de programación consiste entonces en definir las subtareas involucradas: la condición de ganar o perder, las teclas a analizar y sus acciones asociadas,

la preparación del juego y las modificaciones de estado en caso que las haya y finalmente las acciones a realizar luego de ganar o perder.

Siguiendo con la didáctica por indagación, es conveniente que los estudiantes exploren el funcionamiento de los sensores de movimiento y traten de descubrir por sí mismos la estrategia de solución para jugar los juegos. Sin embargo la experiencia de cursos previos muestra que esto puede resultar demasiado complejo para muchos, por lo que se sugiere que o bien se les vayan dando ayudas al respecto de la estrategia de solución recién discutida, o bien que luego de un tiempo prudencial se explique esta estrategia en común a todos.

Además de la lógica de funcionamiento, cada juego usualmente tiene una apariencia visual característica. Se pueden obtener distintas variantes del juego si se cambia la apariencia visual, pero sin cambiar la lógica de funcionamiento. Dado que el programa es el que establece la lógica de funcionamiento, la apariencia visual debe cambiarse en otro lado. En el entorno Scratch, la apariencia visual se denomina *disfraz*, y se controla desde la pestaña `Disfraces` de cada uno de los elementos del entorno. El cuaderno explica con claridad cómo manejarse con disfraces. Es interesante para los estudiantes poder explorar diferentes disfraces, pues esto les permite personalizar sus juegos y los motiva, pero no debe olvidarse que el objetivo del curso es aprender conceptos de programación, y no realizar juegos sofisticados ya sea desde la estructura lógica o desde la apariencia visual.

Las actividades propuestas para programar juegos son *"El juego más fácil del mundo"* (que está inspirado en un juego denominado *"El juego más difícil del mundo"*, de funcionamiento similar pero con mayores obstáculos), *"En búsqueda de la llave"*, *"El jardín abichado"* y *"La defensa de la Tierra"*. Para cada uno se brindan algunos comentarios y sugerencias sobre sus particularidades.

"El juego más fácil del mundo" La mayor dificultad en esta actividad consiste en encontrar la estrategia de solución. Como se mencionó, si luego de un rato de indagación no logran construirla, se sugiere en primer lugar dar pistas para que puedan seguir intentándolo, y si continúan sin poder, ofrecerles la solución. Observar que el ejercicio no sigue todos los principios aquí enunciados: no define un procedimiento que exprese la totalidad del problema y tampoco define procedimientos para analizar las teclas o el fin de juego.

Por la forma en que se mueven las bolitas azules, es posible construir una solución al juego que permita ganarlo aún si solo se utiliza la tecla de moverse a la derecha. Pero luego de cambiar los disfraces, y dado que la bruja es más grande que el cuadrado rojo, se les hará necesario utilizar todas las teclas para poder ganar.

"En búsqueda de la llave" Esta actividad es prácticamente idéntica a la anterior, con la salvedad de que aquí el buzo solo conoce el comando primitivo de `Avanzar`, por lo que solo podrá utilizarse una única tecla. Nuevamente, el cuaderno no provee un procedimiento para expresar la solución completa ni las partes de la estrategia (analizar teclas y finalizar el juego). Pero además, la "solución" provista contiene errores, ya que no utiliza las teclas para jugar además de que la condición de finalización del juego está incompleta.

"El jardín abichado" En esta actividad se requieren un par de detalles más que en los anteriores. En primer lugar, la mariposa debe continuar avanzando aún cuando no se esté tocando ninguna tecla, por lo que, según la estrategia de solución propuesta antes, el avance de la mariposa debe suceder en la acción de `Modificar estado`. Durante la modificación de estado debe controlarse también si corresponde que la mariposa coma uno de los puntos violetas. Otro detalle interesante es que para establecer la condición de fin de juego debe utilizarse el dato primitivo `puntos` y el operador de comparación de igualdad (`=`). El último detalle a tener en cuenta es que se pide que, cada vez que avanza, la mariposa mueva las alas. Para esto, debe utilizar una primitiva llamada `Siguiente disfraz`; según las ideas de legibilidad discutidas, sería mucho más conveniente que esta primitiva se hubiese llamado `Mover las alas una vez`.

Con respecto a la solución provista en el cuaderno, la misma no utiliza un procedimiento principal para expresar la solución, no utiliza un procedimiento para analizar el fin de juego, y tampoco separa adecuadamente las subtareas de analizar teclas y modificar estado, sino que las mezcla a ambas en el procedimiento `Realizar movimiento`.

"La defensa de la Tierra" En esta actividad la condición de finalización es cuando se pierde, o sea, cuando la vida llega a cero. Para controlar esto, nuevamente hace falta utilizar un dato primitivo (`vida`) y el operador de comparación por igualdad (`=`). La operación de finalización de juego es sencillamente la primitiva correspondiente (pues no hace falta controlar si el juego se ganó o perdió, pues si no perdió, sigue jugando). La vida se resta durante la subtarea de modificación de estado, y el movimiento y disparo se realiza durante la fase de análisis de las teclas.

En el cuaderno el código propuesto no utiliza un procedimiento para expresar la solución completa, y el análisis de teclas se realiza en dos procedimientos distintos en lugar de en uno solo. Por lo demás, sigue los principios aquí expuestos.

Esta secuencia presenta la herramienta *sensores de interactividad*, que debe agregarse en el marco conceptual, en la categoría de *expresiones* (3.3.2). Además, sirve para continuar practicando todas las prácticas que se trabajaron durante el curso: *estrategia de solución* y *división en subtareas* (3.2.1) expresadas mediante *procedimientos* (3.3.1) para los que deben elegirse nombres adecuados a fin de garantizar la *legibilidad* del programa (3.2.2).

4.15 Tema adicional 1: Variables

Una vez que se completó el material que propone el cuaderno, el curso puede enriquecerse agregando algunos temas adicionales. El primer tema que el cuaderno no cubre y puede ser interesante agregar es el tema de recordar información durante la ejecución, lo cual se consigue a través del uso de variables y asignación. El tema de variables no posee actualmente actividades en Pilas Bloques, por lo que de desear incluirlo, debe considerarse pasar al entorno Scratch (si el mismo no fue presentado antes).

En los cursos tradicionales el tema de variables es de los primeros en ser presentados, pues en ellos se hace énfasis en la programación de computadoras como una actividad más vinculada al funcionamiento de las máquinas que a la lógica con la que los programas se conciben. Y puesto que las computadoras poseen unidades de memoria, las variables son la herramienta natural para manejarlas. En este curso, en cambio, la propuesta es poner el foco en la forma en que las personas piensan los programas, y consecuentemente las variables solamente sirven como una herramienta más, necesaria cuando hace falta recordar alguna información. Como se puede comprobar al revisar todas las actividades realizadas hasta este momento, ninguna de ellas precisó en ningún momento la utilización de variables, puesto que no hacía falta recordar nada.

El concepto de variable admite muchas variaciones y sutilezas, y por ello es un concepto complejo, adecuado para un curso profesional de programación. Los programadores que aprendieron con cursos tradicionales tienden a pensar que sin variables no se puede programar, pero esto dista muchísimo de ser cierto. Por eso, al dictar este curso de didáctica de la programación, pensando a la programación como un tema de cultura general, los estudiantes más reacios al enfoque serán aquellos que ya tienen conocimientos de programación. Es importante no caer en discusiones técnicas con estos estudiantes durante la clase, ya que es posible que los demás estudiantes se sientan intimidados por las complejidades y se inhiban, retrasando o impidiendo su posterior participación.

Para simplificar las complicaciones, al tratar el tema de variables, lo ideal es trabajar solamente con variables locales a cada procedimiento, ya que esto evita problemas con la comprensión acerca de efectos laterales (un procedimiento que modifica la variable que define y

utiliza otro procedimiento), la noción de alcance (dónde tiene validez utilizar una determinada variable), tiempo de vida (cuándo una variable comienza a existir y cuándo deja de tener sentido como valor). Es muchísimo mejor trabajar nociones como el *propósito* de una variable, la adecuación de su *nombre* a tal propósito y la necesidad de utilizarla como *memoria* que permite recordar información entre diferentes momentos del programa.

En el entorno Scratch, sin embargo, las variables son globales, esto es, son accesibles desde cualquier procedimiento en cualquier momento. Es por ello que de trabajar con variables en Scratch resulta más complicado que hacerlo en otros entornos.

Las operaciones que Scratch provee para el trabajo con variables, ubicadas en la zona de bloques llamada "`Datos`", son las siguientes

- Definir una nueva variable (global).
- Darle un valor inicial a una variable, *inicializar*, mediante el bloque "`fijar variable a número`".
- Modificar su valor, *asignar*, mediante el bloque "`cambiar variable por número`".
- Mostrar una variable, mediante los bloques "`mostrar variable variable`" y "`esconder variable variable`" (y también utilizando el bloque "`decir`" de la zona "`Apariencia`").

El bloque "`cambiar variable por número`" puede generar confusiones, especialmente en personas con conocimiento previo de programación, pues su funcionamiento *incrementa* el valor previo de la variable por la cantidad dada (es el equivalente del operador de incremento del lenguaje C, "`variable += número`").

Las cuestiones que pueden surgir al trabajar con variables son, al momento de definir la variable (pues la idea de elemento global puede resultar confusa, y la elección del nombre de la variable puede ser descartada como poco importante), al momento de darle a la variable un valor inicial (si no se realiza, en Scratch toma el valor final de la variable en la última ejecución, lo que puede dar lugar a confusiones), y al momento de utilizar la variable como contador (por la confusión dada por el funcionamiento del bloque `cambiar`).

Un punto importante a destacar al utilizar variables, es que toda variable que se defina y utilice debe tener un *propósito*, y el mismo debe quedar reflejado en el nombre que se elija para la misma, siguiendo los lineamientos de *legibilidad* discutidos en la sección 3.2.2. Muchos programadores profesionales y docentes de programación minimizan este aspecto, utilizando cualquier nombre para las variables, complicando posteriormente la comprensión del programa. También suele suceder que utilizan la misma variable para más de un propósito, a pesar de que su nombre indica otra cosa, como resultado de supuestas consideraciones de "eficiencia" que actualmente casi no tienen validez. Se recomienda no caer en estas prácticas, y prestar especial atención a la legibilidad del programa. Por ejemplo, si la variable va a representar el número de casilleros que el robot se movió hasta el momento, un buen nombre sería `cantidadDeMovimientosHastaAhora` y no simplemente `cant` o `cantMovs`; observar que el componente `HastaAhora` del nombre de la variable permite entender con mayor facilidad por qué la misma se inicializa en 0, y por qué debe cambiarse por 1 cada vez que se mueve.

Las actividades recomendadas para trabajar con variables en el material adicional son *"Lightbot recargado"* y *"Super Lightbot 1 - El regreso"*. Además de estas dos, que pueden resultar pocas, se puede trabajar con las siguientes: *"Dino come chatarra"*, *"El mono cuenta de nuevo"* y *"Fútbol para robots"*. En cada una de las actividades se propone que además de la tarea original se realice alguna tarea adicional, que implica contar ciertas cantidades, para lo cual deben utilizarse variables (globales). Así, en *"Lightbot recargado"* deben contarse el número de luces prendidas, en *"Dino come chatarra"* deben contarse la cantidad de hamburguesas y de donas que comió, en *"El mono cuenta de nuevo"* y en *"Fútbol para robots"* debe contarse la cantidad de veces que el personaje se movió (hacia adelante, o en cualquier sentido).

La actividad *"Super Lightbot 1 - El regreso"* también requiere contar el número de movimientos, pero a diferencia de las anteriores, donde contar era parte de la tarea, aquí es solamente necesario para llevar a cabo la consigna de volver al inicio. Dado que no se conoce la longitud a recorrer, debe contarse mientras se avanza, para saber cuánto retroceder luego. Es interesante que esto sea descubierto por los estudiantes como parte de su indagación.

En caso de trabajar el tema de variables, en el esquema del marco conceptual debe agregarse la herramienta *asignación de variables* en la categoría de los *comandos* (3.3.1), y *uso de variables* en la categoría de las *expresiones* (3.3.2). Además, debe trabajarse el concepto de *legibilidad* y elección de nombres (3.2.2).

4.16 Tema adicional 2: otros entornos de trabajo

Otro tema adicional que el cuaderno no propone, pero que es interesante considerar, es utilizar otros entornos de trabajo además de Scratch (o Pilas Bloques). El propósito de esto es mostrar que los conceptos adquiridos (dentro del marco conceptual propuesto) no son específicos de un lenguaje o un entorno de trabajo particular, sino que los mismos son aplicables en cualquier entorno de programación. Esto constituye un recurso pedagógico importante para la fijación de las abstracciones que se busca que el estudiante incorpore.

El entorno principal que se propone como adicional a Scratch es Alice, un software de programación con ambientes 3D creado en la Universidad de Carnegie Mellon. El principal atractivo de Alice es justamente que sus escenarios son en 3 dimensiones, lo que permite a los estudiantes crear programas de juegos más parecidos a los que ellos reconocen como usuario, y entonces la motivación y la fijación de conceptos es mayor. La dificultad de utilizar Alice es que maneja más conceptos que los propuestos en este curso, que la nomenclatura utilizada no es necesariamente la misma que la presentada aquí, y que la interfaz puede resultar poco intuitiva, especialmente en el tratamiento de objetos en 3D y de la construcción de los bloques.

Además de Alice, existen otros entornos que se pueden proponer. En las próximas secciones se comentan cada uno de estos posibles entornos. Un aspecto a tener en cuenta si se decide incorporar otros entornos de trabajo al curso es que los mismos deben instalarse en una versión compatible con las computadoras de los estudiantes. Este trabajo puede consumir bastante tiempo, por lo que se recomienda no darlo por sentado y planificarlo como parte de las clases.

4.16.1 Otros entornos además de Alice

Si en lugar de utilizar Scratch durante el curso se utilizó Pilas Bloques, el uso de Scratch como entorno adicional también es posible. La ventaja de utilizar Scratch como complemento a Pilas Bloques, es que mientras que éste provee un entorno controlado y circunscripto a los conceptos que se proponen desde el curso, aquel propone un ambiente abierto donde pueden explorarse muchos otros aspectos (desde variar los escenarios de las actividades, hasta programar sus propias actividades). La desventaja de Scratch, al igual que Alice, es que al ser abierto posee muchas más herramientas y conceptos de los que el curso busca transmitir, y en ocasiones algunos de los conceptos involucrados son complejos o poseen interrelaciones no triviales con otros existentes. Pero si los conceptos básicos fueron adecuadamente transmitidos, los estudiantes muchas veces descubren otros conceptos asociados a las herramientas nuevas que provee Scratch y los utilizan. Alentarlos a realizar su propia indagación del ambiente Scratch como uno de los posibles cierres del curso es extremadamente deseable.

Finalmente, si además de mostrar cómo los conceptos aprendidos durante el curso pueden utilizarse en otros entornos se desea dar un puntapié inicial en algunos aspectos de programación más realista, puede utilizarse otro entorno de trabajo, basado en el lenguaje de programación GOBSTONES (`http://www.gobstones.org`). El lenguaje GOBSTONES y su entorno de trabajo fueron creados en la Universidad Nacional de Quilmes, y están disponibles de manera libre para ser utilizados por la comunidad en general; en el sitio mencionados puede encontrarse

material didáctico (al momento de publicación de esta guía, un libro, una charla y notas de clases) además del entorno para trabajar. Las características distintivas del entorno actual de GOBSTONES son que los programas se escriben utilizando texto (en lugar de bloques) tal y como lo hacen los programadores profesionales, y además que utiliza un escenario basado en un tablero y bolitas, lo que permite trabajar la noción de *representación de la información* y de *abstracción* al representar diversos elementos utilizando combinaciones de bolitas. Dada la naturaleza textual de los programas en GOBSTONES, deben considerarse varias problemáticas nuevas, como el manejo de errores de sintaxis y la adecuada indentación de código (que los bloques permiten evitar completamente). También, debido a la necesidad de representar diversos dominios utilizando bolitas, aparece la problemática de entender cómo modelar un escenario utilizando las bolitas, y conseguir que los estudiantes imaginen las correspondencias necesarias. En este último punto puede servir mucho la presencia de una herramienta que provee el entorno de GOBSTONES denominada *vestimentas*, que permite que ciertas combinaciones de bolitas sean visualizadas como un dibujo provisto por la persona que programa; de esta manera se pueden representar muchas de las actividades de este curso. Finalmente, existe una extensión del lenguaje GOBSTONES, denominada XGOBSTONES, que agrega un tratamiento básico de estructuras de datos, pero lamentablemente aún no hay material didáctico disponible sobre la misma. En resumen, GOBSTONES provee un entorno con varias complejidades adicionales, pero que puede resultar interesante para aquellos estudiantes o cursos que deseen continuar profundizando el aprendizaje de la programación.

4.16.2 Alice

El material adicional para incluir Alice como parte del curso podrá encontrarse en `http://program.ar/material-curso-docentes`. Allí se presentan una serie de actividades que siguen las ideas trabajadas con las actividades del cuaderno y que utilizan los conceptos trabajados.

En el entorno Alice se utilizan bloques para construir los programas, al igual que en Scratch y Pilas Bloques, pero existen varias diferencias importantes. La primera es que el escenario es un ambiente en 3 dimensiones, y que pueden agregarse personajes al mismo. En la mayoría de las actividades propuestas esto no es necesario, pero de usarse, puede representar una dificultad adicional a tener en cuenta, porque hay algunas cuestiones no obvias a la hora de hacerlo. La segunda es que la sintaxis, por más que esté mayormente en castellano (no todas las construcciones han sido traducidas correctamente), es más rara, puesto que cada uno de los personajes del escenario constituye una máquina en sí misma con sus propias instrucciones primitivas, y entonces al dar instrucciones debe indicarse a qué máquina se le envía la misma. Por ejemplo, en la actividad *"Lightbot 3D"*, para indicar al robot que avance debe usarse el comando `lightBot.avanzar`. Para conseguirlo, debe seleccionarse el objeto `lightBot` en la sección de objetos, y luego el comando avanzar de la sección métodos de dicho objeto. Otras cuestiones de interfaz que pueden confundir son que algunos comandos poseen menúes desplegables para seleccionar elementos adicionales, que la repetición simple se denomina enigmáticamente `lazo`[11], que para descartar un comando que ya no se desea debe tirárselo a la papelera, y que para copiar un comando debe utilizarse el portapapeles. Finalmente, una fuente más de confusión es que cada objeto viene munido de un conjunto importante de comandos básicos predefinidos por Alice, pero que no son necesarios en las actividades.

Como se puede observar, la interfaz es más compleja, y requiere ser trabajada. El material adicional provee una guía adecuada para esto.

Otra fuente de dificultad en Alice proviene del hecho de que la repetición condicional no utiliza la forma `repetir-hasta` sino la forma `repetir-mientras`. Esta diferencia fue discutida en esta guía en la sección 4.9, y si no se la trabajó antes, es un buen momento para hacerlo.

[11] La traducción `lazo` corresponde al inglés `loop`, o sea, `ciclo` o lazo, que es una de las formas con las que se suele nombrar a la repetición simple.

Un concepto que no es posible trabajar en Scratch, ya que ese entorno carece de esa noción, pero que es posible trabajar en Alice (y también en GOBSTONES) es la noción de *función*. Las *funciones* son una herramienta similar a los procedimientos, pero en la categoría de las expresiones: mientras un procedimiento sirve para nombrar una tarea compleja expresada por medio de una combinación de comandos, una función sirve para nombrar una forma compleja de expresar cierta información mediante expresiones. Esto tiene sentido, por ejemplo, cuando las condiciones que deben utilizarse para detectar el final de una repetición condicional son extremadamente complejas. En la actividad *"Mantis relijosa"* debe utilizarse la condición `mantis.cubos comidos == 4` para detectar si el juego terminó; en lugar de ello, podría definirse una función `mantis.comió todos los cubos`, cuyo significado fuese el mismo. Las funciones permiten mejorar muchísimo la legibilidad de un programa, al expresar mediante nombres adecuados las condiciones necesarias, y además permiten profundizar aún más la herramienta conceptual de división en subtareas, al permitirle a la persona que escribe el programa dividir en subtareas también en la categoría de las expresiones. De todas maneras, este es un concepto más avanzado que puede elegirse no incluir, aún habiendo presentado el entorno Alice.

Con respecto a las actividades propuestas para el entorno Alice (*"Lightbot 3D"*, *"Super Lightbot 3D"*, *"Parejas de Garfields y Odies"*, *"El Zombie contraataca"*, *"Mantis Remiedosa"*, *"Triceratops a la carrera"* y *"Mantis Relijosa"*), el material adicional indicado provee todas las indicaciones necesarias para su adecuado tratamiento.

Con respecto al marco conceptual se puede indicar cómo los conceptos que se fueron trabajando a lo largo del curso aparecen también en Alice de manera natural, y que a pesar de las diferencias de sintaxis, no resulta demasiado complicado aprender a utilizar este nuevo entorno de trabajo. En caso de haber presentado la noción de *funciones*, la misma puede agregarse en la categoría de *expresiones* (3.3.2). De todas maneras, no debe dejar de hacerse énfasis en que los verdaderos conceptos importantes son la *estrategia de solución* (3.2.1) y la consecuente *división en subtareas* expresadas mediante *procedimientos* (3.3.1) y *funciones*, y la *legibilidad* del programa (3.2.2) conseguida mediante una buena *elección de nombres* de procedimientos, parámetros, variables y funciones.

5 Conclusiones

El propósito de este documento fue presentar un marco conceptual para servir de guía y estructura en el curso de didáctica de la programación que propone la Fundación Sadosky. Este marco conceptual resulta fundamental para que los estudiantes puedan incorporar los conceptos principales de la programación de manera estructurada y coherente, y no simplemente como actividades sueltas o ideas interesantes pero no conectadas. Esta integración resulta en un aprendizaje más significativo por parte del estudiante, lo que le permite luego aprender nuevos entornos o lenguajes de programación con mayor facilidad, comprobando que el curso sirve a su propósito de enseñar a programar. Esto puede comprobarse y explicitarse en el curso si hay tiempo suficiente, presentando diversos entornos a los estudiantes (Pilas Bloques, Scratch, Alice, GOBSTONES). Por otra parte, la presencia de un marco conceptual sirve para tener elementos de juicio acerca de la calidad de una solución, lo que permite al docente, y a los estudiantes que pueden hacer autoevaluación, de una guía inestimable para validar su trabajo y para enseñar a otros en caso necesario. Además, permite focalizar la contribución de cada actividad al cuerpo de conocimientos sobre programación, por lo que redunda en un mayor aprovechamiento de los mismos.

Hoy en día ya resulta obvio decir que es importante aprender a programar para no ser un analfabeto moderno, pues la programación está presente en muchísimos aspectos de nuestras vidas. Pero aprender a programar de manera adecuada es un desafío que los métodos tradicionales no alcanzan a cubrir de manera amplia, cerrando la programación a un grupo selecto de profesionales. Y esto no es deseable, sino justamente todo lo contrario: la programación debe

poder ser aprendida por todos. Por eso es importantísima la iniciativa Program.AR y el trabajo de la Fundación Sadosky en este sentido: sigue principios sólidos, pero al mismo tiempo no se olvida del público al que va dirigido.

Yo estoy convencido que este objetivo, el de enseñar a programar más allá de las modas, los entornos o los lenguajes particulares, es uno de los pilares fundamentales sobre el que todo país debe basar su educación y su desarrollo. Y también que hacerlo nosotros mismos, con herramientas y software libres, y sin que esto se constituya en un negocio, es básico para que sea realmente masivo y exitoso. Me siento honrado de poder ser un protagonista más en esta enorme aventura que es la búsqueda de formas de enseñar a programar de manera adecuada.

6 Agradecimientos

Agradezco a Pablo Factorovich y Gabriela Arévalo por su colaboración, su paciencia y su guía a la hora de definir este documento. Sin ellos para coordinar mi trabajo, establecer los objetivos y el alcance y corregir los numerosos borradores, este trabajo no habría sido posible. Agradezco a Daniel Ciolek por haber provisto la idea de escribir un programa para bailar la polka y a todos mis colegas que me apoyan con sus discusiones, comentarios e ideas.

Referencias

[AK01] Anderson, Lorin W. and David R. Krathwohl (editors): *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives.* Allyn and Bacon, 2001. ISBN 978-0-8013-1903-7.

[Cha11] Chang, J. M.: *A practical approach to inquiry-based learning in linear algebra.* International Journal of Mathematical Education in Science and Technology, 42:245–259, 2011. First published on: 29 November 2010. DOI: 10.1080/0020739X.2010.519795.

[Dos15] Dostál, J.: *Inquiry-based instruction: Concept, essence, importance and contribution.* PhD thesis, Palacký University, Olomouc, Czech Republic, 2015. ISBN 978-80-244-4507-6, doi 10.5507/pdf.15.24445076.

[FSO15] Factorovich, Pablo y Federico Sawady O'Connor: *Actividades para aprender a Program.AR.* Fundación Sadosky, EBook, 2015. ISBN: 978-987-27-4161-7.
URL: `http://programar.gob.ar/descargas/manual-docente-descarga-web.pdf`.

[ML13] Martínez López, Pablo E.: *Las Bases Conceptuales de la Programación. Una nueva forma de aprender a programar.* El autor, EBook, diciembre 2013. ISBN: 978-987-33-4081-9.
URL: `http://www.gobstones.org/bibliografia/Libros/BasesConceptualesProg.pdf`.

[SGW11] Sampson, V., J. Grooms, and J. P. Walker: *Argument-driven inquiry as a way to help students learn how to participate in scientific argumentation and craft written arguments: An exploratory study.* Science Education, 95:217–257, 2011. doi: 10.1002/sce.20421.