

UNIDAD 2: ALGORITMOS, CONCEPTOS BÁSICOS

¿QUÉ ES UN ALGORITMO?

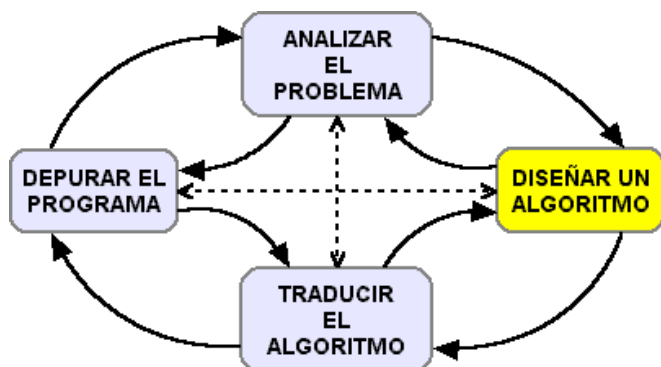


Ilustración 2-1: Segunda fase del ciclo de programación.

Luego de analizar detalladamente el problema hasta entenderlo completamente, se procede a diseñar un algoritmo (trazar un plan) que lo resuelva por medio de pasos sucesivos y organizados en secuencia lógica. El concepto intuitivo de algoritmo (procedimientos y reglas) se puede encontrar en procesos naturales de los cuales muchas veces no se es conciente. Por ejemplo, el proceso digestivo es un concepto intuitivo de algoritmo con el que se convive a diario sin que haga falta una definición “matemática” del mismo. Tener claro el proceso digestivo, no implica que los alimentos consumidos nutran más. La familiaridad de lo cotidiano impide a las personas ver muchos algoritmos que se suceden a su alrededor. Procesos, rutinas o biorritmos naturales como la gestación, las estaciones, la circulación sanguínea, los ciclos cósmicos, etc, son algoritmos naturales que generalmente pasan desapercibidos.

La rama del saber que mayor utilización ha hecho del enfoque algorítmico es las matemáticas. Durante miles de años el ser humano se ha esforzado por abstraer la estructura de la solución de problemas con el fin de determinar claramente cuál es el camino seguro, preciso y rápido que lleva a esas soluciones. Son abundantes los ejemplos: máximo común divisor, teorema de Pitágoras, áreas de figuras geométricas, división, suma de números fraccionarios, etc. Todos estos algoritmos matemáticos independizan los datos iniciales del problema de la estructura de su solución, lo que permite su aplicación con diferentes conjuntos de datos iniciales (variables).

EJEMPLO

Consideremos el algoritmo de Euclides para hallar el Máximo Común Divisor (MCD) de dos números enteros positivos dados. Obsérvese que no se especifica cuáles son los dos números, pero si se establece claramente una restricción: deben ser enteros y positivos.

ALGORITMO EN SEUDOCÓDIGO

Paso 1: Inicio.

Paso 2: Leer los dos números (“a” y “b”). Avanzar al paso 3.

Paso 3: Comparar “a” y “b” para determinar cuál es mayor. Avanzar al paso 4.

Paso 4: Si “a” y “b” son iguales, entonces ambos son el resultado esperado y termina el algoritmo. En caso contrario, avanzar al paso 5.

Paso 5: Si “a” es menor que “b”, se deben intercambiar sus valores. Avanzar al paso 6; si “a” no es menor que “b”, avanzar al paso 6.

Paso 6: realizar la operación “a” menos “b”, asignar el valor de “b” a “a” y asignar el valor de la resta a “b”. Ir al paso 3.

Investigaciones realizadas en Educación Básica (en ambientes constructivistas) recomiendan incluir la solución de problemas en el currículo de matemáticas de forma que provea oportunidades a los estudiantes para crear sus propios algoritmos y generalizarlos a un conjunto específico de aplicaciones (Wilson, Fernández & Hadaway, 1993). Los estudiantes deben reflexionar sobre sus habilidades de planificación y sobre cómo pueden utilizar esas habilidades en diferentes contextos. Por otra parte, en un estudio sobre Logo (Clements & Meredith, 1992), se concluye que cuando los maestros enfatizaron en la elaboración de un plan para desarrollar un procedimiento matemático (este incluía el uso de estrategias como dividir conceptos grandes en otros más pequeños) encontraron que los estudiantes empezaron a utilizar con mayor frecuencia estrategias de planificación y de dibujo para resolver problemas matemáticos en los cuales no utilizaban Logo.

Dato Curioso

La palabra Algoritmo tiene su origen en el nombre del matemático Persa “Mohamed ibn Musa **al Khwarizmi**” (825 d.C.). Su apellido fue traducido al latín como *Algorismus* y posteriormente paso al español como Algoritmo. Khwarizmi fue bibliotecario en la corte del califa al-Mamun y astrónomo en el observatorio de Bagdad. Sus trabajos de álgebra, aritmética y tablas astronómicas adelantaron enormemente el pensamiento matemático y fue el primero en utilizar la expresión *al-yabr* (de la que procede la palabra álgebra). Su trabajo con los algoritmos introdujo el método de cálculo utilizando la numeración arábiga y la notación decimal.

En el ámbito de la computación, los Algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones, ordenadas secuencialmente y libres de ambigüedad, que debe llevar a cabo un computador para lograr un resultado previsible. Vale la pena recordar que un programa de computador consiste de una serie de instrucciones muy precisas y escritas en un lenguaje de programación que el computador entiende (Logo, Java, Pascal, etc).

En resumen, un Algoritmo es una secuencia ordenada

de instrucciones, pasos o procesos que llevan a la solución de un determinado problema. Los hay tan sencillos y cotidianos como seguir la receta del médico, abrir una puerta, lavarse las manos, etc; hasta los que conducen a la solución de problemas muy complejos.

EJEMPLO

Un procedimiento que realizamos varias veces al día consiste en lavarnos los dientes. Veamos la forma de expresar este procedimiento como un Algoritmo:

1. Tomar la crema dental
2. Destapar la crema dental
3. Tomar el cepillo de dientes
4. Aplicar crema dental al cepillo
5. Tapar la crema dental
6. Abrir la llave del lavamanos
7. Remojar el cepillo con la crema dental
8. Cerrar la llave del lavamanos
9. Frotar los dientes con el cepillo
10. Abrir la llave del lavamanos
11. Enjuagarse la boca
12. Enjuagar el cepillo
13. Cerrar la llave del lavamanos
14. Secarse la cara y las manos con una toalla

EJEMPLO

El ejemplo de cambiar una bombilla (foco) fundida es uno de los más utilizados por su sencillez para mostrar los pasos de un Algoritmo:

1. Ubicar una escalera debajo de la bombilla fundida
2. Tomar una bombilla nueva
3. Subir por la escalera
4. Girar la bombilla fundida hacia la izquierda hasta soltarla
5. Enroscar la bombilla nueva en el plafón hasta apretarla
6. Bajar de la escalera
7. Fin

En términos generales, un Algoritmo debe ser:

- **Realizable:** El proceso algorítmico debe terminar después de una cantidad finita de pasos. Se dice que un algoritmo es inaplicable cuando se ejecuta con un conjunto de datos iniciales y el proceso resulta infinito o durante la ejecución se encuentra con un obstáculo insuperable sin arrojar un resultado.
- **Comprensible:** Debe ser claro lo que hace, de forma que quien ejecute los pasos (ser humano o máquina) sepa qué, cómo y cuándo hacerlo. Debe existir un procedimiento que determine el proceso de ejecución.
- **Preciso:** El orden de ejecución de las instrucciones debe estar perfectamente indicado. Cuando se ejecuta varias veces, con los mismos datos iniciales, el resultado debe ser el mismo siempre. La precisión implica determinismo.

Un aspecto muy importante sobre el cual los estudiantes deben reflexionar es la ambigüedad del lenguaje natural que utilizan para comunicarse diariamente con sus semejantes. La informalidad o formalidad en la

comunicación depende de elementos como vocabulario, uso de comodines en lugar de vocablos precisos, uso de adverbios coloquiales en lugar de adverbios formales, etc. Es fundamental que los estudiantes aprendan a diferenciar entre comunicación informal y comunicación formal, cuya principal característica es la precisión. Los algoritmos no admiten ningún tipo de ambigüedad ya que los lenguajes de programación tienen un vocabulario restringido y preciso. Esto exige la utilización de un conjunto determinado de palabras, mandos o primitivas en cualquiera de los procedimientos que se elaboren.

ACTIVIDAD

Discutir en parejas el ejemplo de la bombilla y proponer algunas mejoras. Luego, un voluntario pasa al tablero y escribe un Algoritmo con participación de toda la clase.

Pensamiento Algorítmico

Cuando se habla de algoritmos, con frecuencia aparecen tres tipos de pensamiento que generalmente se relacionan con ellos y que se utilizan indiscriminadamente como sinónimos: Pensamiento Computacional, Pensamiento Algorítmico y Pensamiento Procedimental. Por lo tanto es importante puntualizar a qué se refiere cada uno de estos pensamientos.

Según Moursund (2006), el pensamiento computacional hace referencia a la representación y solución de problemas utilizando inteligencia humana, de máquinas o de otras formas que ayuden a resolver el problema. El pensamiento algorítmico se refiere al desarrollo y uso de algoritmos que puedan ayudar a resolver un tipo específico de problema o a realizar un tipo específico de tarea. Por su parte, el pensamiento procedimental se ocupa del desarrollo y utilización de procedimientos diseñados para resolver un tipo específico de problema o para realizar un tipo específico de tarea, pero que no necesariamente, siempre resulta exitoso.

Por otra parte y de acuerdo con un reporte del Consejo Nacional de Investigación de Estados Unidos (National Research Council, NRC, 2004), conocido como "Being Fluent with Information Technology", el Pensamiento Algorítmico incluye elementos tales como: descomposición funcional, repetición (iteración y/o recursión), organización de datos (registro, campo, arreglo, lista, etc), generalización y parametrización, diseño por descomposición de un problema en partes más pequeñas y manejables (top-down) y refinamiento.

El Pensamiento Algorítmico está fuertemente ligado al pensamiento procedimental requerido en la programación de computadores; sin embargo, su desarrollo puede conducir a los estudiantes a aproximarse guiada y disciplinadamente a los problemas de forma que este pueda transferirse a otros ambientes diferentes a los de la programación. En pocas palabras, la programación de computadores aporta al ámbito escolar un laboratorio para desarrollar habilidades

indispensables en la vida real del Siglo XXI.

Una diferencia notoria entre un algoritmo y un programa es que el algoritmo incorpora las características estructurales básicas de la computación, independientemente de los detalles de su implementación; mientras que un programa tiene un conjunto específico de detalles para resolver un problema. Se puede observar que una técnica de solución (correspondiente al algoritmo) se puede utilizar en diferentes situaciones problemáticas (correspondiente a los programas). De manera inversa, se espera que una solución exitosa de problemas incorpore procesos generales que son independientes de las situaciones específicas (NRC, 2004). Esto se conoce como experiencias de vida y los estudiantes deben adquirirlas en su paso por la educación básica y media para desempeñarse adecuadamente en su vida diaria.

Este es todo un reto para la educación, reto en el que la programación de computadores puede hacer una contribución positiva. Un programa consiste de uno o más procedimientos con instrucciones paso a paso que pueden ejecutarse en un computador; por lo tanto, utilizar el diseño de procedimientos que solucionen o ayuden a solucionar problemas con diferentes niveles de complejidad es un recurso que puede aprovechar el docente para captar el interés de los estudiantes en actividades de programación. Por ejemplo, asignar la tarea de diseñar un procesador de texto básico (ingreso del texto mediante teclado, mostrarlo en la pantalla y guardarlo en el disco duro) es una tarea relativamente sencilla. Pero el proyecto puede aumentar su complejidad si se añaden funciones para dar formato al texto (fuentes, tamaño y características especiales). Posteriormente el proyecto puede crecer si se agregan funcionalidades para manejar imágenes y tablas. Al igual que en este ejemplo, se pueden diseñar proyectos de clase interesantes para mantener motivados a los estudiantes y cuyas tareas y retos sean progresivos en complejidad; que cada nuevo reto parta de lo construido con anterioridad. En resumen, los procedimientos son un tipo particular de tarea que busca solucionar problemas específicos y al desarrollarlos se ponen en juego los pensamientos algorítmico y procedimental.

David Moursund (2006) se basó en sus propias experimentaciones y en la teoría de los cuatro estados de desarrollo cognitivo planteada por Piaget para proponer un planteamiento que amarra la computación con una escala de desarrollo cognitivo en la que se da bastante protagonismo al desarrollo del pensamiento algorítmico en los niños. Según Moursund (2006) en la etapa de las operaciones concretas los niños empiezan a manipular lógica y sistemáticamente símbolos en un computador y aprenden a apoyarse en software para resolver un rango amplio de problemas y tareas de tipo general. De esta manera, ganan habilidad considerable tanto en la utilización de lenguajes como Scratch y MicroMundos, como en la manipulación de ambientes gráficos. Posteriormente, en la etapa de operaciones

formales, los estudiantes demuestran su inteligencia por medio del uso lógico de símbolos relacionados con conceptos abstractos.

Aprestamiento

Una forma motivadora y divertida de aprestamiento a la programación de computadores y que puede ayudar a los estudiantes a desarrollar los pensamientos algorítmico y procedimental consiste en que ellos realicen actividades con juegos de estrategia como "Sokoban", "Misión Escape", "Tetris" e "Implode", así como ejercicios de Razonamiento Abstracto. En Sokoban se deben llevar las piedras hacia el lugar donde aparecen los prismas y para lograrlo, estas se deben empujar con el personaje teniendo cuidado en los movimientos que se hacen para no bloquear el juego ya que el personaje solo puede empujar una piedra a la vez y no puede moverlas hacia atrás, siempre hacia adelante. Hay disponibles varias versiones de Sokoban para descargar y para jugar en línea.

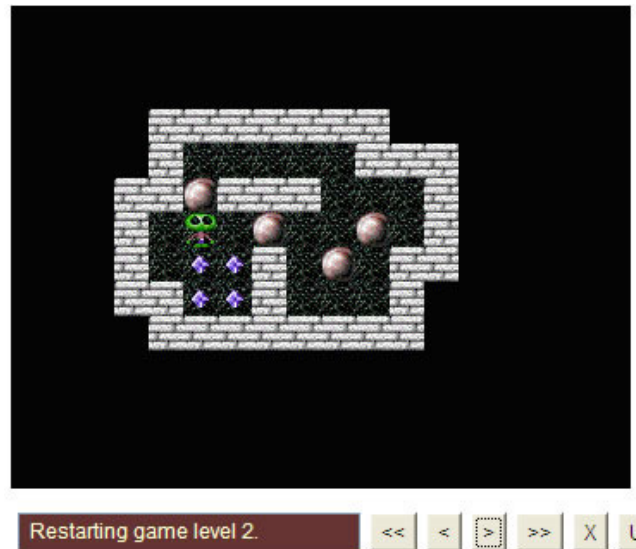


Ilustración 1: El marcianito debe mover la cuatro piedras redondas hasta ubicarlas sobre los rombos morados.

<http://www.matejoven.mendoza.edu.ar/matejue/juegos/sokoban/sokoban.htm>

Por su parte, el juego "Misión Escape" de la serie "Chicos del Barrio" de Cartoon Networks (<http://www.cartoonnetworkla.com/spanish/>) se puede utilizar para mejorar la habilidad de los estudiantes para llevar a cabo tareas en forma ordenada y lógica. En este juego, los participantes deben encontrar la mejor vía de escape a través de la casa del árbol y recorrerla en la menor cantidad de movimientos posibles. Para despejar el camino de objetos hay que seguir las reglas del juego y si no se mueven los objetos precisos, en la dirección correcta y en el orden adecuado, el camino se puede bloquear.



Ilustración 2: Comienzo del nivel tres del juego "Misión Escape" de Cartoon Network. El personaje debe alcanzar la baldosa café que aparece en la parte inferior del cuadrado.

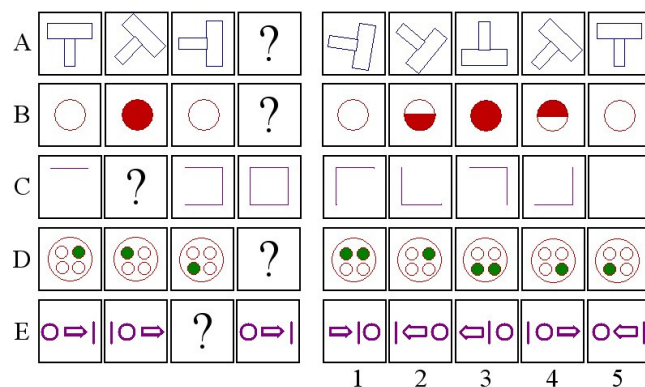


Ilustración 3: El personaje ya ha movido los obstáculos (1, 2, 3) y está a punto de alcanzar la baldosa café que le permite avanzar al nivel siguiente.

La ilustración 2 muestra el comienzo del nivel tres del juego (cada nivel es más difícil que el anterior). El personaje de "Chicos del Barrio" se encuentra en el punto de inicio y debe encontrar el mejor camino para llegar a la baldosa café de la parte inferior del cuadrado. Para lograrlo, debe mover las cajas precisas (marcadas con 1, 2 y 3), en la dirección correcta y en el orden adecuado. En la ilustración 3 se pueden apreciar las cajas movidas y el personaje a punto de alcanzar la baldosa café que le permite avanzar al nivel siguiente.

El **razonamiento abstracto** es otro tipo de actividad de aprestamiento que se puede llevar a cabo con los estudiantes para desarrollar los pensamientos algorítmico y procedimental. El razonamiento abstracto básicamente es un proceso de ordenación de objetos, situaciones o sucesos en secuencias lógicas de acuerdo con algún criterio previamente establecido. Para ello se debe comprender e interpretar los cambios en función de la forma cómo varían las características de interés de los objetos o sucesos estudiados. Todo cambio conduce a una alteración de algún aspecto del objeto, suceso o situación (Sánchez, 1993).

Actividades como la siguiente exige de los estudiantes un alto grado de observación para determinar qué es lo que cambia (figura, forma, posición, etc) y cuál es el patrón de cambio (dirección, tamaño, color, etc):



Adaptado de "Razonamiento Abstracto", Serrano (1998)

Por su parte, juegos como Guido van robot, Tetris, Implode y el mismo Sokoban, además de la versión para Computadores Personales (PCs), ofrecen versiones para los computadores OX de la iniciativa OLPC (One Laptop Per Child). Esto es importante ya que cada día más niños en América Latina disponen de estos equipos y los utilizan como herramienta para el aprendizaje.

ACTIVIDAD

Invitar a los estudiantes a reflexionar sobre el lenguaje que utiliza diariamente para comunicarse con sus padres, hermanos, profesores y compañeros. ¿Utiliza un lenguaje preciso? ¿utiliza vocablos corrientes?

ACTIVIDAD

A diferencia de los seres humanos que realizan actividades sin detenerse a pensar en los pasos que deben seguir, los computadores son muy ordenados y necesitan que el programador les especifique cada uno de los pasos necesarios y su orden lógico de ejecución.

Listar una serie de pasos para realizar una tarea y presentarlos a los estudiantes en forma desordenada para que ellos los ordenen.

Por ejemplo, ordenar los pasos para pescar:

- ___ El pez se traga el anzuelo.
- ___ Enrollar el sedal.
- ___ Tirar el sedal al agua.
- ___ Llevar el pescado a casa.
- ___ Quitar el Anzuelo de la boca del pescado.
- ___ Poner carnada al anzuelo.
- ___ Sacar el pescado del agua.

ACTIVIDAD

Solicitar a los estudiantes que traigan para la próxima clase los siguientes elementos:

- Arroz, lentejas o maíz (medio puñado).
- Una banda de caucho.
- Un vaso plástico.
- Un trozo de papel resistente (15cm x 15cm aproximadamente).

Divida los estudiantes en dos grupos y suministre a un grupo las siguientes instrucciones para elaborar "Maracas":

1. Recortar del papel resistente un trozo más grande que la boca del vaso plástico.
2. Introducir el arroz, las lentejas o el maíz en el vaso (cada elemento produce una sonoridad diferente).
3. Poner sobre la boca del vaso el papel.

4. Fijar el papel al vaso con ayuda de la banda de caucho.
5. Asegurarse que la boca del vaso quede sellada.

Suministre al otro grupo de estudiantes las siguientes instrucciones para elaborar “Maracas”:

1. Recortar del papel resistente un trozo más grande que la boca del vaso plástico.
2. Poner sobre la boca del vaso el papel.
3. Fijar el papel al vaso con ayuda de la banda de caucho.
4. Asegurarse que la boca del vaso quede sellada.
5. Introducir el arroz, las lentejas o el maíz en el vaso (cada elemento produce una sonoridad diferente).

Las instrucciones dadas a ambos grupos son las mismas. Sin embargo, esta actividad ilustra muy claramente la importancia que tiene el orden en que se ejecutan las instrucciones de un algoritmo.

Dato Curioso

En 1936, el lógico y matemático inglés Alan Turing (1912-1954), construyó la primera máquina conceptual como una herramienta matemática para estudiar los procesos algorítmicos. Un cálculo en una máquina de Turing consta de una secuencia de pasos que ejecuta su unidad de control. Si un problema se puede resolver en la máquina de Turing entonces es algorítmico, y recíprocamente si un problema tiene solución algorítmica, entonces se puede resolver en la máquina de Turing.

(Adaptado de ¿Qué es realmente un Algoritmo?, Escuela de Ingeniería, Colombia.)

A continuación se presentan conceptos básicos que los estudiantes deben conocer (y dominar) antes de iniciar el aprendizaje de las estructuras básicas (secuencial, decisión y repetitiva) del lenguaje algorítmico y de programación que abordaremos en la Unidad 3.

REPRESENTACIÓN DE ALGORITMOS

Los Algoritmos se puede expresar de muchas maneras, pero en esta guía se tratarán solo dos formas: Seudocódigo y Diagrama de Flujo. En **Seudocódigo** la secuencia de instrucciones se representa por medio de frases o proposiciones, mientras que en un **Diagrama de Flujo** se representa por medio de gráficos.

EJEMPLO

Elaborar un Algoritmo para calcular el área de cualquier triángulo rectángulo y presentar el resultado en pantalla.

SEUDOCÓDIGO

Paso 1: Inicio

Paso 2: Asignar el número 2 a la constante "Div"

Paso 3: Conocer la base del triángulo y guardarla en la variable "Base"

Paso 4: Conocer la altura del triángulo y guardarla en la variable "Altura"

Paso 5: Guardar en la variable "Area" el valor de multiplicar "Base" por "Altura"

Paso 6: Guardar en la variable "Area" el valor de dividir "Area" entre "Div"

Paso 7: Reportar el valor de la variable "Area"

Paso 8: Final

DIAGRAMA DE FLUJO

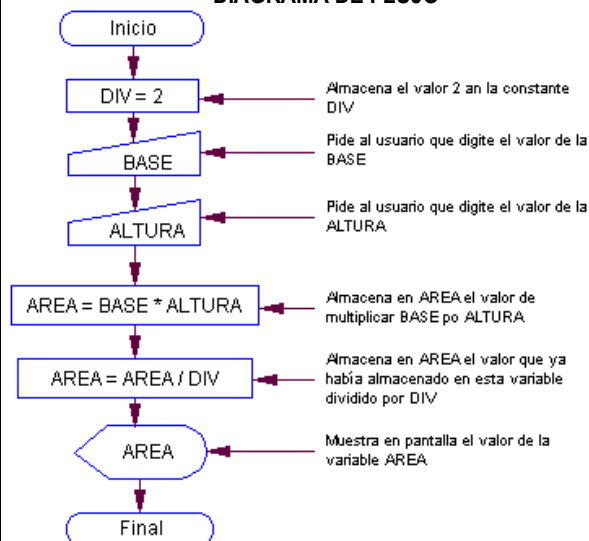


Ilustración 2-4: Algoritmo para calcular el área de cualquier triángulo rectángulo

El pseudocódigo está compuesto por proposiciones informales en español que permiten expresar detalladamente las instrucciones que llevan desde un estado inicial (problema) hasta un resultado deseado (solución). Por lo regular, los algoritmos se escriben por refinamiento: se escribe una primera versión que luego se descompone en varios subproblemas (el número depende de la complejidad del problema) independientes entre sí. Si es necesario se va refinando cada vez las instrucciones hasta que las proposiciones generales en español como las del ejemplo anterior se puedan codificar en el lenguaje seleccionado para hacer la programación (en el caso de esta guía será Logo).

Utilizar Diagramas de Flujo para representar un algoritmo tiene claras ventajas, especialmente cuando son construidos por estudiantes de básica y media. Numerosas investigaciones han mostrado que el Aprendizaje Visual es uno de los mejores métodos para enseñar habilidades del pensamiento. Las técnicas que utilizan formas gráficas para representar ideas e información ayudan a los estudiantes a clarificar su pensamiento, y a procesar, organizar y priorizar nueva información. Los diagramas visuales revelan patrones, interrelaciones e interdependencias además de estimular el pensamiento creativo.

La utilización de Diagramas ayuda a los estudiantes a:

- *Clarificar el pensamiento* : Ellos pueden ver cómo se conectan los procesos y se dan cuenta de cómo estos se pueden organizar o agrupar para darles el orden lógico correcto.
- *Identificar pasos erróneos* : Sobre un diagrama es más fácil identificar los cambios que se requieren para el correcto funcionamiento de un programa de computador que hacerlo sobre el código.

Los Diagramas de Flujo son una de las técnicas más utilizadas para representar gráficamente la secuencia de instrucciones de un Algoritmo. Estas instrucciones están compuestas por operaciones, decisiones lógicas y ciclos repetitivos, entre otros. La solución de un problema puede contener varios conjuntos de instrucciones (procedimientos o métodos) que tienen como finalidad ejecutar cada uno de los procesos necesarios para llegar a la solución de un problema a partir de los datos disponibles (estado inicial).

Las ventajas de diseñar un Diagrama de Flujo antes de empezar a generar el código de un programa (Rojas & Nacato, 1980) son, entre otras:

- Forzar la identificación de todos los pasos de una solución de forma clara y lógica;
- Establecer una visión amplia y objetiva de la solución;
- Verificar si se han tenido en cuenta todas las posibilidades;
- Comprobar si hay procedimientos duplicados;
- Representar gráficamente una solución (es más simple hacerlo con gráficas que mediante palabras);
- Facilitar a otras personas la comprensión de la secuencia lógica de la solución planteada;
- Posibilitar acuerdos con base en la aproximación común a una solución de un problema, resolver ambigüedades o realizar mejoras;
- Establecer posibles modificaciones (resulta más fácil depurar un programa con el diagrama que con el listado del código);
- Agilizar la codificación (traducción) del algoritmo en un lenguaje de programación;
- Servir como elemento de documentación de la solución del problema.

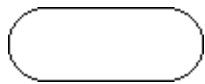
ACTIVIDAD

Basándose en la última actividad planteada en la unidad 1, elaborar un algoritmo en **seudocódigo** para cada uno de los siguientes problemas (se puede utilizar una copia de la plantilla que aparece en el anexo 7):

1. Hallar el perímetro de un cuadrado cuyo lado mide 5 cm
2. Hallar el área de un cuadrado cuyo lado mide 5 cm.
3. Hallar uno de los lados de un rectángulo cuya área es de 15 cm² y uno de sus lados mide 3 cm.
4. Hallar el área y el perímetro de un círculo cuyo radio mide 2 cm.
5. Hallar el área de un pentágono regular de 6 cm de lado y con 4 cm de apotema.

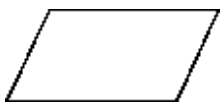
SIMBOLOGÍA DE LOS DIAGRAMAS DE FLUJO

La estandarización de los símbolos para la elaboración de Diagramas de Flujo tardó varios años. Con el fin de evitar la utilización de símbolos diferentes para representar procesos iguales, la Organización Internacional para la Estandarización (ISO, por su sigla en inglés) y el Instituto Nacional Americano de Estandarización (ANSI, por su sigla en inglés), estandarizaron los símbolos que mayor aceptación tenían en 1985. Los siguientes son los principales símbolos para elaborar Diagramas de Flujo:



Inicio/Final

Se utiliza para indicar el inicio y el final de un diagrama; del Inicio sólo puede salir una línea de flujo y al Final sólo debe llegar una línea.



Entrada General

Entrada/Salida de datos en General (en esta guía, solo la usaremos para la Entrada).



Entrada por teclado

Instrucción de entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante.



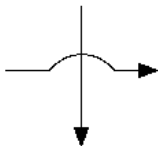
Llamada a subrutina

Indica la llamada a una subrutina o procedimiento determinado.



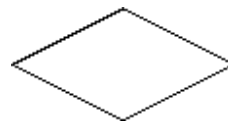
Acción/Proceso General

Indica una acción o instrucción general que debe realizar el computador (cambios de valores de variables, asignaciones, operaciones aritméticas, etc).



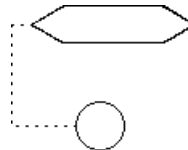
Flujo

Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones.



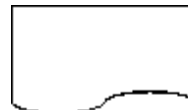
Decisión

Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro.



Iteración

Indica que una instrucción o grupo de instrucciones deben ejecutarse varias veces.



Salida Impresa

Indica la presentación de uno o varios resultados en forma impresa.



Salida en Pantalla

Instrucción de presentación de mensajes o resultados en pantalla.



Conector

Indica el enlace de dos partes de un diagrama dentro de la misma página.



Conector

Indica el enlace de dos partes de un diagrama en páginas diferentes.

El Diagrama de Flujo es una herramienta gráfica valiosa para la representación esquemática de la secuencia de instrucciones de un algoritmo o de los pasos de un proceso. Se recomienda consultar el siguiente componente curricular que apoya la elaboración de Diagramas de Flujo: <http://www.eduteka.org/modulos.php?catx=4&idSubX=124>.

REGLAS PARA LA ELABORACIÓN DE DIAGRAMAS DE FLUJO

Cuando el algoritmo se desea expresar en forma de diagrama de flujo, se deben tener en cuenta algunas reglas o principios básicos para su elaboración (Rojas & Nacato, 1980):

- Poner un encabezado que incluya un título que identifique la función del algoritmo; el nombre del autor; y la fecha de elaboración;
- Sólo se pueden utilizar símbolos estándar (ISO 5807);
- Los diagramas se deben dibujar de arriba hacia abajo y de izquierda a derecha;
- La ejecución del programa siempre empieza en la parte superior del diagrama;
- Los símbolos de "Inicio" y "Final" deben aparecer solo una vez;
- La dirección del flujo se debe representar por medio de flechas (líneas de flujo);
- Todas las líneas de flujo deben llegar a un símbolo o a otra línea;
- Una línea de flujo recta nunca debe cruzar a otra. Cuando dos líneas de flujo se crucen, una de ellas debe incluir una línea arqueada en el sitio donde cruza a la otra (ilustración 2-5);
- Se deben inicializar las variables que se utilicen o permitir la asignación de valores mediante consulta al usuario;
- Las bifurcaciones y ciclos se deben dibujar procurando una cierta simetría;
- Cada rombo de decisión debe tener al menos dos líneas de salida (una para SI y otra para NO);
- Las acciones y decisiones se deben describir utilizando el menor número de palabras posible; sin que resulten confusas o poco claras;
- Si el Diagrama se vuelve complejo y confuso, es mejor utilizar símbolos conectores para reducir las líneas de flujo;
- Todo el Diagrama debe ser claro, ordenado y fácil de recorrer;
- El Diagrama se debe probar recorriéndolo con datos iniciales simples (prueba de escritorio).



Ilustración 2-5: Cruce de líneas de flujo

Los Diagramas se pueden dibujar utilizando lápiz y papel, en cuyo caso resultan muy útiles las plantillas plásticas como la de la ilustración 2-6. Estas descargan al estudiante de la preocupación por lograr uniformidad en el dibujo.

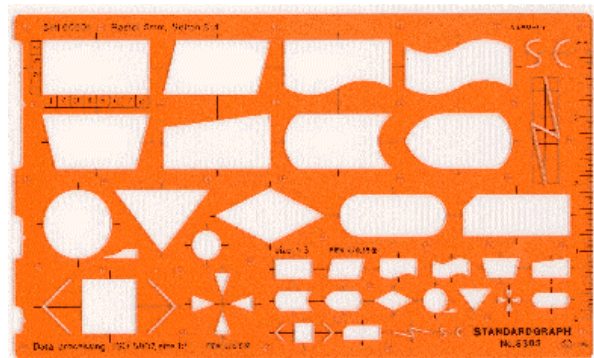


Ilustración 2-6: Plantilla StandardGraph ISO 5807 para la elaboración manual de Diagramas de Flujo.

También existe software especial para elaborar Diagramas de Flujo en forma rápida y fácil. Los programas para esta tarea permiten a los estudiantes:

- Almacenar digitalmente los diagramas construidos;
- Introducirles modificaciones fácilmente;
- Imprimir copias de los diagramas para compartirlos con compañeros o documentar sus trabajos;
- Exportarlos en varios formatos gráficos para utilizarlos en otros programas;
- Alinear y organizar los símbolos automáticamente;
- Agregar colores, tamaño de letra y sombreados para lograr una apariencia profesional;
- Ahorrar tiempo en la modificación de un diagrama ya que no es necesario hacer todo el dibujo nuevamente;

En las siguientes direcciones de Internet se puede encontrar información de software para la elaboración de Diagramas de Flujo:

- Eduteka – Diagramas de Flujo
<http://www.eduteka.org/modulos.php?catx=4&idSubX=117>
- GraFI-co
<http://www.eduteka.org/pdfdir/grafico.rar>
- SmartDraw
<http://www.smartdraw.com>
- WinEsquema
<http://www.softonic.com/ie/27771/WinEsquema>
- Dia Win32 Installer
<http://www.softonic.com/ie/33781/dia>
- DFD 1.0
<http://www.softonic.com/ie/16035/DFD>
- Paraben's Flow Charter
<http://www.paraben.com/html/flow.html>
- Edraw Flowchart
<http://www.edrawsoft.com/flowchart.php>
- Novagraph Chartist
<http://www.tucows.com/preview/289535.html>
- Flow Charting 6
<http://www.patton-patton.com>
- OrgPlus
<http://www.tucows.com/preview/281861.html>
- Antechinus Draw Magic
<http://www.tucows.com/preview/254904.html>

ACTIVIDAD

Basándose en la actividad anterior, convertir los algoritmos elaborados en pseudocódigo en **diagramas de flujo**:

1. Hallar el área de un cuadrado cuyo lado mide 5 cm.
2. Hallar uno de los lados de un rectángulo cuya área es de 15 cm² y uno de sus lados mide 3 cm.
3. Hallar el área y el perímetro de un círculo cuyo radio mide 2 cm.
4. Hallar el área de un pentágono regular de 6 cm de lado y con 4 cm de apotema.

Dato Curioso

En el año 1986, se introdujo en el supercomputador CRAY-2 la cifra $2^{220}+1$, o número de Fermat 20 (que debe su nombre al

matemático Pierre Fermat, 1601-1665), para averiguar si se trataba de un número primo. Al cabo de 10 días, el resultado fue "NO". Este es el cálculo realizado por un computador en el que se ha tardado más en dar una respuesta de "sí" o "no". (Libro Guinness de los Records 2002)

Para avanzar en el tema de los Algoritmos resulta indispensable que los estudiantes comprendan algunos conceptos básicos (variables, constantes, identificadores, funciones, operadores, etc), los cuales serán indispensables tanto para diseñar algoritmos como para traducirlos a un lenguaje de programación, cualquiera que este sea (Logo, Java, Visual Basic, etc).

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Variables

Para poder utilizar algoritmos con diferentes conjuntos de datos iniciales, se debe establecer una independencia clara entre los datos iniciales de un problema y la estructura de su solución. Esto se logra mediante la utilización de Variables (cantidades que se suelen denotar con letras –identificadores- y que pueden tomar cualquier valor de un intervalo de valores posibles).

En programación, las Variables son espacios de trabajo (contenedores) reservados para guardar datos (valores). El valor de una Variable puede cambiar en algún paso del Algoritmo o permanecer invariable; por lo tanto, el valor que contiene una variable es el del último dato asignado a esta. En el Algoritmo de la Ilustración 2-4, "área" es un ejemplo de Variable; en el paso 5 se guardó en ella el resultado de multiplicar "base" por "altura" y en el paso 6 se utilizó nuevamente para guardar el valor de dividir su propio contenido ("área") entre la Constante "div".

MicroMundos ofrece tres tipos de variables: Locales, Globales y de Estado. Las primeras retienen su valor el tiempo que dure la ejecución del procedimiento en el cual se utiliza. Las **variables Locales** se pueden crear con las primitivas *asigna* y *local* o en la línea del título de un procedimiento.

En MicroMundos se utiliza el comando *da* para asignar un valor a una variable o constante. La sintaxis es:

da "nombreVariable valor
que es equivalente a la forma *nombreVariable=Valor* que se utiliza en la mayoría de los lenguajes de programación.

Para utilizar el valor almacenado en una variable o constante se debe anteponer dos puntos (:) al nombre; en

:nombreVariable

los dos puntos significan "no quiero que ejecute el comando nombreVariable; quiero el valor almacenado en nombreVariable".

Las **variables Globales** se crean con los comandos *da* o *nombra*. Estas variables solo pierden su valor cuando se cierra MicroMundos o cuando se borran con el comando *bnombres*.

En Scratch, se debe hacer clic en el botón "Variables" de la paleta de bloques.



Luego se hace clic en el botón "Nueva variable" y se asigna un nombre a la variable, en este caso "Puntaje". Cuando se genera una variable, aparecen los bloques correspondientes a ella. Se puede escoger si la variable es para todos los Objetos (global) o solo para un Objeto (local).

Con el botón "Borrar una variable" se borran todos los bloques asociados con una variable.



Al hacer clic sobre el cuadrado de selección, se empieza a Informar el valor de la variable "Puntaje" en el escenario.



Incrementa la variable en una cantidad determinada (Si se tiene más de una variable, utilice el menú desplegable para seleccionar el nombre de la variable que se desea modificar).



Inicializa la variable a un valor específico.



Muestra el monitor de la variable en el escenario.



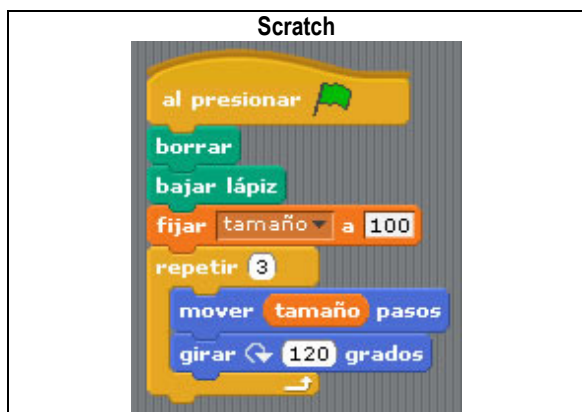
Esta opción esconde el monitor de la variable para que no aparezca en el escenario.

EJEMPLO

MicroMundos
para equilátero :tamaño
limpia
cp
repite 3 [adelante :tamaño derecha 120]
fin

Ahora escriba en el centro de mando de MicroMundos la palabra *equilátero* seguida por un número que representa el tamaño de cada lado del triángulo (ejemplo: *equilátero 70*).

La variable local *:tamaño* creada en la línea de título del procedimiento *equilátero*, contiene el valor 70 mientras el procedimiento se esté ejecutando. Los dos puntos ":" que preceden el nombre de la variable *tamaño* le indican a Logo que no se quiere la palabra tamaño si no el valor que contiene la variable *tamaño*.



En Scratch haga clic en la bandera verde para que se dibuje en el escenario un triángulo equilátero con lado 100. Para dibujar triángulos de tamaños diferentes basta con fijar la variable tamaño a otro valor.

EJEMPLO

Escriba en el área de procedimientos las siguientes líneas de código:

para tipoVariable :valorParámetro
limpia
da "variableGlobal 150
local "variableLocal

da "variableLocal 20

fin

Ahora escriba en el centro de mando de MicroMundos la secuencia de instrucciones en cursiva y debe obtener las respuestas subrayadas:

tipoVariable 70

muestra :valorParámetro

valorParámetro no tiene valor

muestra :variableLocal

variableLocal no tiene valor

muestra :variableGlobal

150

Observe que las variables *:valorParámetro* y *:variableLocal* no conservan su valor por fuera del procedimiento *tipoVariable*, mientras que la variable *:variableGlobal* es de tipo global y conserva su valor (150) por fuera del procedimiento donde fue creada.

En MicroMundos también existen las variables de estado que permiten conocer o modificar los componentes más importantes de una tortuga, un control o una caja de texto.

Constantes

Las Constantes se crean en Logo de la misma forma que las variables y consisten en datos que, luego de ser asignados, no cambian en ninguna instrucción del Algoritmo. Pueden contener constantes matemáticas (pi) o generadas para guardar valores fijos (3.8, "Jorge", etc). En el Algoritmo de la Ilustración 2-4, "div" es un ejemplo de Constante.

EJEMPLO

Las variables y constantes además de tener un Nombre (identificador) para poder referirnos a ellas en los procedimientos, guardan un Valor en su interior.

Nombre (identificador)	Valor
apellido	López
saldo	20000
tamaño	8.5
esTriángulo	SI

ACTIVIDAD

Pedir a los estudiantes que analicen el siguiente ejemplo y que escriban en forma de ecuación las situaciones planteadas.

Ejemplo: El doble de la edad de Carlos Andrés es 32 años:

edadCarlos es la constante donde se guarda la edad de Carlos Andrés;

R/. $2 \times \text{edadCarlos} = 32$

Situaciones:

1. La mitad de un valor (valor1) es 60
2. Cuatro veces un número (número1) equivale a 20
3. Un número (número2) disminuido en 5 es 18
4. El doble (elDoble) del precio de una manzana
5. La mitad (laMitad) del precio de una gaseosa
6. el triple (elTriple) de mi edad

Los valores que pueden tomar valor1, número1 y

número2 (tres primeras situaciones) son constantes: 120, 5 y 23 respectivamente; no pueden tomar otros valores. Además, estas constantes son las incógnitas de las situaciones.

Los valores que pueden tomar elDoble, laMitad y elTriple son variables ya que dependen de un precio o de la edad del estudiante que resuelve el ejercicio. Los valores de estas variables hay que conocerlos para introducirlos en el problema como datos iniciales, pero no son la incógnita. Para ampliar esta actividad, el docente puede plantear nuevas situaciones o pedir a los estudiantes que planteen situaciones similares.

ACTIVIDAD

Pedir a los estudiantes que traigan tres cajas de cartón (del tamaño de las de los zapatos) y marcar cada caja con uno de los siguientes letreros: BASE, ALTURA y DIVISOR. Introducir un papel en blanco en cada una de las cajas. Solicitar a un estudiante del grupo que escriba un valor en cada uno de los papeles guardados en las cajas.

Escribir en el tablero la fórmula para calcular el área de un triángulo rectángulo:

$(\text{Base} * \text{Altura} / 2)$

Luego pedir a otro estudiante que escriba en el tablero los valores de los papeles guardados en cada una de las cajas y aplique la fórmula para calcular el área de un triángulo utilizando esos valores.

Repetir la operación pidiendo a otro estudiante que escriba nuevos valores en el papel de cada una de las cajas, tachando los valores anteriores.

Hacer notar que en los papeles guardados en las cajas marcadas con "BASE" y "ALTURA" se han anotado valores diferentes en cada ocasión. Este es el concepto de variable.

Hacer notar también que en el papel guardado en la caja "DIVISOR" solo se anotó un valor (2) al comienzo del ejercicio y no hubo necesidad de cambiarlo posteriormente. Este es el concepto de constante.

Esta actividad se puede adaptar para reforzar el cálculo de áreas y perímetros de otras figuras geométricas planas.

Contadores

Los contadores en MicroMundos se implementan como una estructura de programación (*da "A :A + 1*) que consiste en almacenar en una variable ("A") el valor de ella misma (:A) más un valor constante (1). Es muy útil para controlar el número de veces que debe ejecutarse un grupo de instrucciones.

En Scratch, se utiliza la instrucción *cambiar ... por ...* para incrementar la variable en una cantidad determinada.



En este caso se almacena en la variable Puntaje el valor que ella tenga en el momento más el valor constante 1.

EJEMPLO

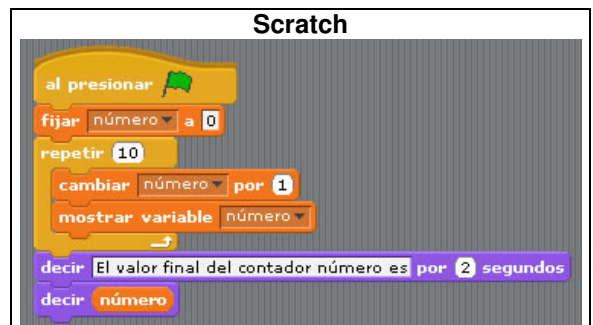
Escribir un procedimiento llamado *contador* para contar los números entre 1 y 10.

```

MicroMundos

para contador
  bnombres
  da "número 0
  repite 10 [da "número :número + 1
              muestra nombres]
  muestra frase
  [El valor final del contador número es ]:número
fin
  
```

Ahora escriba en el centro de mando de MicroMundos *contador*.



Haga clic en la bandera verde de Scratch.

Acumuladores

Estructura muy utilizada en programación (*da "A :A + :B*) y que consiste en almacenar en una variable ("A") el valor de ella misma (:A) más otro valor variable (:B). Es muy útil para calcular sumatorias.

EJEMPLO

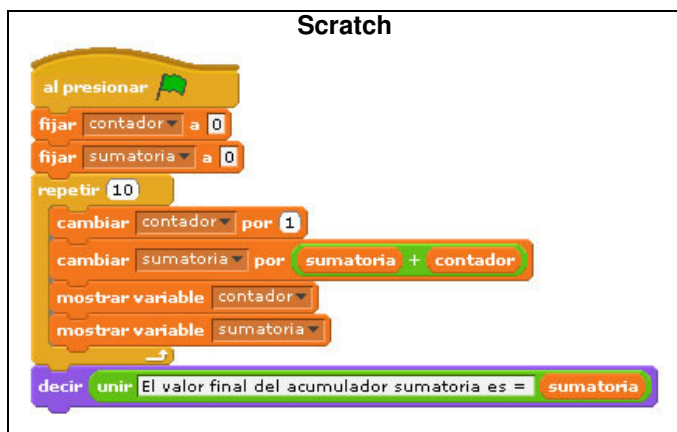
Escribir un procedimiento llamado *acumulador* para calcular la sumatoria de los números entre 1 y 10.

```

MicroMundos

para acumulador
  bnombres
  da "contador 0
  da "sumatoria 0
  repite 10 [da "contador :contador + 1
              da "sumatoria :sumatoria + :contador
              muestra nombres]
  muestra frase
  [El valor final del acumulador sumatoria es ]:sumatoria
fin
  
```

Ahora escriba en el centro de mando de MicroMundos *acumulador*.



Haga clic en la bandera verde de Scratch y el resultado debe ser 2036.

Identificadores

Los identificadores son nombres que se dan a los elementos utilizados para resolver un problema y poder diferenciar unos de otros. Al asignar nombres (identificadores) a variables, constantes y procedimientos se deben tener en cuenta algunas reglas:

- Los nombres pueden estar formados por una combinación de letras y números (*saldoMes*, *salario*, *fecha2*, *baseTriángulo*, etc).
- El primer carácter de un nombre debe ser una letra.
- La mayoría de los lenguajes de programación diferencian las mayúsculas de las minúsculas.
- Los nombres deben ser nemotécnicos, con solo leerlos se puede entender lo que contienen. Deben ser muy descriptivos; no utilizar abreviaturas, a menos que se justifique plenamente.
- Es conveniente utilizar una sola palabra para nombrar páginas, controles, variables, etc.
- No utilizar caracteres reservados (% , + , / , > , etc). MicroMundos admite letras acentuadas (á , é , í , ó , ú). Se debe tener en cuenta que algunos lenguajes de programación no admiten las tildes.
- No utilizar palabras reservadas por los lenguajes de programación.
- Para cumplir con convenciones ampliamente utilizadas (Jiménez, 2002), los nombres de procedimientos, variables y constantes deben empezar con minúscula. Ejemplo, *fecha*, *suma*, etc. Si es un nombre compuesto por varias palabras, cada una de las palabras (con excepción de la primera) deben empezar con mayúscula. Ejemplo: *fechaInicial*, *baseTriángulo*, etc.

El tipo de nombre –identificadores– que se asigne a variables, constantes y procedimientos es muy importante. Cuando los estudiantes dejan de trabajar en un proyecto por varios días, es más fácil para ellos retomar la actividad si los identificadores describen muy bien el contenido de variables, constantes y

procedimientos. Además, el docente podrá ayudarles a revisar y depurar sus programas en forma más eficiente si estos son fáciles de leer (Feicht, 2000).

Palabras reservadas (primitivas)

Todos los lenguajes de programación definen unas palabras para nombrar sus comandos, instrucciones y funciones. Un identificador definido por el usuario no puede tener el nombre de una palabra reservada en MicroMundos.

Algunas palabras reservadas en MicroMundos	
adelante (ad)	muestra
derecha (de)	para
izquierda (iz)	limpia
atrás (at)	rumbo
repite	cp
da	sp

Las palabras reservadas no operan en Scratch ya que todas las instrucciones, incluyendo mandos y reporteros, son bloques de construcción (ver la sección Conceptos básicos de Logo en la Unidad 3). Los estudiantes no deben escribir las instrucciones, solo deben escribir los parámetros en algunas de ellas.

ACTIVIDAD

¿Cuáles de los siguientes identificadores NO son válidos como nombres de Variables en MicroMundos y por qué?

1. númeroX
2. Numero X
3. 7
4. A(45+
5. VII
6. 7mesas
7. sieteMesas

En cuanto a palabras reservadas, Scratch es más flexible que MicroMundos, pues se pueden utilizar como nombres de variables aquellos identificadores que no son válidos en MicroMundos: A(45+, 7, etc.

TIP

Es buena idea asignar, a Variables y Constantes, nombres que indiquen cuál puede ser su contenido. Por ejemplo, a una Variable que contendrá el valor de la base de un triángulo debe asignársele el nombre "baseTriangulo"; "areaCuadrado" a una que guardará el área de un cuadrado; y "radio" a una que contendrá el valor del radio de una circunferencia. Aunque MicroMundos no hace distinción entre mayúsculas y minúsculas, es buena práctica ser consistente a lo largo de todo el algoritmo en su uso (RADIOCIRC ≠ RadioCirc). Esto debido a que la mayoría de lenguajes de programación sí hacen distinción entre mayúsculas y minúsculas.

FUNCIONES MATEMÁTICAS

Cada lenguaje de programación tiene su conjunto de funciones matemáticas predefinidas. Estas se ejecutan haciendo referencia a su nombre. Algunas necesitan, para arrojar un resultado, que se suministre información adicional (parámetros o argumentos). Algunas de las funciones matemáticas más utilizadas en MicroMundos son:

DESCRIPCIÓN	SINTAXIS MicroMundos	Scratch
ARCO TANGENTE. Devuelve el arco tangente (la función inversa de la tangente) de su entrada. Ver tan y cos.	arctan número <i>Ejemplo:</i> cp cumpleveces [i 100] [fx coorx + 1 fy -50 + 2 * arctan :i / 100]	
COSENO. Devuelve el coseno de su entrada. Ver sen y tan.	cos número <i>Ejemplo:</i> cp repite 120 [fy 50 * cos 3 * coorx fx coorx + 1]	
EXPONENCIAL. Devuelve e a la potencia del número.	exp número <i>Ejemplo:</i> cp repite 55 [fx coorx + 1 fy exp coorx / 15]	
LOGARITMO NATURAL. Devuelve el logaritmo natural (el logaritmo en base e) del número. Es el contrario de exp. Ver también log.	ln número <i>Ejemplo:</i> muestra ln 15	
LOGARITMO. Devuelve el logaritmo del número. Ver ln y exp.	log número <i>Ejemplo:</i> muestra log 15	
PI Devuelve la constante PI.	pi <i>Ejemplo:</i> cp repite 360 [ad pi * 100 / 360 de 1] repite 360 [ad pi * 150 / 360 iz 1]	No disponible en Scratch
POTENCIA Devuelve el número1 elevado a la POTENCIA de número2.	potencia número1 número2 <i>Ejemplo:</i> muestra potencia 4 2	Scratch no tiene el operador potencia, sin embargo es fácil programarlo: http://scratch.mit.edu/projects/jualop/752239
RAÍZ CUADRADA. Devuelve la raíz cuadrada de su entrada.	rc número <i>Ejemplo:</i> muestra rc 16	
SENO. Devuelve el seno del número en grados. Ver cos.	sen número <i>Ejemplo:</i> cp repite 260 [fy 25 * sen 6 * coorx fx coorx + 1 / 2]	
TANGENTE. Devuelve la tangente de su entrada. Ver sen y cos.	tan número <i>Ejemplo:</i> cp repite 28 [fy 8 * tan 6 * coorx fx coorx + 1 / 2]	

TIPOS DE DATOS

La mayoría de los lenguajes de programación disponen de una amplia variedad de datos. MicroMundos solo tiene tres tipos de datos: números, palabras y listas.

Números: se utilizan como entradas en las operaciones matemáticas. Cuando se utilizan los signos positivo (+) o negativo (-), estos deben estar pegados al número. MicroMundos acepta tanto el punto como la coma para escribir números decimales (3,14=3.14). Esto es importante tenerlo presente para no utilizar el punto para marcar la separación de miles y millones. Si asignamos a una variable el valor 20.000, MicroMundos guarda en ella el valor 20 y no 20000; si le asignamos 1.345.625 en lugar de 1345625, MicroMundos no aceptará esta notación por tener dos puntos decimales. Por su parte, Scratch solo utiliza el punto decimal; sin embargo, si usted introduce el número 6,2, Scratch lo convertirá automáticamente a 6.2.

EJEMPLO

Los siguientes son números validos en MicroMundos:

- 453
- 19,7
- 19.7
- -14,42
- 856.
- 1E6

El signo debe estar pegado al número: *muestra -3 + 6* da como resultado 3; *muestra -3 + 6* da como resultado el mensaje "necesita más entradas". Scratch no reconoce la notación científica: 1E6.

Palabras: Las palabras están formadas por letras y/o números. Una palabra está delimitada por espacios en blanco; sin embargo, si se quiere tener un texto conformado por dos o más palabras, este debe encerrarse entre barras (|palabra1 palabra2|).

EJEMPLO

Las siguientes son palabras validas en Logo:

- Hola
- x
- 548
- Once-Caldas
- ¿Quién?

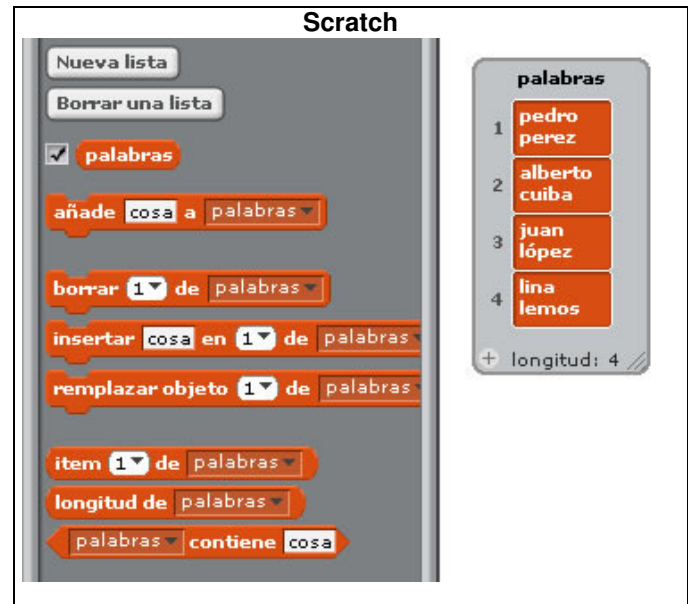
El comando *muestra "hola* da como resultado hola; *muestra "hola"* reporta hola". Varias palabras se deben tratar como una lista.

Listas: una secuencia de palabras puede manipularse igual que una sola palabra mediante el uso de listas. Una lista es una secuencia de palabras separadas por espacios en blanco y encerrada entre corchetes. Las palabras en una lista no necesitan comillas y los espacios en blanco se ignoran.

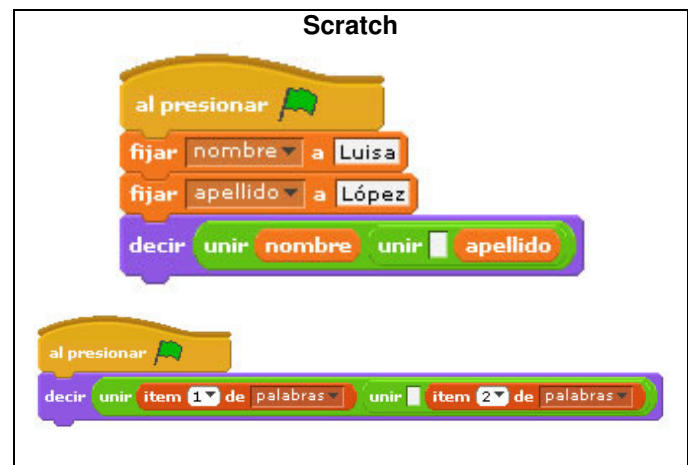
EJEMPLO

Las siguientes son listas validas en MicroMundos:

- [Esta es una lista de 7 elementos]
- [x y z]



Aunque en Scratch se pueden introducir tanto palabras como números en una variable, la operación de suma de dos variables o de elementos de una lista solo opera con números (versión 1.4). Por lo tanto no se pueden concatenar varias palabras para formar una frase con el operador +, debe utilizarse el operador "unir". Varios operadores "unir" se pueden anidar para formar una cadena de varios elementos. En el siguiente ejemplo se requieren tres espacios: uno para mostrar el nombre, otro para separar el nombre del apellido y el tercero para el apellido.



OPERADORES


Son símbolos que sirven para manipular datos. En MicroMundos es necesario dejar un espacio en blanco a cada lado del signo aritmético. Los operadores y las operaciones que se pueden realizar con ellos se clasifican en:

- **Aritméticos:** Posibilitan las operaciones entre datos de tipo numérico y dan como resultado otro valor de tipo numérico. Ejemplo: potencia (potencia); producto (*); división (/); suma (+); resta (-); asignación (=). Este último operador de MicroMundos presenta diferencias con el operador de asignación (=) que utilizan la mayoría de los lenguajes de programación.
- **Alfanuméricos:** Permiten operar con datos de tipo carácter o cadenas. La mayoría de los lenguajes de programación admiten el operador + para realizar la concatenación (unión) de caracteres o cadenas. Ni MicroMundos, ni Scratch tienen esta opción. En Scratch debe utilizarse, para concatenar, el operador &.
- **Relacionales:** Permiten la comparación entre datos del mismo tipo y dan como resultado dos valores posibles: Verdadero o Falso. Ejemplo: igual a (=); menor que (<); mayor que (>).
- **Lógicos:** Posibilitan la evaluación lógica de dos expresiones de tipo lógico. Dan como resultado uno de dos valores posibles: Verdadero o Falso. Ejemplo: negación (no); conjunción (y); disyunción (o).

Orden de evaluación de los operadores

Los computadores ejecutan los operadores en un orden predeterminado. El siguiente es el orden (jerarquía) para ejecutar operadores:

1. Paréntesis (se ejecutan primero los más internos)
2. Signo (-2)
3. Potencias y Raíces (potencia y rc); Productos y Divisiones (* y /)
4. Sumas y Restas (+ y -)
5. Concatenación (+)
6. Relacionales (=, <, >)
7. Negación (no)
8. Conjunción (y)
9. Disyunción (o)

OPERADORES ARITMÉTICOS			
Operador	Operación	Ejemplo	Resultado
potencia ^	Potencia	potencia 4 2 4 ^ 2	16
	Multiplicación	4*2	8

	División	4/2	2
	Suma	4+2	6
	resta	4-2	2
	asignación	da "A 4	Se asigna el valor de 4 a la variable A

La expresión *muestra azar 6 + 1* reporta un resultado diferente al que reporta *muestra (azar 6) + 1*. ¿Por qué? Las operaciones que se encuentran entre paréntesis se evalúan primero; las que tienen el mismo orden de evaluación se ejecutan de izquierda a derecha. Los cálculos aritméticos siempre se realizan antes que cualquier otro mando Logo. Por ejemplo, en la instrucción *muestra azar 6 + 1*, la operación aritmética 6 + 1 se realiza primero que el mando Logo azar y a su vez, el mando azar se ejecuta primero que el mando muestra. En instrucciones como *muestra (azar 6) + 1* hay que tener presente que siempre se deben utilizar pares de paréntesis. En Scratch no está disponible la opción de paréntesis (hasta la versión 1.4).

ACTIVIDAD

Pedir al estudiante que escriba en el Centro de Mando (ilustración 1-2) de MicroMundos las siguientes expresiones y anote en su cuaderno las observaciones sobre los resultados de cada pareja (1 y 2; 3 y 4, 5 y 6):

1. *muestra 243 + 5 - 6 + 86 - 42*
2. *muestra 5 + 86 - 42 - 6 + 243*
3. *muestra 7 + (8 * 16)*
4. *muestra (7 + 8) * 16*
5. *muestra 24 / 4 * 8*
6. *muestra 4 * 8 / 24*

EXPRESIONES

Una Expresión está compuesta por valores, funciones, primitivas, constantes y/o variables, o por una combinación de los anteriores mediante operadores. Son Expresiones:


- Un valor (1.3, "Jorge")
- Una Constante o una Variable (divide, base, área)
- Una función (cos 60, arctan 1)
- Una combinación de valores, constantes, variables, funciones y operadores que siguen reglas de construcción y orden de evaluación de los operadores (cos 60 + 7 - :altura)

Las Expresiones pueden ser:

- **Aritméticas:** Dan como resultado un valor numérico. Contienen únicamente operadores aritméticos y datos numéricos ($\pi * 20 - :X$)
- **Alfanuméricas:** Dan como resultado una serie o cadena de caracteres.
- **Lógicos:** Dan como resultado un valor "Verdadero" o "Falso". Contienen variables y/o constantes enlazadas con operadores lógicos ($A > 0$ y $B \leq 5$).
- **De Asignación:** Estas Expresiones asignan el resultado de una Expresión a una Variable o a una Constante. La Expresión de Asignación (*da "área :base * :altura / 2*) asigna (*da*) el valor resultante de la Expresión Aritmética (*:base * :altura / 2*) a la variable *área*.

EJEMPLO

Para diseñar algoritmos que posteriormente puedan ser traducidos a un lenguaje de programación, es fundamental saber manejar muy bien los operadores y el orden en el que estos se ejecutan. Las fórmulas deben escribirse en una sola línea para que el computador las evalúe.

NOTACIÓN MATEMÁTICA	EXPRESIÓN
$\frac{\sqrt[3]{6^2 + 7}}{8^2}$	(rc (potencia 6 2)+ 7) / (potencia 8 2) Scratch no tiene el operador potencia, sin embargo es fácil programarlo: http://scratch.mit.edu/projects/jualop/752239
$\frac{Base * Altura}{2}$	(base * altura / 2) 

EJEMPLO

Evaluar la expresión *muestra ((potencia (5 + 3) 2) - 10) / 3 + 4 * (2 + 4)* teniendo en cuenta la jerarquía de los operadores:

R/.

$((\text{potencia } (5+3) \ 2) - 10) / 3 + 4 * (2 + 4) = ((\text{potencia } 8 \ 2) - 10) / 3 + 4 * 6$
 $((\text{potencia } 8 \ 2) - 10) / 3 + 4 * 6 = (64 - 10) / 3 + 4 * 6$
 $(64 - 10) / 3 + 4 * 6 = 54 / 3 + 4 * 6$
 $54 / 3 + 4 * 6 = 18 + 24$
 $18 + 24 = 42$
42

ACTIVIDAD

Tomando como modelo el ejemplo anterior y utilizando lápiz y papel,

desarrollar paso a paso las siguientes expresiones. Tener en cuenta la jerarquía de los operadores:

- $(5 + 2) * (4 + 4) = 56$
- $7 + 3 * 2 + (2 - 1) = 14$
- $6 * 2 + 4 * 3 + 5 / 2 = 26,5$
- $5 + 1 * 4 / 2 * 7 - (8 + 2) = 9$
- $8 + (5 * 6) - 6 = 32$
- $9 + 5 * 3 = 24$
- $4 / 2 * (10 - 5) * 3 = 30$

ACTIVIDAD

Pedir al estudiante que escriba en el Centro de Mando de MicroMundos (ilustración 1-2) las siguientes expresiones y anote en el cuaderno sus observaciones sobre el resultado:

- muestra 7 + 5 + 6*
- muestra [7 + 5 + 6]*
- muestra [mañana nos vemos en clase de inglés]*
- muestra mañana nos vemos en clase de inglés*

Obsérvese que las instrucciones 1 y 2 se diferencian en que mientras la primera da como resultado un valor numérico (18), la segunda es una lista y reporta una cadena alfanumérica de caracteres "7 + 5 + 6".

La instrucción 4 reporta "No sé cómo hacer mañana" porque le faltan los corchetes inicial y final para que MicroMundos la considere una lista de cinco palabras. En Scratch no son necesarios los paréntesis ya que el orden de evaluación de las expresiones es inequívoco.

EJEMPLO

Luisa Fernanda quiere llenar 5 cajas de bombones y sabe que en cada caja hay que incluir 12 bombones de menta, 14 de fresa intensa y 10 de limón. Encontrar al menos dos expresiones equivalentes para calcular el número de bombones que necesita Luisa Fernanda.

R/.

Expresión 1: $5 * 12 + 5 * 14 + 5 * 10$

¿Qué significa cada producto?

¿Qué significa la suma de los productos?

Expresión 2: $5 * (12 + 14 + 10)$

¿Qué representa la suma del paréntesis?

¿Por qué la suma se debe multiplicar por 5?

¿Las dos expresiones dan el mismo resultado?

Ejercicio adaptado de "Cuenta Jugando 5", Página 48 (Casasbuenas & Cifuentes, 1998b).

ACTIVIDAD

da "númeroA 5 (asigna el valor 5 a la Constante númeroA)

da "númeroB 8 (asigna el valor 8 a la Constante númeroB)

Utilizando la información de los valores asignados a las Constantes númeroA y númeroB, evaluar las siguientes expresiones:

- $20 + :númeroA$
- $:númeroA + 3 * :númeroB$
- $:númeroA > :númeroB$
- $:númeroA + "123$
- $4 + :númeroA - :númeroB$

UNIDAD 3: ESTRUCTURAS BÁSICAS

LAS ESTRUCTURAS

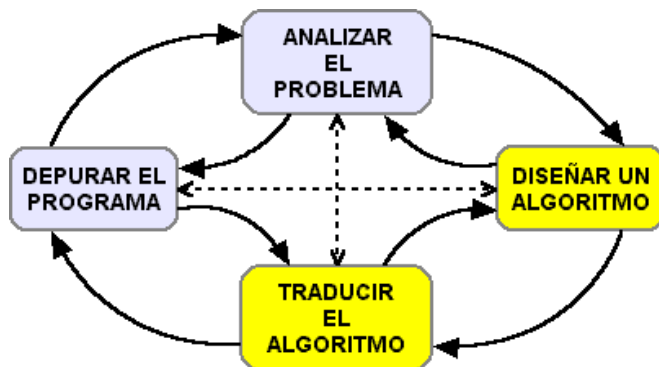


Ilustración 3-1: Fases segunda y tercera del ciclo de programación.

Un Algoritmo está compuesto por instrucciones de diferentes tipos, organizadas secuencialmente, en forma de estructuras de control. De estas estructuras, las más comunes y que se cubren en esta guía son las siguientes:

- Secuencial.
- Iterativa (repetición).
- Condicional (decisión, selección).

Una estructura de control se define como un esquema que permite representar ideas de manera simplificada y que bajo condiciones normales, es constante (Trejos, 1999).

El uso del diseño descendente en los programas, la ejecución de operaciones secuenciales, la utilización de ciclos repetitivos y, la toma de decisiones y alternativas de proceso, ofrecen amplias posibilidades para resolver problemas mediante la construcción de procedimientos (Castellanos & Ferreyra, 2000b).

Un famoso teorema de los años sesenta, formulado por Edsger Wybe Dijkstra, demostraba que se puede escribir cualquier programa utilizando únicamente la tres estructuras de control mencionadas. Actualmente, la programación orientada a objetos (POO) busca reducir al máximo la cantidad de estructuras de control mediante el uso de polimorfismo; pero aún así, todavía se vale de estas estructuras para construir métodos, los cuales podría decirse que son equivalentes a los procedimientos que se plantean en esta guía.

Para traducir los algoritmos diseñados a un lenguaje de programación que el computador pueda entender, en esta guía se utilizan dos entornos de programación basados en Logo: MicroMundos y Scratch. Los docentes interesados en conocer estos ambientes de

programación, pueden descargar gratuitamente las correspondientes Guías de Referencia:

- MicroMundos (proyecto Teddi - PDF; 560KB)
<http://www.eduteka.org/pdfdir/ManualMicroMundos.pdf>
- Scratch (MIT - PDF; 1.5MB)
<http://www.eduteka.org/pdfdir/ScratchGuiaReferencia.pdf>

TIP

Edsger Wybe Dijkstra nació en Rotterdam, (Holanda) en 1930. En 1956 anunció su algoritmo de caminos mínimos; posteriormente propuso el algoritmo del árbol generador minimal. A principios de la década de los 60, aplicó la idea de exclusión mutua a la comunicación entre un computador y su teclado. Su solución de exclusión mutua ha sido usada en muchos procesadores y tarjetas de memoria desde 1964, año en el que fue utilizada por IBM en la arquitectura del "IBM 360". El siguiente problema del que se ocupó Dijkstra fue el de los filósofos comensales. En este problema, cinco filósofos están sentados en una mesa circular con un plato de arroz delante y un palillo a cada lado, de manera que hay cinco palillos en total. El problema trata sobre el uso de recursos comunes sin que los procesos (los filósofos) lleguen a una situación de bloqueo mutuo; además, que los recursos se utilicen por todos los procesos de la manera más eficiente. Dijkstra también contribuyó a desterrar el comando GOTO de la programación: el comando "GOTO es considerado dañino. Cuantas más sentencias GOTO tenga un programa, más confuso será el código fuente".

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Logo

Logo es un lenguaje de programación con un número limitado de palabras y de reglas gramaticales si se lo compara con lenguajes humanos, como el castellano o el inglés. Las instrucciones en Logo son equivalentes a las oraciones en castellano y las reglas para construir esas instrucciones son más simples que las reglas gramaticales de nuestro idioma, pero mucho más precisas.

Logo fue desarrollado por Seymour Paper en el MIT en 1968 con el fin de utilizarlo en el ámbito educativo. Con este lenguaje de programación, inicialmente los niños dan instrucciones a una tortuga (adelante, atrás, derecha, izquierda, etc) para elaborar dibujos sencillos. Pero a medida que logran dominio del lenguaje, ellos utilizan comandos más sofisticados para realizar creaciones más complejas.

Desde 1968 a la fecha se han creado numerosas versiones de Logo (<http://www.elica.net/download/papers/LogoTreeProject.pdf>); sin embargo, en esta Guía se utilizarán los entornos de programación basados en Logo “MicroMundos” y “Scratch”. Ambos ofrecen una serie de comandos llamados “primitivas” para construir procedimientos (ver una lista básica de primitivas en el Anexo 1).

Para obtener mayor información sobre Logo, se recomienda visitar el sitio Web de Daniel Ajoy <http://neoparaiso.com/logo/>

MicroMundos

MicroMundos (<http://www.micromundos.com>) es un entorno de programación desarrollado por la compañía canadiense LCSi. La lista de primitivas es más extensa que la de Scratch; la cual, junto a los procedimientos elaborados por el programador, se pueden clasificar en dos categorías: Mandos y Reporteros.

Los **Mandos** hacen algo. Por ejemplo, *derecha* y *muestra* son Mandos (*derecha 120*, *muestra rumbo*, etc). Los **Reporteros** informan sobre el resultado de un cálculo o sobre el estado de un objeto. Por ejemplo, *rumbo* y *primero* son Reporteros. En la instrucción *muestra rumbo*, el Reportero *rumbo* devuelve al Mando *muestra* el rumbo actual de la tortuga, *rumbo* como tal no puede hacer nada.

Se requiere que los estudiantes aprendan a interpretar las instrucciones tal como lo hacen con las oraciones en castellano (sustantivos, verbos, conectores, adjetivos, etc). En MicroMundos la primera palabra de una instrucción siempre debe ser un Mando, por lo tanto, los Reporteros solo se pueden utilizar como entradas de un Mando o de un procedimiento.

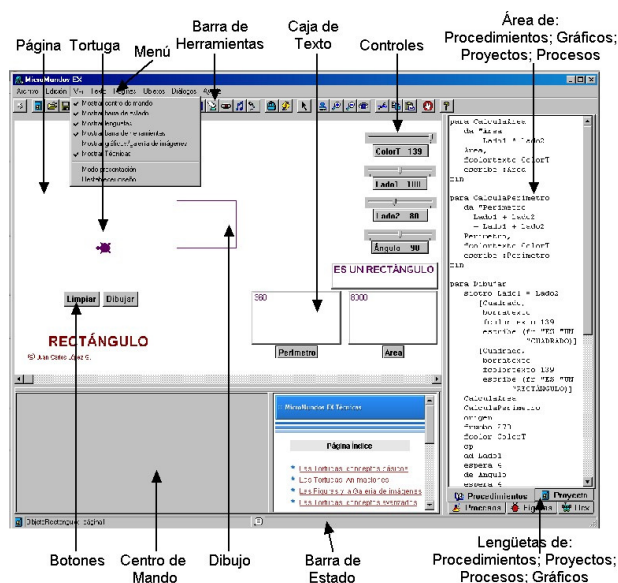


Ilustración 3-2(a): Área de trabajo de MicroMundos (interfaz del programa)

EJEMPLO 3-1

Digitar en el Centro de Mando (Ilustración 3-2) la instrucción: *rumbo*

MicroMundos devuelve un mensaje de error:

No sé qué hacer con 0

Digitar la siguiente instrucción en el Centro de Mando:

muestra rumbo

MicroMundos devuelve el rumbo actual de la tortuga:

0

En este caso el resultado de *rumbo* es reportado a *muestra*.

Los docentes interesados en conocer más a fondo MicroMundos, pueden descargar una versión de prueba por 30 días del sitio: <http://www.micromundos.com> Además, pueden descargar gratuitamente la Guía de Referencia en español: Proyecto Teddi; PDF; 560KB) <http://www.eduteka.org/pdfdir/ManualMicroMundos.pdf>

Aunque MicroMundos es bueno y ampliamente utilizado en escuelas de América Latina, su costo puede constituir una restricción para muchas Instituciones Educativas. Sin embargo, entre las alternativas gratuitas (licencia GNU) de Logo hay una que utiliza en núcleo de Logo creado por Brian Harvey de la Universidad de Berkeley: Microsoft Windows Logo (MSWLogo). Si bien, no es un ambiente de programación tan atractivo y elaborado como MicroMundos, esta versión de Logo es gratuita y tiene una traducción al español realizada por Javier López-Escobar que se puede descargar de: <http://sourceforge.net/projects/mswlogoes> Sin embargo, desde 2006, y debido al notorio abandono de MSWLogo por parte de su desarrollador, David Costanzo asumió la continuación del desarrollo de MSWLogo con el nombre FMSLogo: <http://fmslogo.sourceforge.net/>

Scratch

Scratch (<http://scratch.mit.edu/>) es un entorno de programación desarrollado recientemente por un grupo de investigadores del Lifelong Kindergarten Group del Laboratorio de Medios del MIT, bajo la dirección del Dr. Michael Resnick.

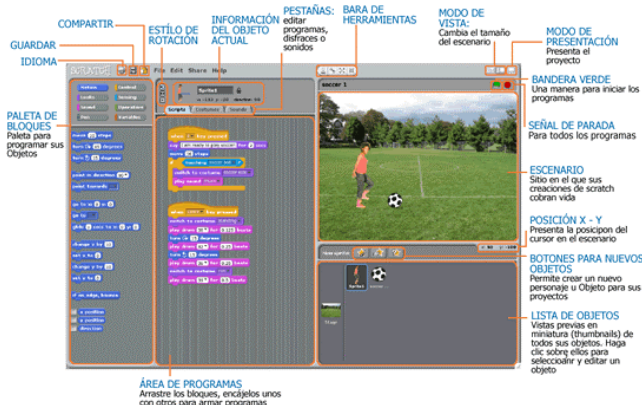


Ilustración 3-2(b): Área de trabajo de Scratch
(Ver imagen con mayor tamaño en el Anexo 4)

Aunque Scratch es un proyecto de código abierto, su desarrollo es cerrado. El código fuente se ofrece de manera libre y gratuita. Actualmente hay disponibles versiones oficiales para Windows, Mac y Sugar (XO); Además, hay versiones no oficiales para Linux.

Este entorno de programación aprovecha los avances en diseño de interfaces para hacer que la programación sea más atractiva y accesible para todo aquel que se enfrente por primera vez a aprender a programar. Según sus creadores, fue diseñado como medio de expresión para ayudar a niños y jóvenes a expresar sus ideas de forma creativa, al tiempo que desarrollan habilidades de pensamiento algorítmico y de aprendizaje del Siglo XXI, a medida que sus maestros superan modelos de educación tradicional en los que utilizan las TIC simplemente para reproducir prácticas educativas obsoletas.

Entre las características más atractivas de Scratch, adicionales a las mencionadas en el párrafo anterior, que lo hacen interesante para muchas Instituciones Educativas se cuentan: es gratuito, tiene el respaldo del MIT, está en permanente desarrollo (cada año se liberan aproximadamente dos versiones con cambios significativos), corre en computadores con bajas prestaciones técnicas y se puede ejecutar desde una memoria USB (pen drive), entre otras.

Scratch es una muy buena alternativa a MicroMundos; sin embargo, las Instituciones Educativas que ya cuentan con MicroMundos, pueden utilizar ambos entornos ya que puede resultar benéfico para los estudiantes exponerlos a diferentes ambientes de programación en los que puedan realizar las mismas cosas.

Los docentes interesados en conocer más a fondo el ambiente de programación Scratch, puede descargarlo gratuitamente de la siguiente dirección: <http://scratch.mit.edu/>. Además, pueden descargar la Guía de Referencia en español (PDF; 1.5MB): <http://www.eduteka.org/pdfdir/ScratchGuiaReferencia.pdf>

Fundamentos de programación

Una vez descritos brevemente los entornos de programación que se utilizan a lo largo de esta Guía, se tratarán a continuación una serie de conceptos básicos requeridos para empezar a utilizar dichos entornos.

COMENTARIOS

Los comentarios no tienen ningún efecto en la ejecución del algoritmo. Se utilizan para aclarar instrucciones que puedan prestarse a confusión o como ayuda a otras personas que deben leerlo y entenderlo. La mayoría de los lenguajes de programación ofrecen la posibilidad de comentar el código de los programas.

Los comentarios en un procedimiento de MicroMundos se hacen con el punto y coma (;). MicroMundos ignora el texto entre el punto y coma (;) y el final de la línea.

MicroMundos

para lectura

```
local "valorUno" ; declarar variables y constantes
local "valorDos"
pregunta [Ingrese el primer valor] ; ingresar valorUno
da "valorUno respuesta"
pregunta [Ingrese el segundo valor] ; ingresar valorDos
da "valorDos respuesta"
muestra frase [El primer valor es ] :valorUno
muestra frase [El segundo valor es ] :valorDos
fin
```

En Scratch, los comentarios se agregan en una caja de texto amarilla que se crea al hacer clic derecho sobre cualquier parte del área de programas (zona central gris) y seleccionar la opción "añadir comentario".

Scratch



Los algoritmos diseñados como ejemplos en esta Guía no están optimizados ya que con ellos se busca mostrar explícita y detalladamente las operaciones básicas requeridas para una solución y no la forma óptima de solución. Algunos ejemplos exponen de manera

deliberada la forma larga y poco elaborada, para luego presentar la forma optimizada.

EJEMPLO 3-2

Escribir un procedimiento que se llame triángulo para hallar el área de un triángulo rectángulo cuya Base mide 3 cm, la Altura 4 cm y la Hipotenusa 5 cm. Introducir en el código comentarios que aclaren lo que está sucediendo en cada uno de los pasos importantes.

R/

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya se encuentra claramente planteado.

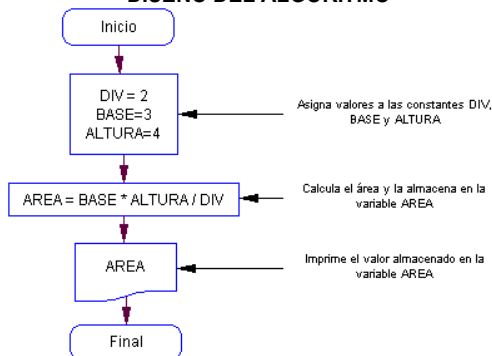
Resultados esperados: El área de un triángulo rectángulo.

Datos disponibles: Base, Altura, Hipotenusa, tipo de triángulo. La incógnita es el área y todos los valores son constantes. El valor de la hipotenusa se puede omitir. El estudiante debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

Restricciones: Utilizar las medidas dadas.

Procesos necesarios: Guardar en dos variables (BASE y ALTURA) los valores de Base y Altura; Guardar en una constante (DIV) el divisor 2; aplicar la fórmula $BASE * ALTURA / DIV$ y guardar el resultado en la variable AREA; comunicar el resultado (AREA).

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO

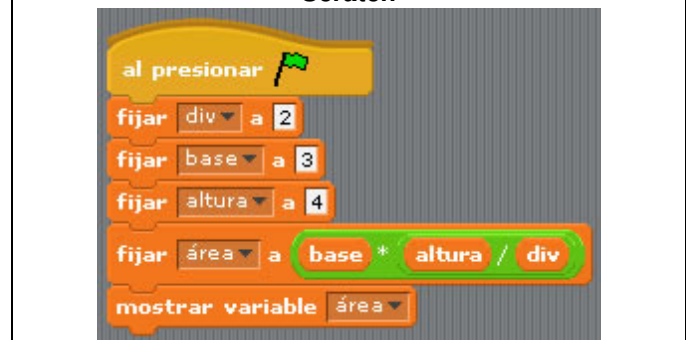
MicroMundos

```

para triángulo
  local "div"      ;declara las constantes como locales.
  local "base"
  local "altura"
  local "área"     ;declara esta variable como local.
  da "div 2"       ;almacena 2 en la constante div (div=2)
  da "base 3"      ;equivalente a base=3
  da "altura 4"
  da "área :base * :altura / :div" ;aplica la fórmula.
  muestra :área     ;reporta el contenido de la variable área
fin
  
```

Tal como se puede comprobar al ejecutar el procedimiento anterior, el texto entre un punto y coma y el final de una línea de código no produce ningún resultado, ni causa errores de sintaxis.

Scratch



PROCESOS

Se llama procesos a todas las instrucciones contenidas en un algoritmo para:

- declarar variables y constantes (solo en MicroMundos);
- asignar valores iniciales a variables y constantes;
- leer datos que suministra el usuario por medio del teclado o del ratón (mouse);
- realizar operaciones matemáticas (aplicar fórmulas);
- reportar o mostrar contenidos de variables y constantes;
- mostrar en pantalla resultados de procedimientos activados por el programa.

Vale la pena recordar que la declaración de variables y constantes se realiza en MicroMundos de la siguiente forma:

local "nombreVariable" (Ej. local "altura").

La sintaxis para asignación de un valor a una variable o constante es:

da "nombreVariable Valor" (Ej. da "altura 4").

Luego de asociar valores a variables o constantes, estas se pueden utilizar anteponiendo dos puntos (:) al nombre de la variable o constante:

da "nuevaAltura 2 + :altura"

El Mando *da* asigna a la variable *nuevaAltura* el valor 2 más el contenido de la variable *altura*.

Por otra parte, al momento de crear una variable en Scratch, el entorno pregunta si esta será visible "para todos los objetos" (global) o solo será visible "para este objeto" (local).

Scratch



INTERACTIVIDAD

La interactividad entre el usuario y el programa es un recurso muy valioso en programación y se logra permitiendo la comunicación con el programa mediante la utilización del teclado y/o el ratón (mouse).

En MicroMundos, la forma más común de interactuar con un programa es por medio de controles y botones, los cuales se manipulan con el ratón. Ambos se pueden programar para que realicen acciones (ejecutar un procedimiento, ir a otra página, cambiar el valor a una variable, etc).

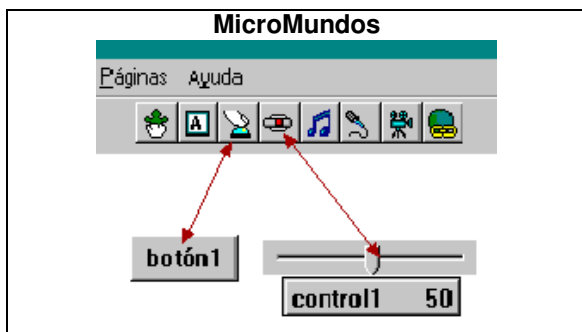


Ilustración 3-3: Con el ratón se puede hacer clic sobre los Botones y deslizar el indicador de los Controles (a la izquierda disminuye y hacia la derecha aumenta).

En Scratch algunos objetos se les puede dar forma de botón y programarlos para que cumplan la misma función que cumplen estos en MicroMundos. Respecto a los controles, en Scratch cada variable se puede convertir en un control deslizante. Basta con hacer clic derecho sobre cualquier variable que se muestre en el escenario y seleccionar “deslizador”; luego se hace clic derecho nuevamente y se eligen los valores mínimo y máximo que puede almacenar esa variable mediante la manipulación con el deslizador.



Otra forma de interactuar con un programa consiste en que el usuario utilice el teclado para responder a las preguntas que le plantea el programa. En MicroMundos las respuestas que el usuario aporta se pueden almacenar en las variables correspondientes con el comando *respuesta*. Para mostrar lo que el usuario contesta se puede utilizar varios comandos de MicroMundos (*mostrar*, *escribe*, *anuncia*, *anuncia frase*).

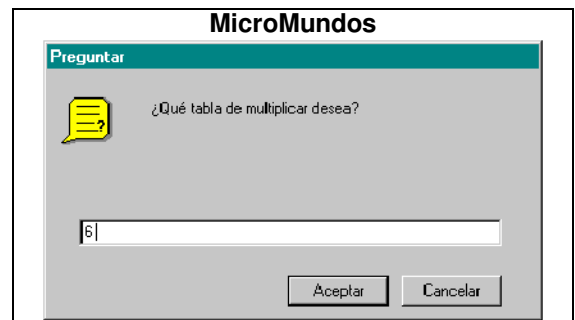


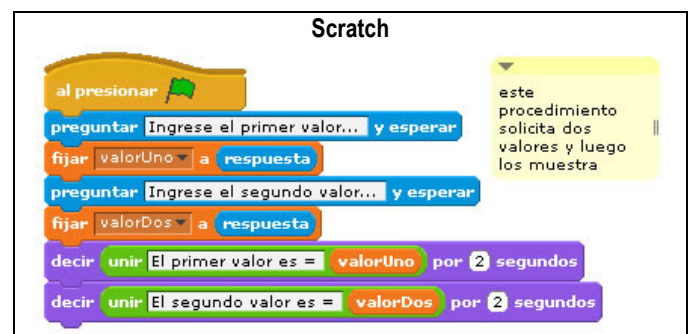
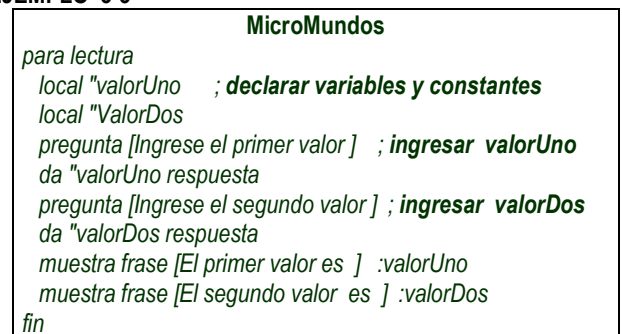
Ilustración 3-4: Los comandos pregunta y respuesta permiten interactuar con el usuario para obtener información de él.



Ilustración 3-5: Los comandos pregunta y respuesta permiten interactuar con el usuario para obtener información de él.

En Scratch las respuestas que el usuario aporta se pueden almacenar en variables con el comando respuesta. Para mostrar lo que el usuario contesta se puede utilizar los comandos “decir” y “pensar”.

EJEMPLO 3-3



El ejemplo 3-6 de la sección “Estructura Secuencial” ilustra muy bien la interactividad que se puede establecer entre usuario y programa. La interactividad

facilita generalizar la solución de un problema. Por ejemplo, si se elabora un procedimiento que calcule el área de un triángulo rectángulo, cada vez que se cambien los valores iniciales de “base” y “altura”, estos deben actualizarse en el cuerpo del procedimiento. La interactividad, por el contrario, permite que cada vez que se ejecute el procedimiento, este le pregunte al usuario por los valores de “base” y “altura”, y los tome para calcular el área sin la necesidad de modificar nada en el procedimiento.

PROCEDIMIENTOS

Según Papert (1993), los problemas que experimentan muchos estudiantes con las matemáticas se deben más a la falta de comprensión de los algoritmos apropiados, que a la falta de manejo de los conceptos involucrados en estos. Cuando experimentan problemas con la suma, lo primero que se debe revisar es el procedimiento de la adición.

Los procedimientos se utilizan muy a menudo en la vida diaria. Participar en un juego o dar instrucciones a alguien que se encuentra perdido son actividades que requieren pensamiento procedimental o algorítmico. Los estudiantes activan a diario este tipo de pensamiento sin percatarse y sin hacer algún tipo de reflexión sobre esto. De hecho, ellos disponen de un conocimiento procedimental que utilizan en muchos aspectos de sus vidas, tanto para planear una estrategia en un juego, como para dar instrucciones a alguien perdido en el vecindario. Lo curioso es que en raras ocasiones utilizan este conocimiento procedimental en las clases de matemáticas. Los ambientes Logo ayudan a los estudiantes a hacer conciencia de la idea de procedimiento, ya que en él, los procedimientos se convierten en algo que se puede nombrar, manipular y reconocer (Papert, 1993).

Además del pensamiento algorítmico, los razonamientos temporal y condicional juegan un papel muy importante en la gestión, organización y planificación de cualquier procedimiento. Estos facilitan al programador plantear y seguir instrucciones, fragmentar una tarea en módulos con funciones precisas y plantear decisiones que un procedimiento debe tomar de acuerdo a ciertas condiciones. La programación está muy ligada al control de la sucesión temporal de instrucciones organizadas en secuencias. Este control, guiado por un razonamiento temporal, es necesario para organizar el orden de las instrucciones a ejecutar, para llamar otros procedimientos en el orden correcto y ejecutar ciertas instrucciones mientras una condición se dé o a partir de cuando esta se dé (Dufoyer, 1991). Y, según Friedman (1982) y Piaget (1946), citados por Dufoyer (1991), hay que esperar necesariamente hasta los 7 u 8 años (nivel de las operaciones concretas) para que estas operaciones sean posibles. Por lo tanto, utilizar la programación de computadores con estudiantes de cuarto grado en adelante es viable si se hace con el objetivo de ofrecerles oportunidades de acceso a conceptos relacionados con procedimientos, creando

condiciones para que ellos ejerciten en forma efectiva y divertida el pensamiento algorítmico y los razonamientos temporal y condicional.

En este sentido, la utilización del área de procedimientos de MicroMundos (ilustración 3-2) ofrece mayores posibilidades para gestionar, organizar y planificar, en contraposición a introducir instrucciones una a una en el “centro de mando” (lo que permite al estudiante ver inmediatamente cuál es el efecto que produce cada instrucción ejecutada).

Como se expuso en la Unidad 1, los procedimientos son módulos con instrucciones que inician con el comando “para” y que el computador ejecuta automáticamente, una tras otra, hasta encontrar el comando “fin”. Todo procedimiento debe tener un nombre que lo identifique y que sirve para ejecutarlo cuando se ejecuta dicho nombre. Pero antes de ejecutar un procedimiento, los estudiantes deben utilizar pensamiento algorítmico, razonamiento temporal y razonamiento condicional, para determinar y escribir todas las instrucciones que lo deben componer y el orden lógico de ejecución.

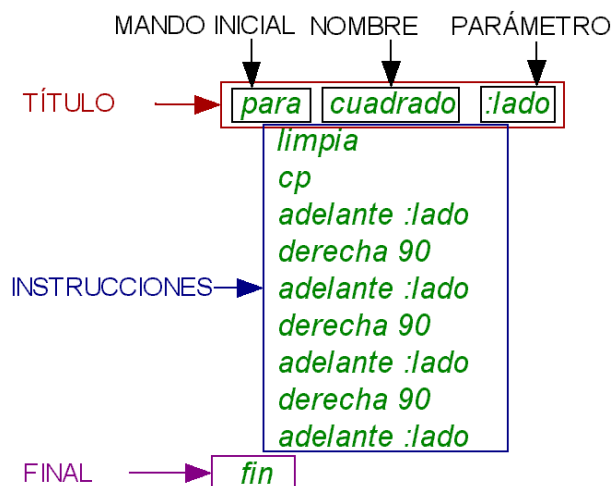


Ilustración 3-5: Elementos que componen el procedimiento “cuadrado”; el parámetro es el único elemento opcional.

Por otra parte, el vocabulario disponible en cualquier proyecto Logo está compuesto por los comandos propios del lenguaje (Mandos y Reporteros) más los nombres de los procedimientos definidos en ese proyecto. En otras palabras, los procedimientos definidos (en este caso “cuadrado”) entran a formar parte del vocabulario de ese proyecto. Cuando se abre un proyecto nuevo, esos procedimientos ya no están disponibles; por tanto, para utilizarlos hay que copiarlos de un proyecto existente y pegarlos en el nuevo proyecto.

Todo procedimiento debe tener una línea de título que incluye un nombre; el cual, al invocarlo, ejecuta en orden, una a una, las líneas de instrucciones que contiene hasta que llega a la línea con el Mando *fin*.

Para escribir procedimientos se deben tener en cuenta las siguientes recomendaciones:

- El título está compuesto por: el Mando “*para*”, el nombre del procedimiento y los parámetros (opcional).
- El nombre del procedimiento debe ser una palabra. Cuando se deben utilizar varias palabras para nombrar un procedimiento, se escriben sin dejar espacio entre ellas y con mayúscula la primera letra de la segunda palabra y de las subsiguientes palabras (ejemplo: *valorMasAlto*).
- Cada instrucción completa debe ocupar una línea de código
- Puede contener líneas en blanco
- En la última línea solo puede aparecer el Mando “*fin*”
- Desde un procedimiento se pueden invocar otros procedimientos.

Ejemplo 3-4

Escribir un procedimiento para dibujar en la pantalla un cuadrado de tamaño variable.

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya está claramente planteado.

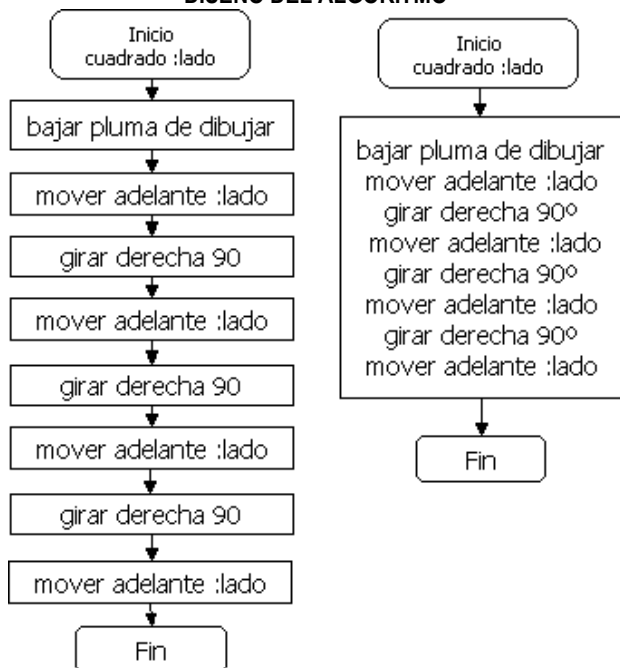
Resultados esperados: El dibujo de un cuadrado en la pantalla.

Datos disponibles: El tamaño de los lados del cuadrado debe ingresarlo el usuario; se sabe que todos los ángulos internos de un cuadrado son de 90 grados. El estudiante debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

Restricciones: El tamaño del cuadrado lo suministra el usuario.

Procesos necesarios: Leer el tamaño del cuadrado como un parámetro llamado *lado*; bajar la pluma de dibujar; avanzar adelante una distancia igual a *lado*; girar 90 grados a la derecha; avanzar adelante una distancia igual a *lado*; girar 90 grados a la derecha; avanzar adelante una distancia igual a *lado*; girar 90 grados a la derecha; avanzar adelante una distancia igual a *lado*.

DISEÑO DEL ALGORITMO



Aquí tenemos dos formas equivalentes e igualmente correctas de algoritmo para dibujar un cuadrado. El algoritmo de la izquierda utiliza un rectángulo de proceso para cada instrucción; el de la derecha agrupa en un solo rectángulo las instrucciones de proceso adyacentes.

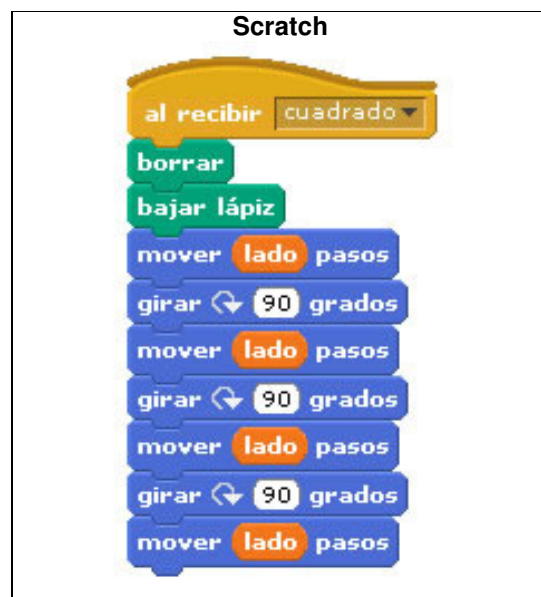
TRADUCCIÓN DEL ALGORITMO

El siguiente procedimiento es la traducción al lenguaje Logo del algoritmo representado por el diagrama de flujo. Pedir a los estudiantes que lo escriban en el área de procedimientos.

```

MicroMundos
para cuadrado :lado
  limpia
  cp
  adelante :lado
  derecha 90
  adelante :lado
  derecha 90
  adelante :lado
  derecha 90
  adelante :lado
  fin
  
```

Cuando escriban en el Centro de Mando el nombre del procedimiento, seguido del parámetro correspondiente al tamaño del lado (Ejemplo: *cuadrado 100*) debe dibujarse en la pantalla un cuadrado.



Con el fin de poder ejemplificar más adelante las dos formas de ejecutar un procedimiento, se debe escribir este otro procedimiento:

```

MicroMundos
para dibujaCuadrado
  cuadrado 120
  fin
  
```

El procedimiento *cuadrado* ilustra el modelo general para escribir procedimientos con parámetros. Este tipo de procedimientos son muy útiles cuando varias

instrucciones se deben ejecutar en un programa varias veces de manera idéntica, cuya única diferencia sea el valor inicial que tomen algunas variables. En estos casos, las instrucciones se “empaquetan” en un procedimiento y los distintos valores iniciales de las variables se asignan por medio de parámetros.

El nombre del parámetro se debe escribir en la línea del título, después del nombre del procedimiento, ajustándose a las indicaciones establecidas en la Unidad 2 de esta guía para nombrar identificadores. No olvidar que al parámetro se debe anteponer el signo de dos puntos (:) tal como se puede apreciar en el ejemplo 3-4 (:lado).

Luego de definir el nombre de un parámetro en la línea de título de un procedimiento (*para cuadrado :lado*), este parámetro se puede utilizar en cualquiera de las instrucciones (ejemplo: *adelante :lado*).

En otras palabras, la línea de título del procedimiento *cuadrado* le dice a Logo que solo hay un parámetro, llamado lado. En el cuerpo del procedimiento se utiliza el Mando *adelante* acompañado del parámetro *:lado*. Esta instrucción dibuja una línea cuya medida es igual al valor que contiene *:lado*.

Cuando en el título de un procedimiento se definen uno o varios parámetros, estos no tienen valor. Solo cuando se ejecuta el procedimiento es que se conoce el valor de cada parámetro. Por esta razón, todo parámetro debe tener un nombre. Por ejemplo, al cambiar la línea de título del procedimiento *cuadrado* por la siguiente (con dos parámetros):

para cuadrado :lado :ángulo

el valor del ángulo se tratará como un parámetro cuando se invoque el procedimiento.

En resumen, un procedimiento, internamente, es un algoritmo que resuelve en forma parcial un problema. Desde el punto de vista externo es una orden o instrucción única que puede formar parte de otro procedimiento o algoritmo (Cajaraville, 1989).

Por otra parte, los estudiantes deben aprender a distinguir entre escribir un procedimiento y ejecutarlo. La primera actividad es el resultado de las etapas de análisis del problema, diseño del algoritmo y traducción de este a un lenguaje de programación.

La segunda actividad (ejecutar un procedimiento) hay dos formas de realizarla. La manera más simple consiste en escribir el nombre del procedimiento en el Centro de Mando (ver ilustración 3-2). Escribir *cuadrado 100* en el Centro de Mando ejecuta el procedimiento llamado *cuadrado*, asigna el valor 100 al parámetro *:lado* y dibuja en la pantalla un cuadrado. Este procedimiento requiere que el usuario especifique la medida de los lados. Para dibujar un cuadrado cuyos lados midan 60 unidades se debe escribir *cuadrado 60* en el Centro de Mando. Para dibujar un cuadrado más grande se debe aumentar el valor del parámetro; por ejemplo:

cuadrado 150.

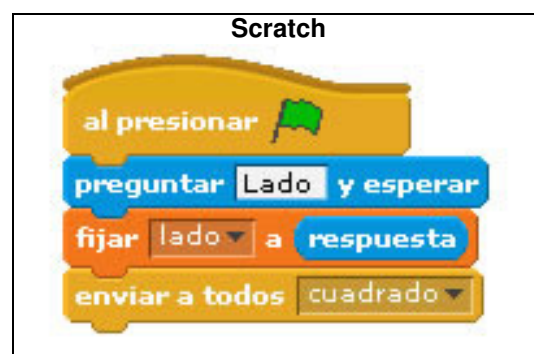
Otra forma de ejecutar un procedimiento es invocándolo desde otro procedimiento. Cuando se ejecuta el procedimiento *dibujaCuadrado* (sin parámetros), indirectamente se ejecuta el procedimiento *cuadrado* con 120 como parámetro. Este procedimiento dibuja exactamente el mismo cuadrado cada vez que se ejecuta (ver ejemplo 3-4).

Adicionalmente, poder invocar un procedimiento desde otro, simplifica el dibujo de figuras simétricas. Se dice que una figura es simétrica cuando al doblarla por la mitad sus dos partes coinciden (Casasbuenas & Cifuentes, 1998b). El ejemplo 3-5 en la sección “Estructura Secuencial” ilustra muy bien la construcción de figuras simétricas mediante la utilización de procedimientos que invocan a otros procedimientos.

Cada procedimiento tiene su propia biblioteca de nombres, esta es la razón por la cual un procedimiento puede transferirle parámetros a otros, manteniendo los valores de sus propios parámetros. Como los nombres de los parámetros son privados, diferentes procedimientos pueden usar el mismo nombre para un parámetro sin que haya confusión acerca de su valor. Cuando Logo ejecuta un procedimiento, éste establece una biblioteca privada que contiene los nombres de los parámetros que hay en la definición del procedimiento, asociados con los valores dados al momento de llamar al procedimiento. Al ejecutarse una instrucción que contiene el nombre del parámetro, el procedimiento busca en su biblioteca privada el valor de ese nombre. Este es el motivo por el cual el mismo nombre de parámetro puede estar en dos bibliotecas distintas y tener información diferente.

Los parámetros son como “variables locales” del procedimiento en el cual fueron definidos. La importancia de los nombres de entrada privados es que ofrecen la posibilidad de ejecutar un procedimiento sin tener que preocuparse por detalles de su definición.

Por su parte, en Scratch no hay procedimientos con parámetros como en MicroMundos; sin embargo, se pueden simular con las instrucciones de Control “Al enviar a todos” y “Al recibir”.





Para ejecutar los procedimientos en Scratch basta con hacer clic en la bandera verde ubicada en la esquina superior derecha de la pantalla:



En la siguiente sección se presentan las estructuras de control básicas que los estudiantes deben dominar para poder diseñar algoritmos y traducirlos a un lenguaje de programación.

ESTRUCTURA SECUENCIAL

Una estructura se define como un esquema con cierta distribución y orden que permite representar una idea de forma simplificada y que bajo ciertas condiciones es constante (Trejos, 1999). La estructura de control secuencial es la más sencilla. También se la conoce como estructura lineal. Se compone de instrucciones que deben ejecutarse en forma consecutiva, una tras otra, siguiendo una línea de flujo. Solamente los problemas muy sencillos pueden resolverse haciendo uso únicamente de esta estructura. Normalmente, la estructura secuencial hace parte de soluciones a problemas complejos en las que se la utiliza mezclada con estructuras iterativas (repetir varias veces un conjunto de instrucciones) y condicionales (tomar decisiones).



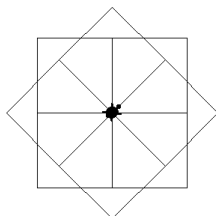
Ilustración 3-6: Modelo de estructura secuencial.

Una estructura de control secuencial puede contener cualquiera de las siguientes instrucciones:

- declaración variables
- asignación de valores
- entrada de datos
- procesamiento de datos (operaciones)
- reporte de resultados

EJEMPLO 3-5

Escribir un procedimiento para dibujar en la pantalla una figura simétrica igual a la siguiente:



R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya está claramente planteado.

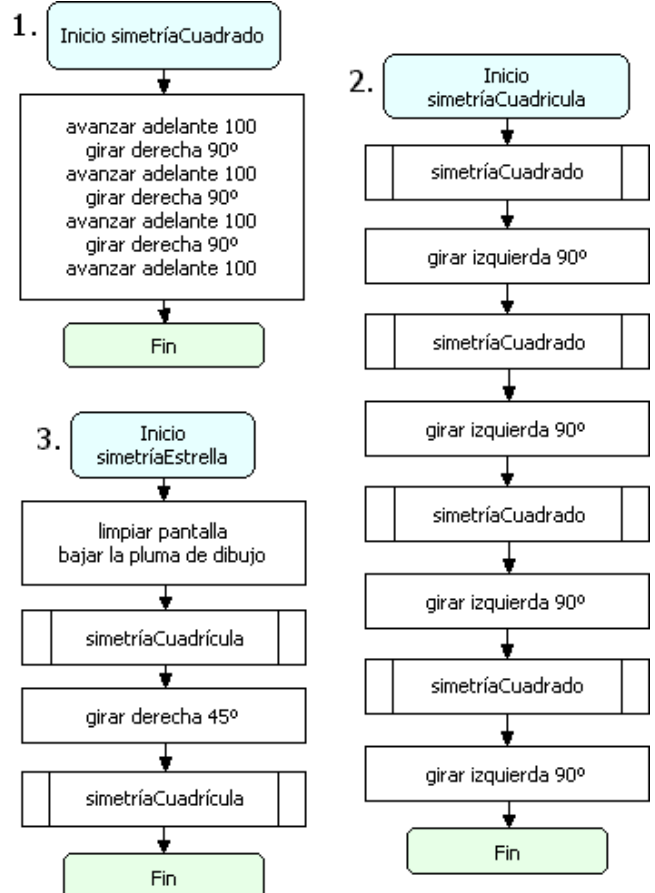
Resultados esperados: El dibujo dado.

Datos disponibles: La figura geométrica suministrada. El análisis de la figura permite establecer que está construida con varios cuadrados de igual tamaño.

Restricciones: La figura resultante debe ser igual en su forma a la muestra. Las dimensiones pueden variar.

Procesos necesarios: la figura se puede realizar mediante tres procedimientos. El primero, que puede llamarse *simetríaCuadrado*, dibuja un cuadrado con los comandos *adelante* y *derecha*. El segundo procedimiento (*simetríaCuadrícula*), dibuja cuatro cuadrados utilizando el procedimiento *simetríaCuadrado* y el comando *izquierda*. El tercero (*simetríaEstrella*), dibuja dos veces la figura que se forma con el procedimiento *simetríaCuadrado*; se debe girar la tortuga 45 grados a la derecha luego de ejecutar *simetríaCuadrado* por primera vez.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO

La solución de este problema tiene tres procedimientos:

(1) *simetríaCuadrado*, (2) *simetríaCuadrícula* y (3) *simetríaEstrella*.

para *simetríaCuadrado*

adelante 100

derecha 90

adelante 100

derecha 90

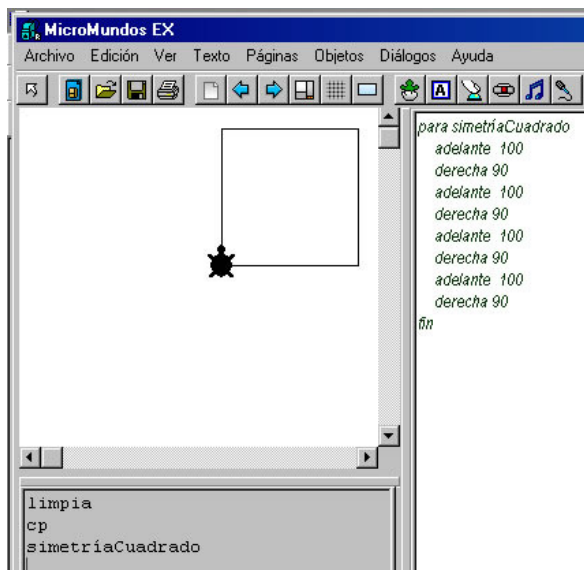
adelante 100

derecha 90

adelante 100

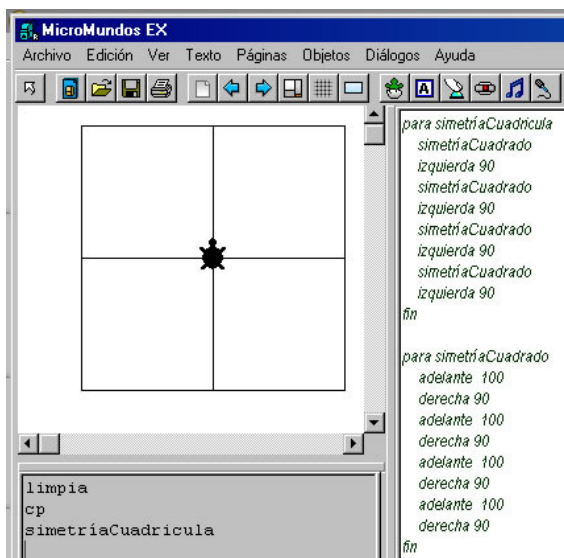
derecha 90

fin



El procedimiento *simetríaCuadrado* (1) está compuesto solamente por una estructura secuencial que realiza el dibujo de un cuadrado cuyos lados miden 100.

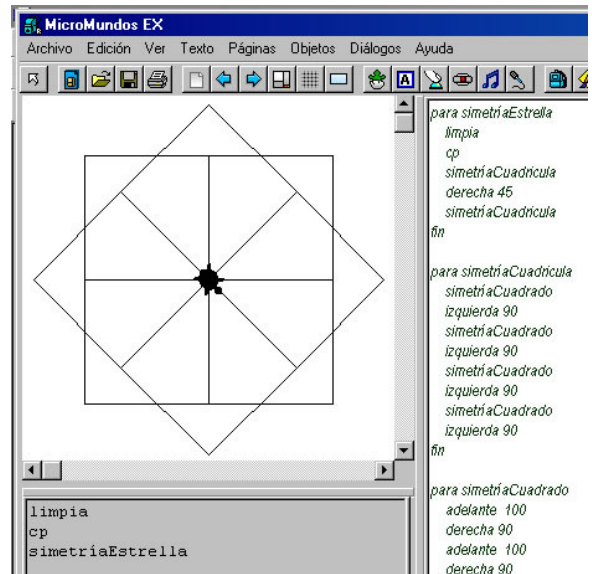
```
para simetríaCuadrícula
  simetríaCuadrado
  izquierda 90
  simetríaCuadrado
  izquierda 90
  simetríaCuadrado
  izquierda 90
  simetríaCuadrado
  izquierda 90
fin
```



El procedimiento *simetríaCuadrícula* (2) está compuesto por una estructura secuencial que dibuja cuatro cuadrados con lados adyacentes entre sí. Esta es una figura simétrica ya que si se dobla por la mitad, ambas mitades coinciden. Para construir esta cuadrícula no fue necesario escribir un procedimiento con el código para dibujar cuadro cuadrados; en realidad, el procedimiento

simetríaCuadrícula no realiza ningún dibujo, lo que hace es llamar cuatro veces el procedimiento *simetríaCuadrado* que es el que realmente dibuja un cuadrado cada vez que se invoca. Para que los cuadrados sean adyacentes, se necesita girar la tortuga 90 grados a la izquierda después de cada llamada al procedimiento *simetríaCuadrado*.

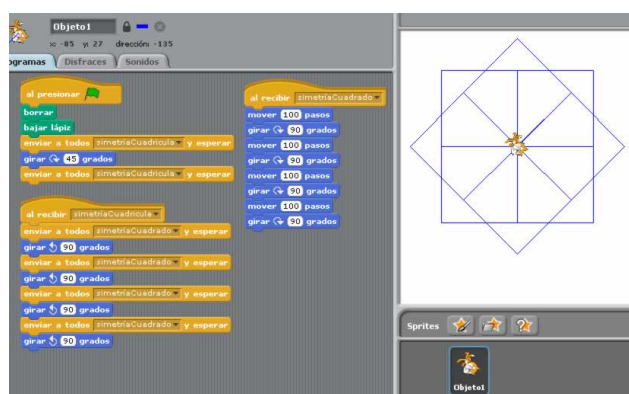
```
para simetríaEstrella
  limpia
  cp
  simetríaCuadrícula
  derecha 45
  simetríaCuadrícula
fin
```



El procedimiento *simetríaEstrella* (3) también está compuesto únicamente por una estructura secuencial que permite dibujar una estrella perfectamente simétrica. Este procedimiento llama dos veces al procedimiento *simetríaCuadrícula* que a su vez, cada que es invocado, llama cuatro veces al procedimiento *simetríaCuadrado*. Para lograr el efecto de estrella, luego de llamar la primera vez al procedimiento *simetríaCuadrícula*, se gira la tortuga 45 grados, antes de llamar el mismo procedimiento por segunda vez.

Las siguientes son las instrucciones equivalentes en Scratch para elaborar la misma estrella:





<http://scratch.mit.edu/projects/jualop/42800>

Muchos estudiantes logran construir la figura del procedimiento *simetríaEstrella* utilizando gran cantidad de comandos que se repiten sin estructura alguna (mediante experimentación). Es muy importante que ellos reflexionen sobre las ventajas que ofrecen los procedimientos cuando se los utiliza a manera de objetos que cumplen con una función determinada (dibujar un cuadrado, calcular un área, etc). Una tarea que debe realizarse varias veces es candidata ideal para tratarla como un procedimiento. Con la utilización de parámetros se pueden cambiar algunos valores cada vez que se ejecute esa tarea. De esta manera, si necesitamos dibujar varios cuadrados de diferentes tamaños, lo más adecuado será construir un procedimiento con el valor de Lado como parámetro y ejecutarlo varias veces asignando a este el valor del Lado del cuadro a dibujar, cada vez que se ejecute (ver el ejemplo 3-4).

Este ejemplo ilustra la construcción de figuras simétricas mediante la utilización de procedimientos invocados desde otros procedimientos. El concepto de simetría es muy importante tanto en disciplinas como matemáticas y arte como en la cultura general del estudiante. La programación de computadores puede apoyar muy efectivamente el afianzamiento en el niño del concepto que este tiene de simetría, alcanzado en forma intuitiva a través del espejo. “Los espejos son para los niños su primera experiencia y permiten examinar muchos aspectos de las simetrías. Se puede uno preguntar acerca de la inversión mutua entre derecha e izquierda, acerca de las distancias entre el objeto y su imagen en el espejo, lo que ocurre cuando se mueve el objeto o se mueve el espejo, o lo que ocurre cuando estos giran” Fletcher, T. J. citado por Cajaraville (1989).

Por otra parte, la construcción de figuras como estas requiere un dominio espacial del estudiantes ya que es mucho más difícil reproducir una acción correctamente en el pensamiento que llevarla a cabo en el nivel de la conducta. Por ejemplo, es más sencillo moverse de un lugar a otro en un espacio físico o dar vueltas en torno a objetos que representar mentalmente esos movimientos con precisión o representarlos en un plano e invertir mentalmente las posiciones de los objetos haciendo girar el plano (Piaget, 1993).

Por último, en este ejemplo se utilizan más comandos de los que realmente se requieren. Como el objetivo no es presentar códigos optimizados sino más bien que el estudiante se familiarice con los comandos disponibles, más adelante, cuando se vea la estructura repetitiva, el docente puede repetir este ejemplo y utilizar el comando “repetir”. Posteriormente, promover la reflexión de los estudiantes sobre la optimización del código y las múltiples maneras que hay en programación para realizar la misma tarea.

EJEMPLO 3-6

Escribir un procedimiento para calcular el área de cualquier triángulo rectángulo. En él se debe pedir al usuario que ingrese los valores de la Altura y la Base del triángulo.

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya está claramente planteado.

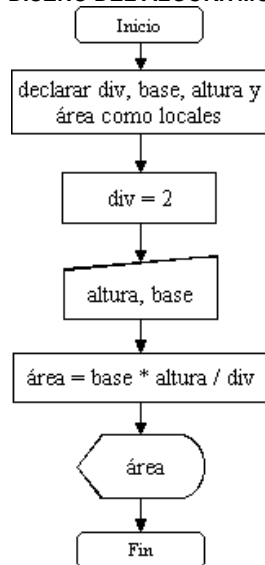
Resultados esperados: Un procedimiento que permita calcular el área de cualquier triángulo rectángulo.

Datos disponibles: Base y Altura del triángulo (se deben solicitar al usuario). El estudiante debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

Restricciones: Los valores de base y altura son variables y se deben solicitar al usuario.

Procesos necesarios: definir variables; asignar el valor 2 a la constante div; solicitar al usuario el valor de la altura del triángulo; solicitar al usuario el valor de la base; aplicar la fórmula de área; mostrar el resultado.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

para triánguloRectángulo

```

local "div" ; declarar variables y constantes
local "base"
local "altura"
local "área"
da "div" 2
pregunta [Ingrese la Altura del Triángulo] ; ingresar altura y
base
da "altura respuesta"
pregunta [Ingrese la Base del Triángulo]
da "base respuesta"
da "área :base * :altura / :div" ; realizar cálculos
anuncia frase [El Área del triángulo es:] :área ; reportar el
resultado
fin
  
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



En este ejemplo, el procedimiento *triánguloRectángulo* también está compuesto únicamente por una estructura secuencial de instrucciones. En ella se utilizan las primitivas “pregunta” y “respuesta” para permitir que el usuario del programa suministre al programa los datos “altura” y “base” del triángulo. De esta forma, se logra un procedimiento generalizado para calcular el área de CUALQUIER triángulo rectángulo. En otras palabras,

cada vez que se ejecute el procedimiento *triánguloRectángulo*, este le preguntará al usuario cuál es la altura y la base del triángulo del cual desea calcular su área.

Tanto en la utilización de la estructura secuencial, como en las dos que veremos más adelante, es muy importante que los estudiantes reflexionen y determinen el orden de ejecución de las instrucciones (posición) ya que la conmutatividad no es una propiedad aplicable a los algoritmos. El lenguaje algorítmico, al igual que el lenguaje formal de las matemáticas, tiene carácter gráfico y posicional; busca la precisión, el rigor, la abreviación y la universalidad; y, su finalidad fundamental consiste en obtener resultados internamente consistentes (Onrubia & Rochera & Barbarà, 2001).

La construcción de estructuras algorítmicas, entendidas estas como secuencias de instrucciones y operaciones, con el fin de lograr un resultado concreto, ayuda a afianzar en los estudiantes el conocimiento procedimental matemático. Este conocimiento se caracteriza por la acción (saber hacer) frente al conocimiento declarativo que se basa en la enunciación (saber decir). Saber explicar (enunciar) un teorema no garantiza que este se sepa aplicar (actuar) correctamente en la solución de una situación problemática determinada (Onrubia & Rochera & Barbarà, 2001).

Toda secuencia de acciones tiene una estructura que debe planearse (consciente o inconscientemente) antes de ejecutarla. Cuando la acción se realiza de manera automática, quien actúa no es consciente de la estructura y por tanto no puede ver las correlaciones en su actuación y con el entorno de dicha acción; las acciones se convierten en operaciones cuando quien las realiza es consciente de las relaciones inherentes. Pero las acciones prácticas requieren tanta atención que puede ser difícil realizarlas dándose cuenta al mismo tiempo de las correlaciones inherentes a ellas. Por esto, son fundamentales los sistemas de signos a los cuales se traducen las acciones; con los signos se pueden expresar las relaciones que existen dentro de las acciones y entre sus objetos, y se puede proceder con los signos del mismo modo que con los objetos reales (Aebli, 2001).

Según Saussure (1916), citado por Aebli (2001), hay tres grandes grupos dentro de los signos: los símbolos, los signos propiamente dichos y las señales. Un signo, a diferencia de un símbolo, no se parece a su significado; es elegido arbitrariamente y para conocer su significado hay que aprenderlo y fijarlo en la memoria: palabras de lenguajes naturales, cifras, signos algebraicos, etc. Los significados se pueden codificar básicamente de cuatro formas: mediante la palabra hablada, la palabra escrita, el signo gráfico, y la variable. Así, el número 2 se puede representar mediante el fonema “dos”, la palabra “dos”, el signo “2” o “.” y la variable “a”.

De lo anterior se puede deducir que elaborar programas de computador para resolver problemas matemáticos ofrece al estudiante la oportunidad de fijar la atención en la estructura de las operaciones que realiza, mediante su traducción a un sistema de signos que el computador pueda entender. Dirigir la atención a la estructura tiene como consecuencia que operaciones clásicas, como la suma con números naturales, se hagan cada vez más móviles y puedan constituir sistemas cada vez más complejos (Aebli, 2001).

EJEMPLO 3-7

Escribir un procedimiento que muestre 3 veces en pantalla la frase "Esto es un camello".
R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya se encuentra claramente formulado.

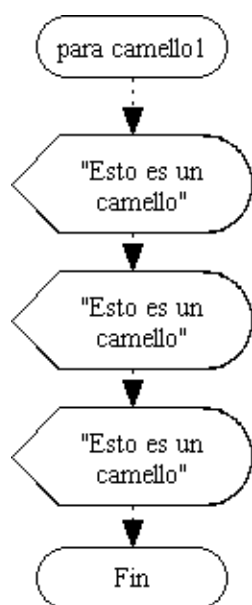
Resultados esperados: Que aparezca tres veces en pantalla la frase "Esto es un camello".

Datos disponibles: La frase dada.

Restricciones: Ninguna.

Procesos necesarios: Ninguno.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para camello1
  muestra [Esto es un camello]
  muestra [Esto es un camello]
  muestra [Esto es un camello]
fin
  
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



Este es un problema muy sencillo de resolver mediante la utilización de una estructura secuencial. Pedir a los estudiantes que analicen qué tan eficiente sería utilizar la misma estructura si el enunciado del problema fuera: Escribir un procedimiento que muestre 60 veces en pantalla la frase "Esto es un camello". ¿Qué habría que hacer si se cambiara 3 veces por 60?

ACTIVIDADES

1. Diseñar un algoritmo que pida al usuario dos números y calcule la suma, la resta, la multiplicación y la división del primero por el segundo. Traducir el algoritmo al lenguaje Logo y probarlo.
2. Diseñar un algoritmo para calcular cuántos litros caben en un tanque. Los datos de entrada (profundidad, largo y ancho) deben estar dados en metros. Se debe tener presente que 1 litro equivale a 1 dm³. Traducir el algoritmo al lenguaje Logo y probarlo.

ESTRUCTURA ITERATIVA (REPETICIÓN)

La estructura iterativa o de repetición permite ejecutar una o varias instrucciones, un número determinado de veces o, indefinidamente, mientras se cumpla una condición. Esta estructura ayuda a simplificar los algoritmos, ahorrando tiempo valioso a quien resuelve problemas con ayuda del computador.

En programación existen al menos dos tipos de estructuras repetitivas, las cuales a su vez tienen variantes en los diferentes lenguajes de programación. La característica común es que ambos tipos permiten ejecutar una o varias instrucciones:

- un número determinado de veces.
- mientras se cumpla una condición.

Debido a que esta guía está diseñada para educación básica, solo se cubre aquí el primer tipo de estructura repetitiva: Ejecutar una o varias instrucciones un número determinado de veces.

La cantidad de horas disponibles para informática es otro de los factores que impiden desarrollar un programa curricular con todas las variantes que puede ofrecer la estructura repetitiva. Por lo general, la mayoría de instituciones educativas destina una o dos horas semanales a la clase de informática, tiempo muy limitado para cubrir todas las variantes que ofrecen los lenguajes de programación. Ante esta situación, se requiere concentrar esfuerzos en lograr la comprensión de conceptos fundamentales de los tres tipos básicos de estructura (secuencial, repetitiva y condicional). Esto solo se consigue resolviendo una buena cantidad de problemas cuya solución requiera de estas estructuras en forma combinada.

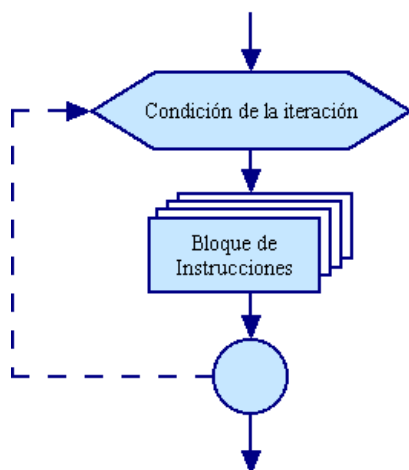


Ilustración 3-7: Modelo de estructura iterativa.

En MicroMundos, la estructura repetitiva se implementa con los Mandos *repite* y *cumpleveces*.

La sintaxis de *repite* en MMP es:

repite número [lista-de-instrucciones]

ejemplo: *repite 60 [muestra "hola"]*
donde *número* (60) indica las veces que se ejecutará la *lista-de-instrucciones* ([muestra "hola"]).

La sintaxis de *cumpleveces* en MMP es:

cumpleveces [serie] [lista-de-instrucciones]

ejemplo: *cumpleveces [i 8][muestra :i]*

donde *serie* ([i 8]) indica el nombre del parámetro (*i*) y el número de veces (8) que se ejecutará la *lista-de-instrucciones* ([muestra :i]).

Por su parte, la estructura repetitiva se implementa en Scratch con los Mandos repetir (n veces); repetir hasta que; por siempre; por siempre si <una condición es verdadera>:



EJEMPLO 3-8

Escribir un procedimiento que muestre 85 veces en pantalla la frase "Esto es un camello".

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya se encuentra claramente formulado.

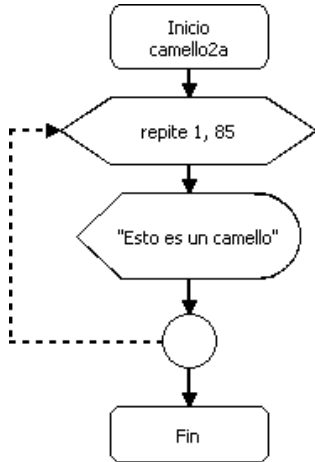
Resultados esperados: Que aparezca 85 veces en pantalla la frase "Esto es un camello".

Datos disponibles: La frase dada.

Restricciones: Ninguna.

Procesos necesarios: Mostrar la frase mencionada 85 veces.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

Con el uso del comando repite:

```

para camello2a
  repite 85
  [
    muestra [Esto es un camello]
  ]
fin
  
```

Ahora con el uso del comando cumpleveces:

```

para camello2b
  cumpleveces [i 85]
  [
    muestra [Esto es un camello]
  ]
fin
  
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



Un problema similar fue resuelto en el Ejemplo 3-7 de la estructura secuencial. En ese ejemplo se escribió el procedimiento *camello1* que mostraba tres veces en pantalla la frase “Esto es un camello”. Tal como debieron advertirlo los estudiantes, resolver este nuevo enunciado agregando instrucciones al procedimiento *camello1* no es práctico. Por eso, en este ejemplo se diseñó un algoritmo muy sencillo mediante la utilización de una estructura iterativa que repite la frase 85 veces. El número de veces que se repite la frase no tiene incidencia en la estructura del algoritmo, sea este 85 ó 1385. Es muy importante que los estudiantes tengan muy claro la diferencia entre los procedimientos *camello1* y *camello2a*.

En el Ejemplo 3-4 se solicitó a los estudiantes dibujar un

cuadrado cuyos lados fueran variables. Es muy probable que en una primera aproximación a la solución de este problema ellos elaboren un procedimiento similar al que se planteó en ese ejemplo, utilizando una estructura secuencial:

```

para cuadrado :lado
  limpia
  cp
  adelante :lado
  derecha 90
  adelante :lado
  derecha 90
  adelante :lado
  derecha 90
  adelante :lado
fin
  
```

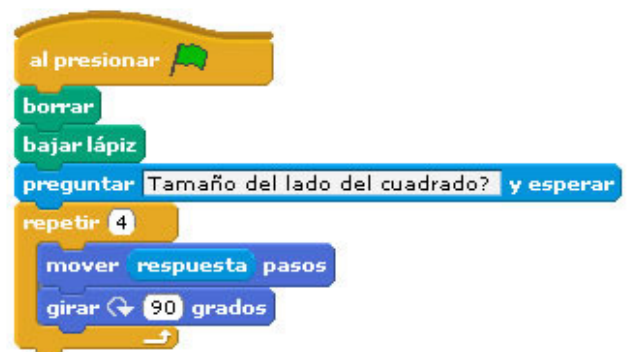
Un análisis más elaborado de este problema permitirá a los estudiantes adquirir una conciencia mayor de la organización global de un cuadrado. Determinar los elementos comunes presentes en todos los cuadrados permite identificar qué permanece estático (cuatro segmentos de recta iguales y cuatro ángulos iguales) y qué es lo que cambia (longitud de los segmentos). Aquello que cambia entre un cuadrado y otro se puede tratar como un parámetro. Además, este análisis permite descubrir que es posible utilizar otro tipo de estructura de control para elaborar el mismo dibujo:

TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para cuadrado :lado
  limpia
  cp
  repite 4
  [
    adelante :lado
    derecha 90
  ]
fin
  
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



Esta situación evidencia cómo en las actividades de programación el estudiante debe utilizar conocimientos adquiridos con anterioridad, cómo la etapa de análisis favorece y alienta el rigor y la disciplina en el razonamiento y cómo ese análisis puede conducir a nuevos descubrimientos que deriven en reorganizaciones del pensamiento, reestructuraciones de esquemas, etc. (Dufoyer, 1991). Además, algunos

psicólogos han llegado a sugerir que la programación alienta el estudio de las matemáticas, facilita la comprensión de conceptos de esta disciplina, admite explorar activamente campos de conocimiento, permite desarrollar habilidades y ofrece un lenguaje que permite describir la forma personal de resolver problemas.

EJEMPLO 3-9

Calcular el valor de la sumatoria: $1 + 2 + 3 + 4 + 5 + \dots + 100$.

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya se encuentra claramente formulado.

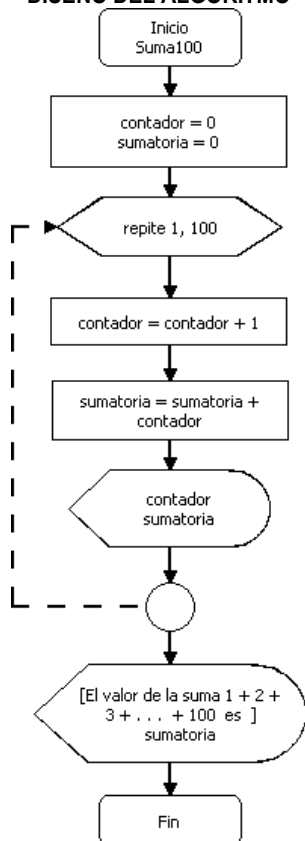
Resultados esperados: El resultado de la suma de los números entre 1 y 100.

Datos disponibles: El rango de números dado.

Restricciones: Ninguna.

Procesos necesarios: guardar el número 0 en una variable e incrementarla en 1 cada vez que se ejecute el ciclo repetitivo. Guardar 0 en otra variable e ir acumulando en ella su propio valor más el valor de la primera variable.

DISEÑO DEL ALGORITMO



Este algoritmo utiliza una operación muy útil en programación:

$sumatoria = sumatoria + contador$

Consiste en almacenar en una variable *sumatoria* el valor de ella misma (sumatoria) más otro valor variable (*contador*). Es muy utilizada para acumular valores.

TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

Con el Mando repite:

para suma100a
b nombres

```

da "contador 0
da "sumatoria 0
repite 100
[
  da "contador :contador + 1
  da "sumatoria :sumatoria + :contador
  muestra nombres
]
muestra frase
[El valor de la suma 1 + 2 + 3 + ... + 100 es ]:sumatoria
fin
  
```

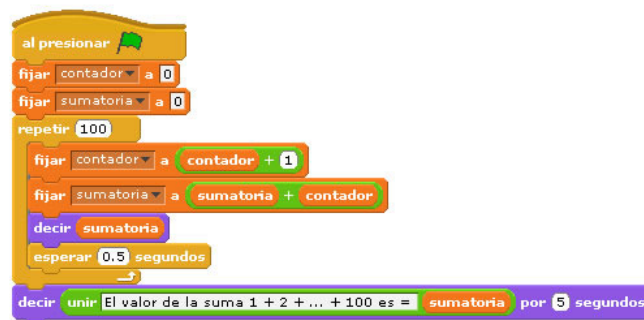
Ahora el mismo algoritmo pero con el Mando cumpleveces de MicroMundos:

```

para suma100b
  b nombres
  da "sumatoria 0
  cumpleveces [contador 100]
  [
    da "sumatoria :sumatoria + :contador + 1
    muestra nombres
  ]
  muestra frase
  [El valor de la suma 1 + 2 + 3 + ... + 100 es ]:sumatoria
fin
  
```

Los procedimientos *suma100a* y *suma100b* son equivalentes, realizan la misma tarea. La primitiva de MicroMundos *cumpleveces* utilizada en el procedimiento *suma100b* tiene una ventaja adicional con respecto a *repite*: incorpora una variable que aumenta en uno su valor cada vez que se ejecuta un ciclo de la estructura iterativa. La variable, que en este caso se llama *contador* inicia en 0 y termina en 99, para un total de 100 ciclos. Por este motivo se necesita sumarle uno a *contador* para que tome valores entre 1 y 100.

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



EJEMPLO 3-10

La profesora Ángela Cristina necesita calcular la nota definitiva para cada uno de los 22 alumnos que asisten a su curso de geometría. Ella realizó a todos sus estudiantes, en el primer periodo del año lectivo, dos exámenes y asignó un trabajo de investigación. ¿Cómo puedes ayudarle?

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Se requiere calcular un promedio de tres notas para cada uno de los 22 alumnos.

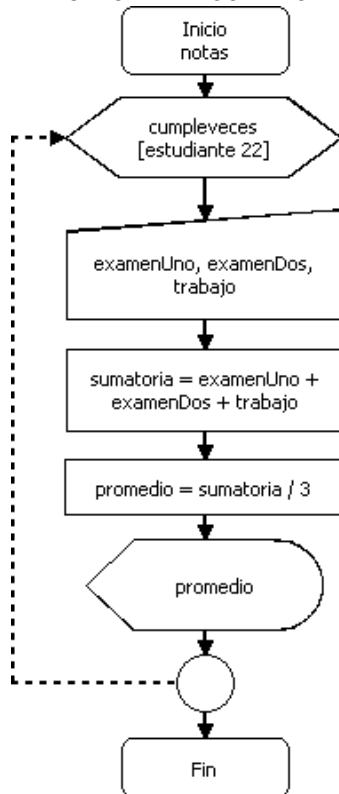
Resultados esperados: La nota definitiva de cada uno de los 22 alumnos.

Datos disponibles: El número de alumnos: 22. Las notas de cada alumno las debe digitar la profesora.

Restricciones: Cada una de las tres notas tienen el mismo porcentaje en la nota definitiva. Tres notas por alumno y 22 alumnos.

Procesos necesarios: Para cada uno de los 22 alumnos: Leer las tres notas, sumarlas, calcular el promedio y mostrar el promedio.

DISEÑO DEL ALGORITMO

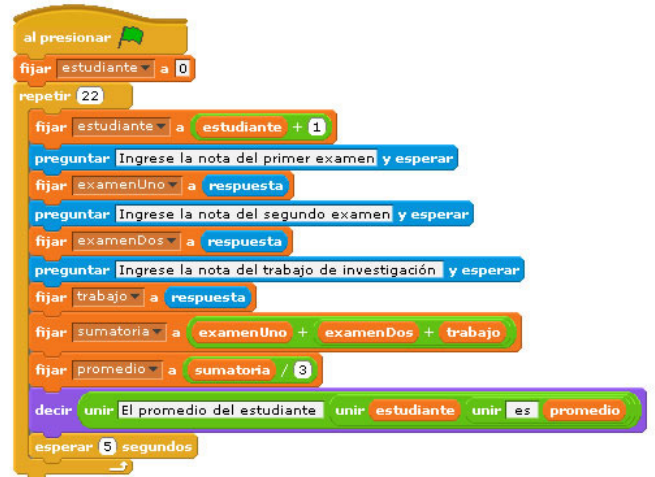


TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

para notas

```
cumpleveces [estudiante 22]
[
  pregunta [Ingrese la nota del primer examen]
  da "examenUno respuesta
  pregunta [Ingrese la nota del segundo examen]
  da "examenDos respuesta
  pregunta [Ingrese la nota del trabajo de investigación]
  da "trabajo respuesta
  da "sumatoria :examenUno + :examenDos + :trabajo
  da "promedio :sumatoria / 3
  muestra (frase [El promedio del estudiante] :estudiante + 1 [ es
]
:promedio)
]
Fin
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



El procedimiento *notas* realiza la misma tarea 22 veces: leer tres notas, sumarlas, promediarlas y mostrar el promedio. La manera más eficiente de resolver este problema es mediante una estructura iterativa. Nótese que los valores de las notas que ingresa el usuario no se validan, esto puede ocasionar que alguien digite una nota de, por ejemplo, 960; lo que dará como resultado un promedio fuera del rango permitido. Este aspecto se atiende con la siguiente estructura, la condicional.

Los ejemplos anteriores ilustran la solución de problemas mediante la utilización de estructuras repetitivas. En ellos se puede apreciar claramente la forma como esta estructura simplifica algunas soluciones, en comparación con soluciones cuya estructura es secuencial.

Por otro lado, la estructura repetitiva es muy apropiada para explorar los conceptos de multiplicación (suma cuyos sumandos son iguales) y potenciación (multiplicación de factores iguales).

EJEMPLO 3-11

Elaborar un procedimiento para calcular tablas de multiplicar. El usuario debe ingresar qué tabla de multiplicar desea.

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Ya se encuentra claramente formulado.

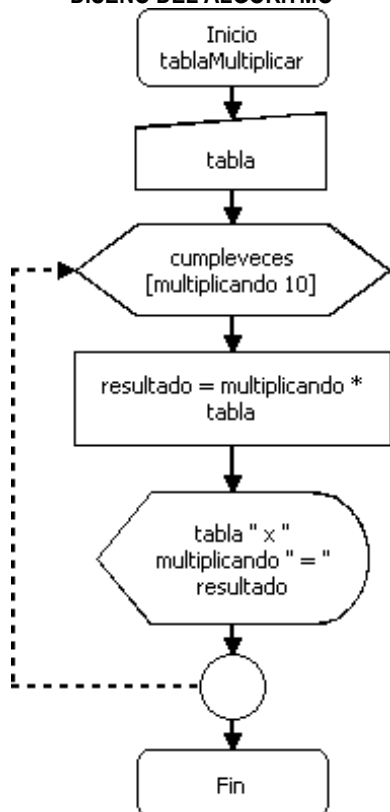
Resultados esperados: La tabla de multiplicar que el usuario indique.

Datos disponibles: El número de la tabla (indicada por el usuario).

Restricciones: Ninguna.

Procesos necesarios: pedir al usuario que ingrese la tabla de multiplicar que desea. Guardar ese valor en una variable (tabla). Multiplicar cada uno de los valores entre 0 y 9 por la variable tabla. Mostrar el resultado de cada multiplicación.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para tablaMultiplicar
  pregunta [¿Qué tabla de multiplicar desea? ]
  da "tabla respuesta
  cumpleveces [multiplicando 10]
  [
    da "resultado :multiplicando * :tabla
    muestra (frase :tabla [x] :multiplicando [=] :resultado)
  ]
fin
  
```

Nótese que, al igual que en el ejemplo 3-10, el reportero *frase* devuelve una lista formada por sus entradas (palabras o listas). *frase* puede tomar más de dos entradas cuando se utilizan paréntesis para encerrar el reportero y sus entradas, las cuales pueden incluir texto entre corchetes:

```

muestra (frase [texto 1] :variable1 [ texto2] :variable2 + 1
:variable3 [texto3])
  
```

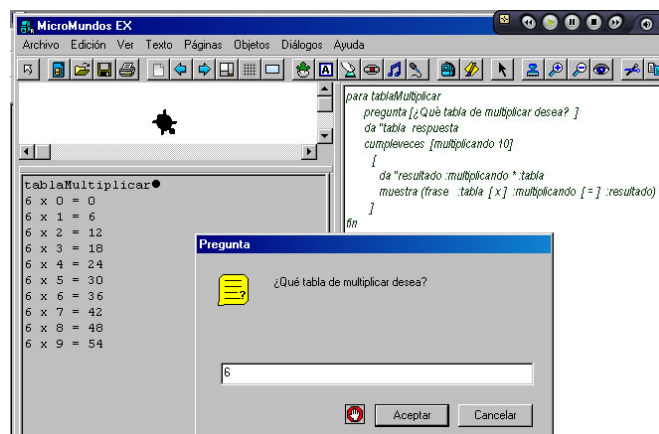


Ilustración 3-8: Ejecución del procedimiento tablaMultiplicar.

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



La estructura repetitiva es muy apropiada para reforzar los conceptos de multiplicación y potenciación. La suma $6 + 6 + 6 + 6 + 6 + 6 + 6 + 6$ puede expresarse como una multiplicación: $6 * 7$; de forma similar, la multiplicación $3 * 3 * 3 * 3$ puede expresarse como una potencia: $3^4 = 81$. El número 3, que se multiplica varias veces, se conoce como base; 4, la cantidad de veces que se debe multiplicar la base, se conoce como exponente, y 81, el resultado, se conoce como potencia. Se lee: 81 es la potencia de 3 elevado a la 4.

Existe una leyenda muy antigua sobre el origen del juego de ajedrez. La leyenda evidencia claramente la velocidad con que aumenta una progresión geométrica y es un buen punto de partida para empezar a resolver problemas de potenciación con ayuda del computador.

Historia Curiosa

Un día, en la India, un joven brahmán llamado Lahur Sessa pidió una audiencia con el Rey para obsequiarle el juego que había inventado. La curiosidad del rey lo llevó a conceder la cita que pedía el joven Sessa. El rey quedó maravillado y aprendió rápidamente las reglas de aquel juego que consistía de un tablero cuadrado dividido en sesenta y cuatro cuadritos iguales (32 blancos y 32 negros); sobre este tablero se ubicaban dos colecciones de piezas, que se distinguían unas de otras por el color, blancas y negras, repitiendo simétricamente los motivos y subordinadas a reglas que permitían de varios modos su movimiento.

Algún tiempo después, el rey mandó llamar a su presencia al joven brahmán y dirigiéndose a él le dijo:

- Quiero recompensarte, amigo mío, por este maravilloso obsequio, que de tanto me sirvió para aliviar viejas angustias. Pide, pues, lo que desees, para que yo pueda demostrar, una vez más, como soy de agradecido con aquellos que son dignos de una recompensa.

Ante tal ofrecimiento, el joven respondió:

- Voy, pues, a aceptar por el juego que inventé, una recompensa que corresponda a vuestra generosidad; no deseo, sin embargo, ni oro, ni tierras, ni palacios. Deseo mi recompensa en granos de trigo.

-¿Granos de trigo? –exclamó el rey, sin ocultar la sorpresa que le causara semejante propuesta-. ¿Cómo podré pagarte con tan insignificante moneda?

-Nada más simple –aclará Sessa-. Dadme un grano de trigo por la primera casilla del tablero, dos por la segunda, cuatro por la tercera, ocho por la cuarta, y así sucesivamente hasta la sexagésima cuarta y última casilla del tablero.

No sólo el rey, sino también los visires y venerables bracmanes, se rieron estrepitosamente al oír la extraña solicitud del joven.

Insensato –exclamó el rey-. ¿Dónde aprendiste tan grande indiferencia por la fortuna? La recompensa que me pides es ridícula.

Mando llamar al rey a los algebristas más hábiles de la Corte y les ordenó calculasen la porción de trigo que Sessa pretendía.

Los sabios matemáticos, al cabo de algunas horas de realizar cálculos dispendiosos, volvieron al salón para hacer conocer al rey el resultado completo de sus cálculos.

Preguntóles el rey, interrumpiendo el juego:

-¿Con cuantos granos de trigo podré cumplir, finalmente, con la promesa hecha al joven Sessa?

-Rey magnánimo –declaró el más sabio de los geómetras-: calculamos el número de granos de trigo que constituirá la recompensa elegida por Sessa, y obtuvimos un número cuya magnitud es inconcebible para la imaginación humana (el número en cuestión contiene 20 guarismos y es el siguiente: 18.446.744.073.709. 551. 615. Se obtiene restando 1 a la potencia 64 de 2).

-La cantidad de trigo que debe entregarse a Lahur Sessa –continúo el geómetra- equivale a una montaña que teniendo por base la ciudad de Taligana, fuese 100 veces más alta que el Himalaya. La India entera, sembrados todos sus campos, y destruidas todas sus ciudades, no produciría en un siglo la cantidad de trigo que, por vuestra promesa, debe entregarse al joven Sessa.

¿Cómo describir aquí la sorpresa y el asombro que esas palabras causaron al Rey Ladava y a sus dignos visires? El soberano hindú se veía, por primera vez, en la imposibilidad de cumplir una promesa.

Lahur Sessa –refiere la leyenda de la época-, como buen súbdito, no quiso dejar afligido a su soberano. Después de declarar públicamente que se desdecía del pedido que formulara, se dirigió respetuosamente al monarca y le dijo: los hombres más precavidos, eluden no sólo la apariencia engañosa de los números, sino también la falsa modestia de los ambiciosos.

El rey, olvidando la montaña de trigo que prometiera al joven bracmán, lo nombró su primer ministro.

(Tomado del libro “El hombre que calculaba” escrito por Malba Tahan)

EJEMPLO 3-12

Elaborar un procedimiento para ayudar a los hábiles algebristas de la corte del Rey Ladava con el cálculo del número de granos de trigo que deben entregar a Lahur Sessa como pago por haber inventado el juego de ajedrez.

RI.

ANÁLISIS DEL PROBLEMA

Formular el problema: Es un problema de multiplicaciones de factores iguales que pueden expresarse en forma de potencias; además, para llegar al resultado final se deben acumular los resultados parciales.

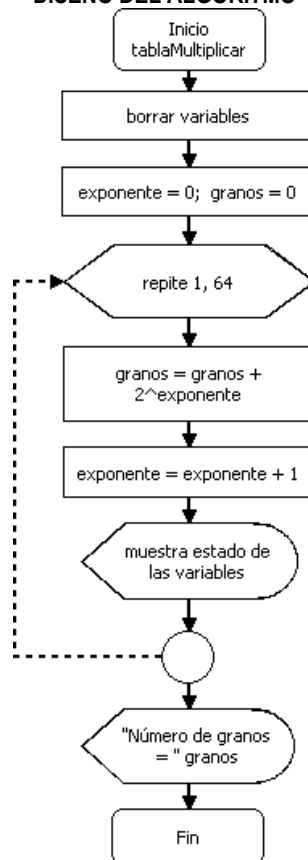
Resultados esperados: El número de granos que el Rey Ladava debe entregar a Lahur Sessa.

Datos disponibles: El número de cuadros del tablero de ajedrez (64) y la regla dada por Sessa: “un grano de trigo por la primera casilla del tablero, dos por la segunda, cuatro por la tercera, ocho por la cuarta, y así sucesivamente hasta la sexagésima cuarta y última casilla del tablero”.

Restricciones: Aplicar la regla planteada por Sessa.

Procesos necesarios: Un ciclo que se repita 64 veces. En cada iteración se debe acumular en una variable (granos), su propio valor más el resultado de 2 elevado a un exponente que aumenta su valor en uno con cada iteración.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

Utilizando el comando repite:

para ajedrez

bnombres

da "exponente 1

da "granos 0

repite 64

[

da "granos :granos + potencia 2 :exponente

da "exponente :exponente + 1

muestra nombres

]

muestra frase
 [El número de granos que ganó Sessa es]:granos
 fin

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



Nótese que como Scratch (hasta la versión 1.4) no incluye un operador para calcular potencias, una forma de solucionar esta situación consiste en elaborar un subprocedimiento llamado potenciación para realizar este cálculo a partir de multiplicaciones sucesivas.

- ¿Obtuvieron los estudiantes el mismo resultado (18446744073709551615)?
- ¿El computador también tardó varias horas para calcular el resultado final?
- ¿Qué función cumplen los comandos nombres y nombres en MicroMundos?
- ¿Cuál es la diferencia entre crecimiento aritmético y crecimiento geométrico?

ACTIVIDADES

Los estudiantes deben encontrar solución a los siguientes problemas empleando la metodología expuesta en la Unidad 1: Analizar el problema (formulación del problema, resultados esperados, datos disponibles, restricciones y procesos necesarios), diseñar el algoritmo, traducirlo al lenguaje Logo y probar el programa resultante.

1. Elaborar un procedimiento que calcule y muestre las áreas de 100 círculos con radio de 1 a 100 cm.
2. Elaborar un procedimiento que calcule y muestre el cuadrado de los números 1 a 90.




3. Elaborar un procedimiento que le reporte al electricista de un edificio recién construido cuantos bombillos debe comprar. Se sabe que el edificio tiene 8 pisos, 8 apartamento en cada piso y cada apartamento tiene 8 bombillos. En la solución se debe emplear una estructura repetitiva.
4. Elaborar un procedimiento que calcule el área de cualquier cubo.
5. Elaborar un procedimiento que dibuje polígonos regulares de 5, 6, 7, 8 y 9 lados. El usuario debe indicar el número de lados del polígono.

ESTRUCTURA CONDICIONAL

Es fundamental que los estudiantes presten atención especial a las estructuras que utilizan para resolver problemas y las reconozcan para lograr mayor control sobre la solución planteada. De esta manera, la programación de computadores les ayuda a planear conscientemente las secuencias de acciones que resuelven un problema planteado y las estructuras involucradas en una solución dada.

La estructura condicional se utiliza para indicarle al computador que debe evaluar una condición y, a partir del resultado, ejecutar el bloque de instrucciones correspondiente. La forma más común está compuesta por una proposición (condición) que se evalúa y dos bloques de instrucciones que se ejecutan, uno cuando la condición es verdadera (selección simple y doble) y otro cuando ésta es falsa (únicamente en la selección doble). Algunos autores se refieren a este tipo de estructura como estructura de selección, estructura selectiva o estructura de decisión; en esta guía, todas estas denominaciones son consideradas sinónimas.

Para que una proposición (frase declarativa) sea válida, debe poder afirmarse que es verdadera o falsa. En programación, se utilizan operadores relacionales (<, =, >) para establecer la relación que existe entre dos elementos de la proposición. Por ejemplo, "La calificación de Esteban en Historia es mayor que 6.0", es una proposición válida. De una parte tenemos "La calificación de Esteban en Historia" (A) y, de la otra, el valor "6.0" (B); de A con respecto a B, se afirma que "A es mayor que B", por lo tanto, la relación existente entre A y B es "ser mayor que". Para que el computador entienda esta proposición, debe expresarse así: "*:calificación > 6.0*", donde *:calificación* es la variable que contiene el valor de "la calificación de Esteban en Historia".

OPERADOR	DESCRIPCIÓN	EJEMPLO
= 	Igual que	:ánguloUno = 90 :tipo = "SI"
< 	Menor que	:ánguloUno < 90
> 	Mayor que	:ánguloUno > 90

Adicionalmente, las proposiciones pueden ser sencillas o compuestas. Las proposiciones compuestas se forman con dos o más proposiciones sencillas unidas por operadores lógicos (y, o, no). Cuando se unen dos proposiciones por medio del operador lógico "y", significa que ambas proposiciones deben ser verdaderas (conjunción). Cuando se unen dos proposiciones por medio del operador lógico "o", significa que por lo menos una de las dos proposiciones debe ser verdadera (disyunción).

Por su parte, un bloque de instrucciones puede contener una o varias instrucciones que se ejecutan una detrás de otra. La estructura condicional tiene tres variantes:

- selección simple.
- selección doble.
- selección múltiple.

Las estructuras condicionales simple y doble evalúan una proposición (condición) que devuelve como resultado únicamente dos valores posibles y excluyentes: verdadero o falso. En cambio, la estructura condicional de selección múltiple permite que la condición devuelva más de un valor posible y que para cada uno de esos valores se ejecute el bloque de instrucciones correspondiente. Por ejemplo, una situación típica de selección múltiple es cuando la incorporación al ejército, de un joven al terminar sus estudios de educación media, depende del color de una balota: si saca una balota roja, su incorporación al ejército es inmediata; si es azul, la incorporación será en julio; y si es blanca, el estudiante no debe prestar servicio militar. En esta situación hay tres valores posibles y cada uno de esos valores implica la ejecución de una instrucción diferente (Jiménez, 2002).

Debido al alcance de esta guía, solo se cubren aquí los dos primeros tipos de estructura condicional: simple y doble.

Selección simple

La estructura condicional de selección simple ejecuta un bloque de instrucciones cuando la proposición (condición) es verdadera; si esta es falsa, no hace nada.

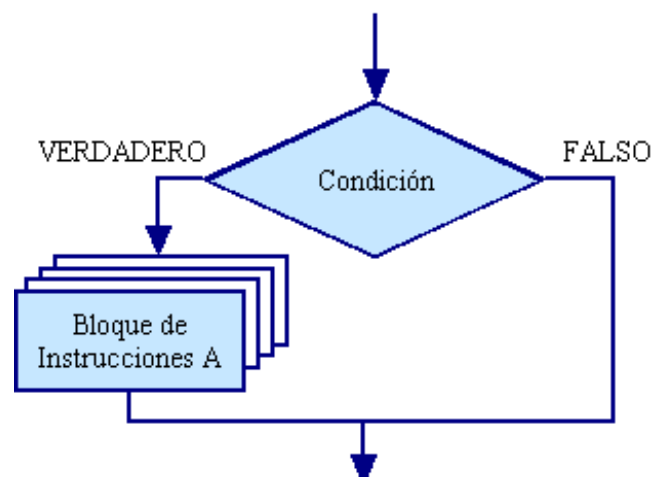


Ilustración 3-9: Modelo de estructura condicional simple.

Para la estructura condicional de selección simple, MicroMundos ofrece el comando "si". La sintaxis es:

si cierto-o-falso

```
[
  lista-de-instrucciones
]
```

el comando “si” ejecuta la lista-de-instrucciones únicamente si al evaluarse la proposición, esta devuelve cierto (verdadero).

Por su parte, la estructura condicional de selección simple se implementa en Scratch con el bloque “si” (condición):



EJEMPLO 3-13

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```
para selecciónSimple
  pregunta [Ingrese el ángulo]
  da "ánguloUno respuesta
  si :ánguloUno = 90
  [
    da "reportar [ es un ángulo recto]
    muestra frase :ánguloUno :reportar
  ]
fin
```

En este ejemplo, *cierto-o-falso* (:ánguloUno = 90) indica la condición que se debe evaluar la cual puede devolver únicamente uno de dos valores posibles: verdadero o falso. En caso de ser verdadera la proposición, se ejecuta la [lista-de-instrucciones] indicada entre corchetes; esta puede contener una o varias instrucciones. Cuando es falsa la proposición evaluada, no se ejecutan instrucciones.

Además, se puede observar un recurso gráfico muy importante para dar claridad a las líneas de código de los procedimientos en MicroMundos: (1) dejar líneas en blanco para dividir bloques de código; (2) utilizar sangrías para indicar porciones de código subordinadas a un comando; (3) abrir y cerrar los corchetes que indican bloques de código en una línea a parte, de tal

forma que se aprecie muy claramente dónde inicia y dónde termina una *lista-de-instrucciones*.

Selección doble

La estructura condicional de selección doble ejecuta un bloque de instrucciones (A) cuando la proposición (condición) es verdadera y un bloque diferente (B) cuando esta es falsa.

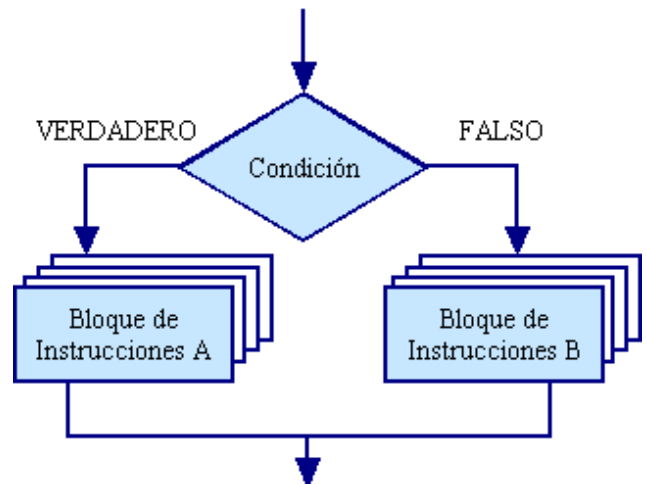


Ilustración 3-10: Modelo de estructura condicional doble.

Para la estructura condicional de selección doble, MicroMundos ofrece el comando “siotro”. La sintaxis es:

```
siotro cierto-o-falso
[
  lista-de-instrucciones-A
]
[
  lista-de-instrucciones-B
]
```

El comando “siotro” ejecuta la *lista-de-instrucciones-A* si al evaluarse la proposición, esta es verdadera. Si la proposición es falsa, se ejecuta la *lista-de-instrucciones-B*. Ambas listas de instrucciones se deben indicar entre corchetes [] y pueden estar compuestas por una o más instrucciones.

En Scratch, la estructura condicional de selección doble se implementa con el bloque “si (condición) si no”:



EJEMPLO 3-14

```
para selecciónDoble
  pregunta [Ingrese el ángulo]
  da "ánguloUno respuesta
  siotro (:ánguloUno = 90)
  [
    da "reportar [ es un ángulo recto]
```

```

]
[
  da "reportar [ NO es un ángulo recto]
]
muestra frase :ánguloUno :reportar
fin

```



En este ejemplo, *cierto-o-falso* (:ánguloUno = 90) indica la proposición que se debe evaluar, la cual solo puede devolver uno de dos valores posibles: verdadero o falso. En caso de que la proposición sea verdadera, se ejecuta la [lista-de-instrucciones-A] indicada entre corchetes: ([da "reportar [es un ángulo recto]]). Cuando la proposición evaluada es falsa, se ejecuta la [lista-de-instrucciones-B] ([da "reportar [NO es un ángulo recto]]).

Nótese que en MicroMundos la instrucción *muestra frase :ánguloUno :reportar* se encuentra fuera de los corchetes; por tanto, se ejecutará sin importar si la proposición es verdadera o falsa. Además, ejemplifica muy bien el concepto de variable ya que el valor del ángulo se guarda en la variable denominada *ánguloUno* y el aviso que se debe mostrar acerca de si el ángulo es o no recto, también se guarda en una variable (*reportar*).

Tanto en la estructura de selección simple como en la doble se debe tener en cuenta lo siguiente:

- La proposición debe ser una frase declarativa, la cual se pueda afirmar o negar.
- En MicroMundos, se requiere que en el encabezado vayan las palabras reservadas *si* y *siotro* respectivamente.
- En MicroMundos, cuando la proposición es sencilla (sin operadores lógicos) no es necesario que vaya entre paréntesis; si es compuesta (dos o más proposiciones unidas con operadores lógicos como: o, y, no) tiene que encerrarse con paréntesis. Como en el primer caso no sobran los paréntesis (no genera error), es recomendable utilizarlos siempre. Por ejemplo: (*ánguloUno = 90*) es una proposición sencilla equivalente a *ánguloUno = 90*, pero es mejor utilizar la primera forma.
- En MicroMundos, las listas de instrucciones deben estar agrupadas con corchetes, estos indican dónde

empieza y dónde termina la lista que conforma el bloque que se debe ejecutar.

EJEMPLO 3-15

Un estudiante aprueba un examen cuando obtiene una calificación mayor o igual a seis. Elaborar un procedimiento que pida al usuario una calificación, aplique el criterio de aprobación e imprima "Aprobado" o "Reprobado", según sea el caso.

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Es un problema sencillo de selección doble.

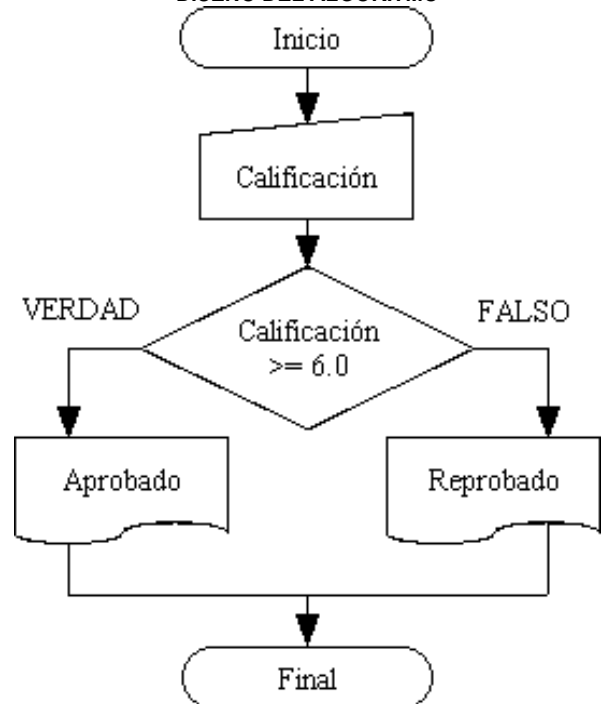
Resultados esperados: Un aviso que reporte si el estudiante "Aprobó" o "Reprobó" el examen.

Datos disponibles: La calificación ingresada por el usuario. Para aprobar, la nota debe ser mayor o igual a 6.0.

Restricciones: Aplicar el criterio de aprobación.

Procesos necesarios: Solicitar al usuario que ingrese la calificación. Evaluar si la calificación es igual o superior a 6.0; en caso de ser verdadero, reportar "Aprobado"; en caso contrario, reportar "Reprobado".

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para aprueba
local "calificación
pregunta [Ingrese la Calificación]
da "calificación respuesta
siotro o :calificación > 6.0 :calificación = 6.0
[
  anuncia [Aprobado]
]
[
  anuncia [Reprobado]
]
fin

```


TRADUCCIÓN DEL ALGORITMO EN SCRATCH



En este ejemplo, se puede observar la sintaxis de los operadores lógicos (y, o, no), mediante los cuales se unen proposiciones sencillas para construir proposiciones compuestas. Estos deben ir en seguida del paréntesis que abre la proposición:

siotro o :calificación > 6.0 :calificación = 6.0

La proposición se lee así:

"calificación mayor que 6.0 o calificación igual a 6.0".

Proposiciones

Según Piaget (1993), las operaciones verbales o proposicionales surgen hacia los once o doce años con la capacidad para razonar por hipótesis. Esta capacidad hipotética-deductiva es la que hace posible que los niños entre los once y los catorce años piensen en términos de proposiciones y no únicamente sobre objetos; acepten cualquier tipo de dato como puramente hipotético y razonen correctamente a partir de él; deduzcan las implicaciones de enunciados posibles y así distingan entre lo posible y lo necesario; formulen todas las hipótesis posibles relativas a los factores que entran en juego en una actividad y organicen la información en función de estos factores.

De acuerdo con Piaget y sus seguidores, es en este estado del desarrollo cognitivo en el que se constituye un conjunto de estructuras proposicionales basadas en lo que en teoría de conjuntos se llama un "conjunto de todos los subconjuntos". Esta estructura está conformada por operaciones equivalentes a ciertas estructuras del pensamiento verbal, tales como implicación ($p \rightarrow q$: si..., entonces...; si la hipótesis p es verdadera, entonces la consecuencia q se sigue necesariamente); disyunción ($p \vee q$; ó p , ó q , ó los dos); unión ($p \wedge q$); incompatibilidad ($p \mid q$).

Una forma efectiva para iniciar a los estudiantes más pequeños en el tema de las proposiciones puede ser la propuesta por Marquínez & Sanz (1988): empezar con cadenas de palabras (sin sentido), avanzar a expresiones (con sentido incompleto), continuar con oraciones (con sentido completo) y finalizar con

proposiciones simples y compuestas (calificables como falsas o verdaderas).

EJEMPLO

"La escuela tiene pan francés caído de China" es una CADENA de palabras que carece de sentido.

"Los amigos de lo ajeno" es una EXPRESIÓN que tiene sentido pero no completo.

"Ojalá que mañana no llueva" es una ORACIÓN con sentido completo pero no es calificable.

"Simón Bolívar nació en Santa Marta" es una PROPOSICIÓN que puede calificarse de verdadera o falsa.

ACTIVIDAD

Escribir en el espacio si la propuesta corresponde a una cadena, expresión, oración o proposición:

- _____ prohibido fumar en el salón de clase.
- _____ el oro es un elemento de la tabla periódica.
- _____ calle perfecta para perro azul.
- _____ el carro sedán azul.
- _____ ¿qué hora es?
- _____ el nevado del Ruiz es un volcán.
- _____ Simón Bolívar murió en Santa Marta
- _____ Cali es una ciudad colombiana.
- _____ camisa cuadrada por carro naciente.
- _____ Perú y Chile son países Iberoamericanos.
- _____ el cuaderno verde de geometría.
- _____ está permitido subir las escaleras.
- _____ cuatro y diez son números menores que veinte.
- _____ si alguien es chileno, entonces es español.
- _____ hace mucho frío
- _____ en un lugar de la Mancha de cuyo nombre
- _____ ojalá no me llame.
- _____ apague la luz cuando salga.

Un curso de algoritmos y programación puede contribuir significativamente a desarrollar la capacidad hipotética-deductiva en la que el pensamiento no proceda de lo real a lo teórico, sino que parta de la teoría y establezca o verifique relaciones reales entre cosas. Concretamente, dos tipos de actividades pueden ayudar a lograr este propósito: utilizar estructuras condicionales las cuales están basadas en la operación de implicación (si..., entonces...) y formular enunciados declarativos compuestos (proposiciones simples unidas por los conectores lógicos "y", "ó") que el computador pueda evaluar como verdaderos o falsos. Adicionalmente, estos enunciados promueven el razonamiento por atribución o relación (Felipe es más joven que Ángela) en contraposición al razonamiento por predicados (Felipe es joven).

Precisamente, la estructura condicional utilizada en programación (si... entonces...) ofrece al estudiante oportunidades para desarrollar habilidades con proposiciones y relaciones de orden. Sin embargo hay que tener en cuenta que la construcción "si P entonces

S", que utilizan los lenguajes de programación MicroMundos y Scratch, es procedimental y no declarativa ya que hace énfasis en la acción y no en el concepto semántico de verdad (Iranzo, 2005). Mientras que en lógica se indica que entre P y S hay una relación de dependencia en la que al suceder P, necesariamente se causa S; en programación se indica que cuando P es verdadero, necesariamente se ejecuta un conjunto de instrucciones A y en caso de ser falso no se ejecuta ninguna instrucción (selección simple) o necesariamente se ejecuta un conjunto de instrucciones B (selección doble).

Según Bustamante (2007), "una proposición es una frase declarativa que puede ser afirmada o negada" y para Iranzo (2005) la lógica proposicional "se ocupa de los enunciados declarativos simples como un todo indivisible y que pueden combinarse mediante partículas lógicas denominadas conectores (no, y, o, si... entonces..., etc)". A esta lógica también se le conoce con el nombre de lógica de enunciados o lógica de conectores. De acuerdo con estos dos autores, los siguientes enunciados declarativos se pueden negar o afirmar, por lo tanto pueden considerarse proposiciones:

1. Cali es la capital del Valle del Cauca.
2. El cuatro es un número impar.
3. Seis es menor que doce.
4. El INSA es un colegio regentado por la comunidad de Padres Basilianos.
5. Andrés Pastrana es el presidente de Colombia.
6. Es verano
7. Hace calor


De las proposiciones primera, tercera, cuarta y quinta podemos decir que son verdaderas y de la segunda podemos afirmar que es falsa. Sin embargo, para poder afirmar que la cuarta proposición es verdadera, hay que disponer del conocimiento suficiente sobre este colegio ubicado en el barrio Andrés Sanín en la ciudad de Cali. Esto nos conduce a hacer otra consideración: establecer explícitamente si una proposición es verdadera o falsa puede resultar en algunos casos muy difícil o imposible. Por otra parte, la quinta proposición fue verdadera durante un lapso de tiempo (1998-2002).

En relación a las proposiciones sexta y séptima, su valor de verdad depende del momento en el cual se haga la afirmación. Esto nos lleva a otra forma de clasificar los enunciados declarativos: de acción cuando el sujeto no está determinado (6 y 7); de atribución cuando el sujeto es determinado y se le atribuye una propiedad (1, 2 y 5); y de relación cuando hay dos o más sujetos (3 y 4).

Con respecto a las relaciones de orden podemos decir que consisten en un par de elementos presentes en una proposición relacionados por medio de un atributo gradado. Por ejemplo, "el elemento A es *mayor o igual que* el elemento B" o "seis es *menor que* doce". Proposiciones en las cuales "mayor o igual que" y "menor que" son las relaciones de orden que se establecen entre los elementos A y B, y entre seis y

doce, respectivamente.

Hay que tener cuidado con el uso del lenguaje cotidiano en el que dos relaciones pueden ser equivalentes como "igual o superior a" y "mayor o igual que". En cambio, "entre 0 y 10, inclusive" y "entre 0 y 10" no son equivalentes; en la primera relación los valores 0 y 10 hacen que la proposición sea verdadera, en la segunda relación, no.

RELACIÓN	(MicroMundos) y Scratch
A es igual a B	(:a = :b) 
A es mayor que B	(:a > :b) 
A es mayor o igual que B	(o :a > :b :a = :b) 
A es como mínimo igual a B	(o :a > :b :a = :b) 
A es menor que B	(:a < :b) 
A es menor o igual que B	(o :a < :b :a = :b) 
A es al menos igual a B	(o :a < :b :a = :b) 
A está entre 0 y 10	(y :a > 0 :a < 10) 
A está entre 0 y 10, inclusive	(y (o :a > 0 :a = 0) (o :a < 10 :a = 10)) 

Un aspecto fundamental de la estructura condicional es la reflexión sobre el papel del lenguaje natural en la formulación y uso de relaciones de orden y de proposiciones. Diversos autores que se han ocupado de la lógica y el lenguaje han establecido tres categorías generales para el uso del lenguaje: informativa (suministra información definiendo, declarando, aclarando, describiendo), expresiva (expresa sentimientos, emociones, deseos) y directiva (busca inducir a alguien a que haga u omita algo). Son ejemplos de cada una de estas categorías lo siguiente:

Uso informativo:

- La línea recta es la más corta entre dos puntos de un plano.
- Colombia es un país andino
- Los noruegos son altos, delgados y de ojos azules.

Uso expresivo:

- Ojalá haga buen día mañana!
- Qué horror! no podría soportar algo tan doloroso.

Uso directivo:

- Prohibido fumar
- Cierre la puerta
- Se solicita comportarse bien

Para la programación y en especial para la estructura condicional, resulta imprescindible el uso informativo del lenguaje. Este se encarga de comunicar información mediante la formulación y afirmación o negación de proposiciones. El discurso informativo se utiliza para describir el mundo y para razonar sobre él, sin importar si las proposiciones son importantes o no, si son generales o específicas, o si son verdaderas o falsas (Copi & Cohen, 2000).

Los estudiantes deben estar en capacidad de distinguir el discurso informativo en un texto o en el planteamiento verbal de un problema. Pero en ciertos textos o planteamientos resulta difícil identificar de manera inmediata la existencia de proposiciones que se puedan contestar con un “verdadero” o con un “falso” (Solano, 1991). En lenguajes de programación como Logo es muy importante que las proposiciones se puedan expresar directamente, en forma de notación matemática o mediante texto. Para ello, es fundamental que los estudiantes identifiquen los componentes de las proposiciones (enunciados y relación entre ellos) y verifiquen que sean válidos. Luego determinen en cada proposición el sujeto (objetos o individuos acerca de los cuales se afirma algo) y el predicado (propiedad que posee el sujeto) y en seguida identifiquen con un nombre (identificador) al que puede variar (sujeto o predicado).

Por ejemplo, en la proposición número 1 “Cali es la capital del Valle del Cauca”, el sujeto es “Cali”, el predicado es “capital del Valle del Cauca” y la relación es de igualdad “es”. Se debe asignar un nombre al predicado (capitalValle) para guardar el valor “Cali”. En el caso de la proposición “Seis es menor que doce”, el sujeto es “Seis”, el predicado es “doce” y la relación es “menor que”.

Por otra parte, de las siete proposiciones planteadas, solo la número tres se puede expresar en notación matemática; las otras proposiciones hay que expresarlas como texto, con excepción de la número 2 que no se puede expresar directamente:

1. (:capitalValle = "Cali")
2. "El cuatro es un número impar" no se puede expresar directamente. Hay que elaborar un procedimiento para determinar si un número es par o impar.
3. (6 < 12)
4. (:rectorINSA = "Basiliano")
5. (:presidenteColombia = "Álvaro Uribe")
6. (:verano = true)
7. (:haceCalor = false)

TIP

Hay que tener cuidado cuando se copia de un procesador de texto una porción de texto que contenga comillas (") y se pega en el área de procedimientos de MicroMundos [4]. Las comillas (") que generan estos programas no son equivalentes a las comillas de MicroMundos [4] (").

También hay que tener cuidado cuando se quiere comparar un texto conformado por dos o más palabras, este debe encerrarse entre barras (/palabra1 palabra2/).

Otro aspecto a tener en cuenta con las proposiciones y que se debe trabajar con los estudiantes es la riqueza del lenguaje natural (Marquínez & Sanz, 1998). Por ejemplo, el conector lógico "y" (^) se presenta de diversas formas en el lenguaje común utilizado para formular problemas y los estudiantes deben aprender a identificarlo:

- Cali Y Medellín son ciudades ecuatorianas.
- Bogotá, Quito, Lima, Montevideo son ciudades capitales (*Bogotá es ciudad capital Y Quito es ciudad capital Y Lima es ciudad capital Y Montevideo es ciudad capital*)
- Luisa estudia, Cristina también (*Luisa estudia Y Cristina estudia*)
- En Bogotá hace frío, IGUALMENTE en Tunja (*En Bogotá hace frío Y en Tunja hace frío*)
- En Bogotá hace frío, DEL MISMO MODO en Tunja (*En Bogotá hace frío Y en Tunja hace frío*)
- En Bogotá hace frío, MIENTRAS QUE en Cartagena calor (*En Bogotá hace frío Y en Cartagena hace calor*)
- Ángela tiene un automóvil, PERO no sabe manejarlo aún (*Ángela tiene automóvil Y Ángela no sabe manejar automóvil*)
- Luisa no viene, SIN EMBARGO escribe correos electrónicos todos los días (*Luisa no viene Y Luisa escribe correos electrónicos todos los días*)
- Esteban no estudia, NO OBSTANTE quiere hacerlo (*Esteban no estudia Y Esteban quiere estudiar*)
- A PESAR DEL buen tiempo, no vamos a la piscina (*Hace buen tiempo Y no vamos a piscina*)
- PESE A QUE lo sabe, no lo puede decir (*Él lo sabe Y él no lo puede decir*)
- En Cali no hay energía eléctrica, TAMPOCO en Bogotá (*En Cali no hay energía eléctrica Y en Bogotá no hay energía eléctrica*)

Lo mismo ocurre con la determinación de si una proposición está expresada en afirmativo o en negativo:

- Colombia NO es un país europeo.
- El Nilo es un río Incontrolable (*El Nilo es un río que NO se puede controlar*)
- La vida humana en Marte es Imposible (*NO es posible la vida humana en Marte*)
- Luisa es una diseñadora DESconocida (*Luisa NO es conocida como diseñadora*)
- La aparición de cometas es un fenómeno DIScontinuo (*NO es continua la aparición de cometas*)
- Los animales son Amorales (*Los animales NO tienen moral*)
- Los castigos son ANTipedagógicos (*NO son pedagógicos los castigos*)

- NUNCA me ganó la lotería (*NO me he ganado la lotería*)
- Ricardo JAMÁS miente (*Ricardo NO ha mentado*)
- NINGÚN hombre colombiano usa falda (Los hombres colombianos NO usan falda)*

* Es una afirmación falsa ya que los hombres colombianos de la etnia guambiana si usan falda.

Un último aspecto a tener en cuenta son los cuantificadores que se utilizan en algunas proposiciones: todos, algunos, ningún, ninguno, sólo, hay, etc. Incluso, proposiciones que no contienen cuantificadores se pueden transformar en proposiciones cuantificadas: "Cada planeta gira sobre su eje" se puede escribir como "todos los planetas giran sobre su eje" (Melo, 2001).

ACTIVIDADES

1. Identificar cuál(es) de las siguientes proposiciones son validas (calificables), explicar por qué son validas o por qué no lo son:

- El año 1200 a.C. es más reciente que el año 970 de la era Cristiana*
- El jugo de lulo tiene muy buen sabor*
- La nota máxima en un examen es 10*
- Esteban es alto*
- Ojalá que no llueva mañana*
- ¿Podría decirme, por favor, qué hora es?*
- Cuatro es mayor que 2*

2. Identificar para cuál(es) de las siguientes proposiciones es muy difícil o imposible establecer con toda certeza si son ciertas o falsas.

- Edith Piaf es la alcaldesa de París.*
- Juan Roa Sierra fue el asesino de Jorge Eliécer Gaitán el 9 de abril de 1948.*
- Marco Fidel Suárez fue presidente de Colombia.*
- Bogotá es la capital de Bolivia.*

3. Identificar las partes que componen las siguientes proposiciones (sujeto, predicado y la relación entre ambas).

- 7.0 es menor o igual que 20.5*
- El ánguloUno es mayor que 90*
- La calificación de Juan Felipe en Historia es menor que 5.0*
- Cali y Medellín son ciudades colombianas*
- 4 y 8 son números menores que 10*
- La capital de Colombia es Bogotá*

4. Expresar las siguientes proposiciones en un formato que pueda entender un computador.

- 7.0 es menor o igual que 20.5*
- El ánguloUno es mayor que 90*
- El jugo de lulo tiene muy buen sabor*
- La calificación de Juan Felipe en Historia es menor que 5.0*
- Esteban es alto*
- El valor de una calificación no puede ser mayor que 10*

- El valor de una calificación no puede exceder a 10*
- La capital de Colombia es Bogotá*
- 4 y 8 son números menores que 10*

EJEMPLO

Supongamos que Mónica quiere ir a comer helado y su padre le propone: "Como hoy entregan tus calificaciones del segundo período, si haz obtenido en matemáticas más de 8.0, vamos a comer helado el próximo sábado, de lo contrario no vamos". La situación "comer helado" está sujeta a la condición "obtener más de 8.0 en matemáticas para el segundo período".

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Es un problema sencillo de selección doble.

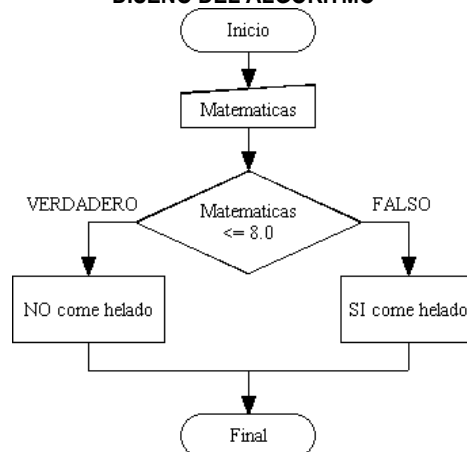
Resultados esperados: Un aviso que indique si el estudiante puede ir a comer helado el próximo sábado o no.

Datos disponibles: La calificación de matemáticas ingresada por el usuario. La regla dice: para ir a comer helado, la nota debe ser mayor que 8.0.

Restricciones: Aplicar la regla dada.

Procesos necesarios: Solicitar al usuario que ingrese la calificación de matemáticas. Evaluar si la calificación es igual o inferior a 8.0; en caso de ser verdadero, reportar "NO come helado"; en caso contrario, reportar "SI come helado".

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

para helado

local "matemáticas"

pregunta [Ingrese la calificación de Matemáticas]

da "matemáticas respuesta"

siotro (o :matemáticas < 8.0 :matemáticas = 8.0)

[
anuncia [NO come helado]

]
[
anuncia [SI come helado]

]

fin

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



En este ejemplo, la proposición se puede expresar de dos formas equivalentes:

$(\text{matemáticas} > 8.0)$

$(\text{matemáticas} \leq 8.0)$

La primera forma es más fácil de manipular por los estudiantes, ya que si la proposición es verdadera entonces “Si come helado” y si la proposición es falsa entonces “NO come helado”; además, utiliza el operador relacional mayor que ($>$).

La segunda forma (utilizada en el algoritmo) es más compleja. En ella, si la proposición es verdadera entonces “NO come helado” y si la proposición es falsa entonces “SI come helado”. En esta forma se presenta un “contrasentido” que puede desorientar a los estudiantes. Además, hay que usar el operador relacional menor o igual que, el cual se traduce en MicroMundos [4] así:

$(o : \text{matemáticas} < 8.0 : \text{matemáticas} = 8.0)$

la relación de igualdad no se menciona explícitamente en el enunciado del problema.

EJEMPLO

La profesora Ángela Cristina necesita calcular la nota definitiva para cada uno de los 22 alumnos que asisten a su curso de geometría, con el fin de saber quiénes aprobaron y quiénes reprobaron (para aprobar hay que obtener una nota igual o superior a 6.5). Ella realizó a todos sus estudiantes, en el primer periodo del año lectivo, dos exámenes y asignó un trabajo de investigación. ¿Cómo puedes ayudarle?

R/.

ANÁLISIS DEL PROBLEMA

Formular el problema: Se requiere calcular un promedio de tres notas para cada uno de los 22 alumnos.

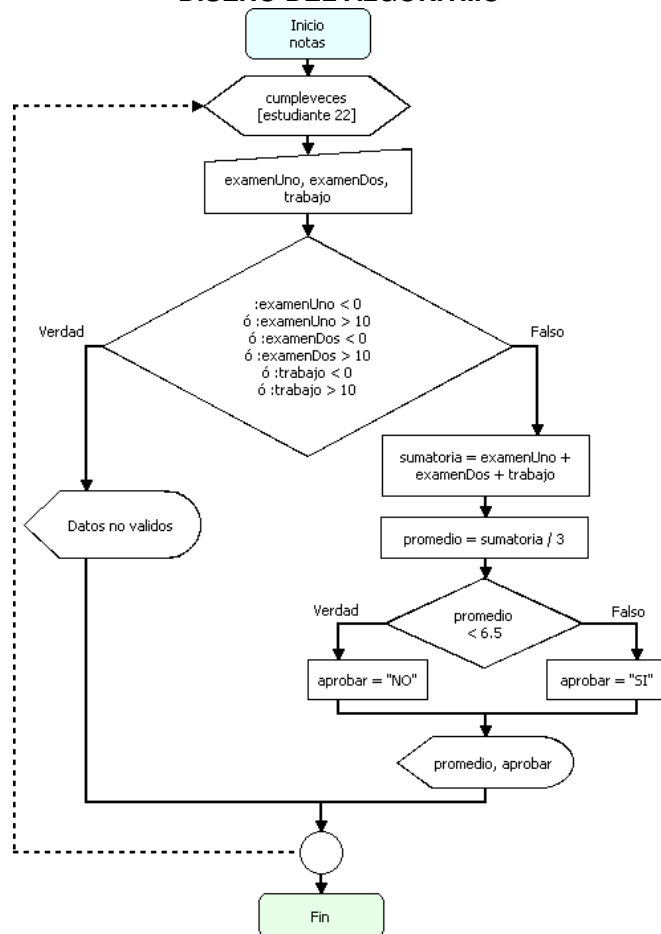
Resultados esperados: La nota definitiva de cada uno de los 22 alumnos y un aviso que indique si aprobó o no.

Datos disponibles: El número de alumnos: 22. Las notas de cada alumno las debe digitar la profesora.

Restricciones: Cada una de las tres notas tienen el mismo porcentaje en la nota definitiva. Tres notas por alumno y 22 alumnos. Todas las notas deben ser mayores o iguales a 1 y menores o iguales a 10. Para aprobar hay que tener un promedio igual o superior a 6.5.

Procesos necesarios: Para cada uno de los 22 alumnos: Leer las tres notas, verificar que estén en el rango permitido (entre 1 y 10), sumárlas, calcular el promedio, verificar si aprobó o no. Mostrar el promedio y un aviso que informe si aprobó o no.

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para notasDos
  cumpleveces [estudiante 22]
  [
    pregunta [Ingrese la nota del primer examen]
    da "examenUno respuesta
    pregunta [Ingrese la nota del segundo examen]
    da "examenDos respuesta
    pregunta [Ingrese la nota del trabajo de investigación]
    da "trabajo respuesta
    siotro (o :examenUno < 1 :examenUno > 10 :examenDos < 1
      :examenDos > 10 :trabajo < 1 :trabajo > 10)
    [
      anuncia [Datos no validos]
    ]
    [
      da "sumatoria :examenUno + :examenDos + :trabajo
      da "promedio :sumatoria / 3
      siotro (:promedio < 6.5)
      [
        da "aprobar [ -> NO aprobó el primer periodo de
        Geometría ]
      ]
      [
        da "aprobar [ -> SI aprobó el primer periodo de
        Geometría ]
      ]
    ]
  ]

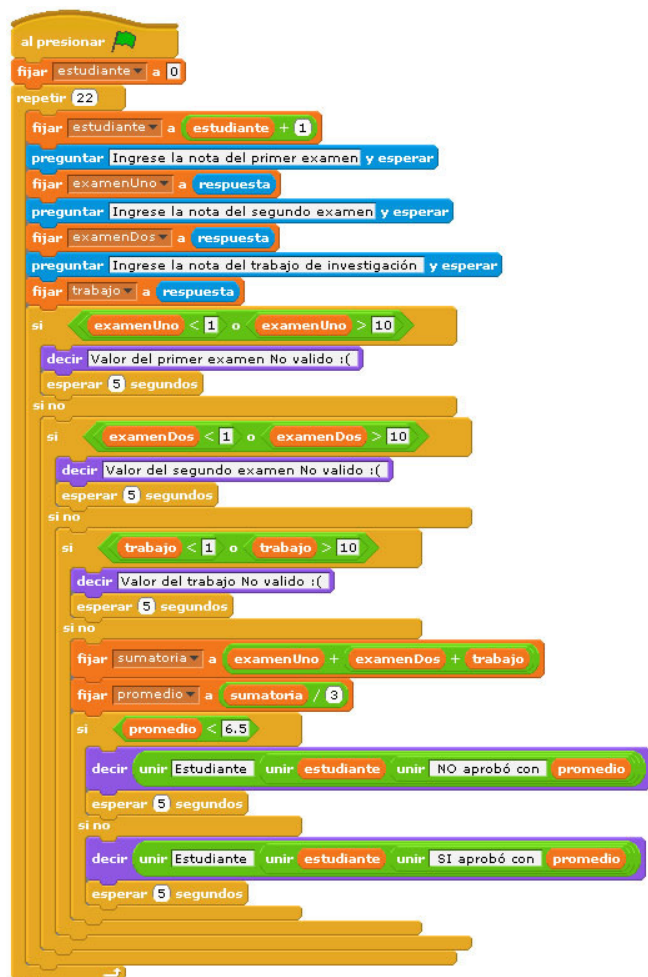
```

```

[es]
    muestra (frase [El promedio del estudiante ]:estudiante + 1
    :promedio :aprobar)
]
fin

```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH

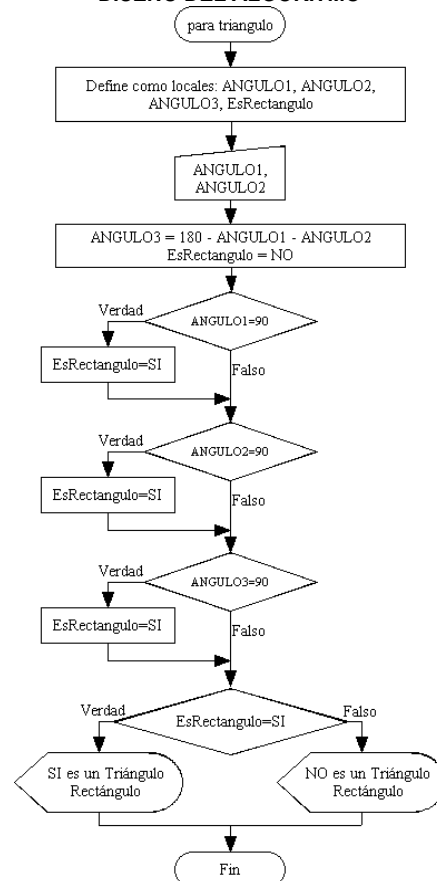


Nótese que en este ejemplo se evalúa si la proposición compuesta es verdadera entonces los datos no son validos. Como se utiliza el operador lógico "o", basta con que una de las proposiciones sea verdadera para que toda la proposición compuesta también lo sea. Adicionalmente, en la traducción a Scratch se utilizan estructuras condicionales anidadas (ver la sustentación educativa del uso de estructuras anidadas al final de esta sección).

EJEMPLO

Escribir un procedimiento para leer los valores de dos de los tres ángulos internos de un triángulo y mostrar en pantalla "Es un Triángulo Rectángulo" si efectivamente es un triángulo de este tipo o, en caso contrario, mostrar "No es un Triángulo Rectángulo".

DISEÑO DEL ALGORITMO



TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para triángulo
  local "ANGULO1
  local "ANGULO2
  local "ANGULO3
  local "EsRectangulo
  pregunta [Ingrese el 1er Ángulo del Triángulo]
  da "ANGULO1 respuesta
  pregunta [Ingrese el 2do Ángulo del Triángulo]
  da "ANGULO2 respuesta

```

```

  da "ANGULO3 180 - :ANGULO1 - :ANGULO2
  da "EsRectangulo "NO ;inicializa la variable TIPO en NO

```

;si uno de los ángulos es igual a 90 cambia el valor de TIPO a SI

```

  si :ANGULO1 = 90 [da "EsRectangulo "SI]
  si :ANGULO2 = 90 [da "EsRectangulo "SI]
  si :ANGULO3 = 90 [da "EsRectangulo "SI]

```

;dependiendo del valor de EsRectangulo, muestra que tipo de triángulo es

```

  siotro :EsRectangulo = "SI
  [
    anuncia [SI es un Triángulo Rectángulo]
  ]
  [
    anuncia [NO es un Triángulo Rectángulo]
  ]

```

fin

TRADUCCIÓN DEL ALGORITMO EN SCRATCH

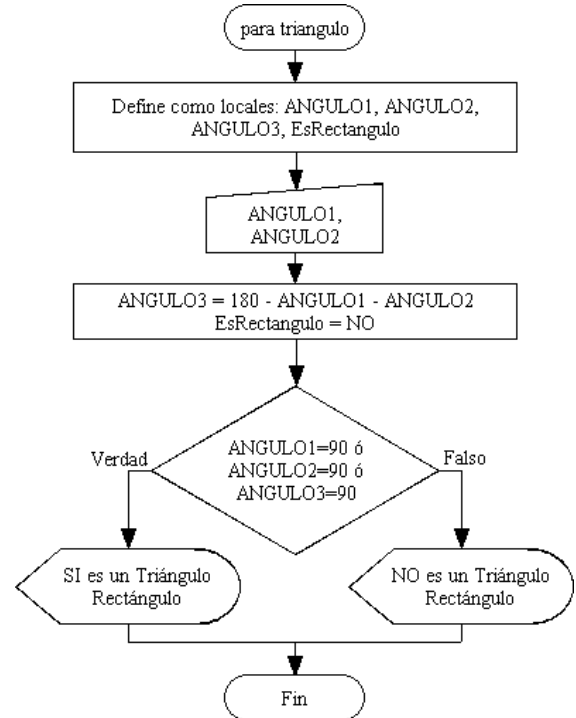


En este ejemplo se evalúa una a una las proposiciones para determinar si uno de los ángulos es igual a 90. Nótese que la variable esRectángulo se inicializa con el valor "NO", en caso de que cualquiera de los ángulos sea igual a 90, entonces la variable esRectángulo se cambia a "SI". Finalmente se evalúa el valor resultante de esta variable para mostrar el mensaje "si" o "no" es un triángulo rectángulo.

EJEMPLO

Escribir un procedimiento para leer los valores de dos de los tres ángulos internos de un triángulo y mostrar en pantalla "Es un Triángulo Rectángulo" si efectivamente es un triángulo de este tipo o, en caso contrario, mostrar "No es un Triángulo Rectángulo". Utilizar operadores lógicos en la solución.

DISEÑO DEL ALGORITMO

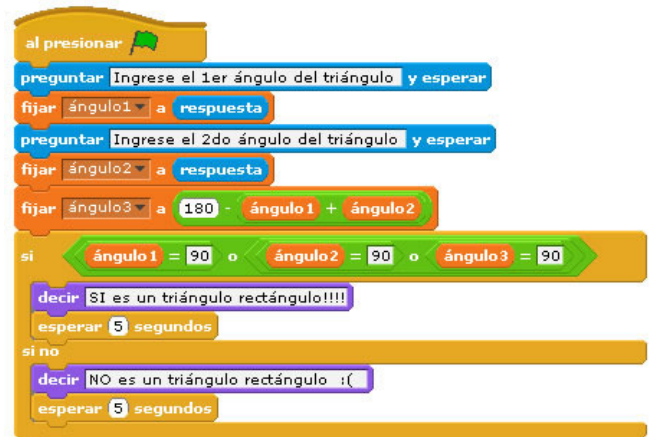


TRADUCCIÓN DEL ALGORITMO EN MICROMUNDOS

```

para triángulo
  local "ANGULO1"
  local "ANGULO2"
  local "ANGULO3"
  pregunta [Ingrese el 1er Ángulo del Triángulo]
  da "ANGULO1" respuesta
  pregunta [Ingrese el 2do Ángulo del Triángulo]
  da "ANGULO2" respuesta
  da "ANGULO3" 180 - :ANGULO1 - :ANGULO2
  siotro (o :ANGULO1 = 90 :ANGULO2 = 90 :ANGULO3 = 90)
    [
      anuncia [SI es un Triángulo Rectángulo]
    ]
    [
      anuncia [NO es un Triángulo Rectángulo]
    ]
  fin
  
```

TRADUCCIÓN DEL ALGORITMO EN SCRATCH



Esta es una solución más simple, corta y elegante que la hallada en el ejemplo anterior. La utilización del operador lógico “o” permite evaluar en una sola instrucción si alguno de los ángulos vale 90. Un problema inesperado a plantear a los estudiantes consiste en preguntarles ¿qué pasa si alguien digita valores para los ángulos que sumados den más de 180? ¿cómo controlar que esto no suceda?

TIP

En un programa se pueden incluir varias cláusulas condicionales anidadas. Los comandos si y siotro se pueden anidar de igual manera como se anida la función si (if) en la Hoja de Cálculo.

ACTIVIDAD

Tomando como base el ejemplo anterior, realizar las modificaciones necesarias para que adicionalmente, el programa muestre en pantalla “No es un Triángulo” en caso de que cualquiera de los ángulos sea menor o igual a cero.

Estructuras condicionales anidadas

Hay situaciones que requieren el uso de estructuras condicionales anidadas. En estas, el resultado de la primera proposición implica evaluar a continuación una segunda proposición y esta a su vez requiere que se evalúe una tercera proposición, y así sucesivamente, hasta agotar todas las condiciones.

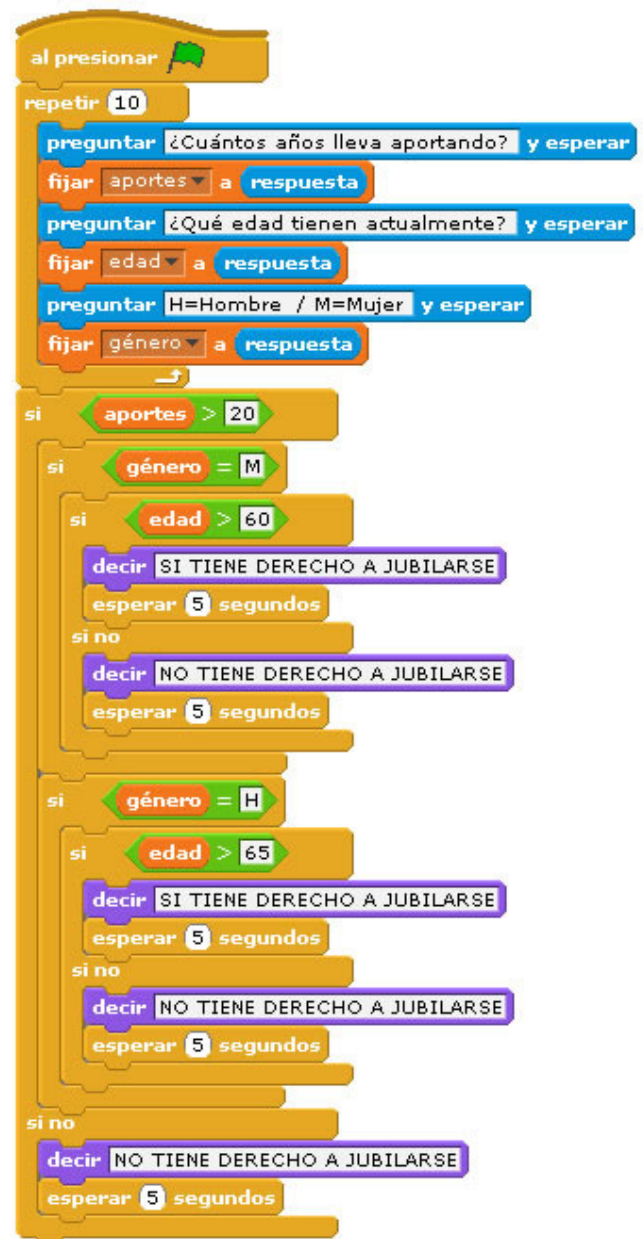
Plantear estructuras algorítmicas anidables (iterativa y condicional) requiere procesos de pensamiento asociados con el sistema operatorio de clasificación o inclusión. Este sistema se empieza a adquirir cuando los niños trabajan con inclusiones de clases tales como: $\text{gatos}(A) < \text{felinos}(B) < \text{animales}(C) < \text{seres vivos}(D)$. Pero el sistema de clasificación está basado en cinco operaciones: Composición ($A+A'=B$, $B+B'=C$, $B-A'=A$, $C-B'=B$); Inversión ($-A-A'=-B$); Identidad ($A-A=0$); Tautología ($A+A=A$, donde $A+B=B$); Asociatividad ($A+(A'+B')=(A+A')+B'$).

Para poder determinar rápida y efectivamente cuándo la solución de un problema requiere estructuras anidadas, se deben manejar sistemas de clasificación o inclusión de clases.

El siguiente caso ilustra muy bien este punto: "Se requiere elaborar un procedimiento que permita determinar para un grupo de 10 personas si tienen derecho o no a jubilarse a partir de los datos género, edad y años de aportes; y las siguientes condiciones: si es hombre debe tener más de 65 años de edad y más de 60 años si es mujer, pero en todo caso se deben

haber realizado aportes por más de 20 años".

Para resolver este problema se puede plantear la siguiente inclusión de clases: $\text{edad}(A) < \text{género}(B) < \text{aportes}(C)$.



UNIDAD 4: DEPURACIÓN

CUANDO SE PRESENTAN PROBLEMAS

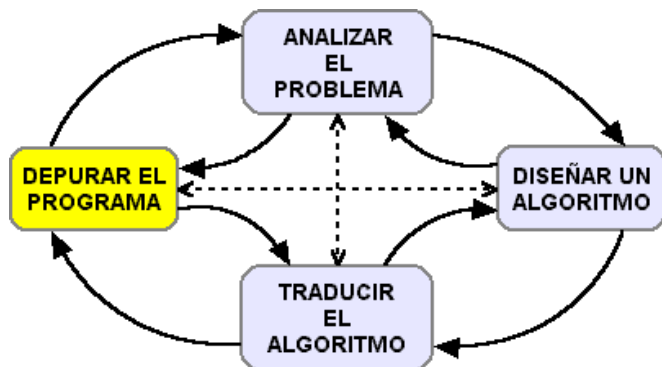


Ilustración 4-1: Cuarta fase del ciclo de programación.

Es muy difícil elaborar procedimientos perfectos en un primer intento y la dificultad aumenta a medida que los problemas se vuelven más complejos. Después de traducir el algoritmo en un lenguaje de programación, el procedimiento resultante debe ser probado y los resultados validados (revisión). A este proceso se le conoce como depuración y contribuye a mejorar en los estudiantes la capacidad para resolver problemas puesto que la depuración basada en la retroalimentación es una habilidad útil para toda la vida (Stager, 2003).

La depuración de un procedimiento hace parte fundamental del ciclo de programación y desde el punto de vista educativo estimula en los estudiantes la curiosidad, la perspectiva, la comunicación y promueve valores como responsabilidad, fortaleza, laboriosidad, paciencia y perseverancia. La programación facilita un diálogo interior en el cual la retroalimentación constante y el éxito gradual empujan a los alumnos a ir más allá de sus expectativas (Stager, 2003).

Otras dos actividades relacionadas con esta etapa, que no se tratarán en esta guía, son la afinación y la documentación. La primera consiste en realizar retoques para lograr una mejor apariencia del programa (en pantalla o en los resultados impresos) o para ofrecer funcionalidades más allá de los resultados esperados, especificados en la fase de análisis del problema. La segunda tiene un carácter eminentemente comunicativo, con la documentación de un programa se pone a prueba la capacidad del estudiante para informar a otras personas qué hace su programa, cómo lo hace y el significado de cada elemento utilizado. Esta actividad se puede llevar a cabo mediante comentarios introducidos al código o por medio de documentación formal en un documento que se anexa al procedimiento elaborado.

Depuración

La corrección de fallas es una de las situaciones que mayor frecuencia tienen en el mundo profesional. Con esta actividad se intenta identificar fallas sintácticas o lógicas en programas que no funcionan adecuadamente; una vez aislada la falla, esta se soluciona y se vuelve a probar el programa y a validar los resultados. Según Jonassen (2003), para corregir fallas efectiva y eficientemente se requiere conocimiento del sistema (comprensión conceptual de cómo funciona el sistema), conocimiento procedimental (cómo llevar a cabo tanto procedimientos de solución de fallas, como actividades de prueba) y conocimiento estratégico (saber cuándo, dónde y por qué aplicar procedimientos de solución de fallas y actividades de prueba).

Fallas de sintaxis

Este tipo de fallas solo se presenta en MicroMundos ya que el entorno de programación de Scratch es gráfico y los estudiantes no deben escribir el código. Además, los bloques con las instrucciones son autoencajables, por tanto no es posible ubicar un bloque de manera que se generen fallas de sintaxis.

Las fallas sintácticas son las más sencillas de identificar ya que el entorno de programación indica dónde se ha producido el error y de qué tipo es. Por ejemplo, MicroMundos reportará el error “valorDos no tiene valor en prueba” cuando, en el procedimiento “prueba”, se intenta mostrar el contenido de la variable “valorDos” sin haberla asignado antes.

En caso de presentarse una falla de sintaxis, el estudiante debe:

- Comprender el mensaje de error que reporta el ambiente de programación (apoyarse en las opciones de ayuda que ofrece MicroMundos o en el docente).
- Examinar el código del programa para identificar en cuál instrucción se encuentra la falla
- Corregir la falla
- Probar el programa de nuevo

EJEMPLO 4-1

Retomemos el ejemplo del estudiante que aprueba un examen cuando obtiene una calificación mayor o igual a seis (ejemplo 3-14). Se requiere elaborar un procedimiento que pida al usuario una calificación, aplique el criterio de aprobación e imprima “Aprobado” o “Reprobado”, según sea el caso.

R/.

ANÁLISIS DEL PROBLEMA

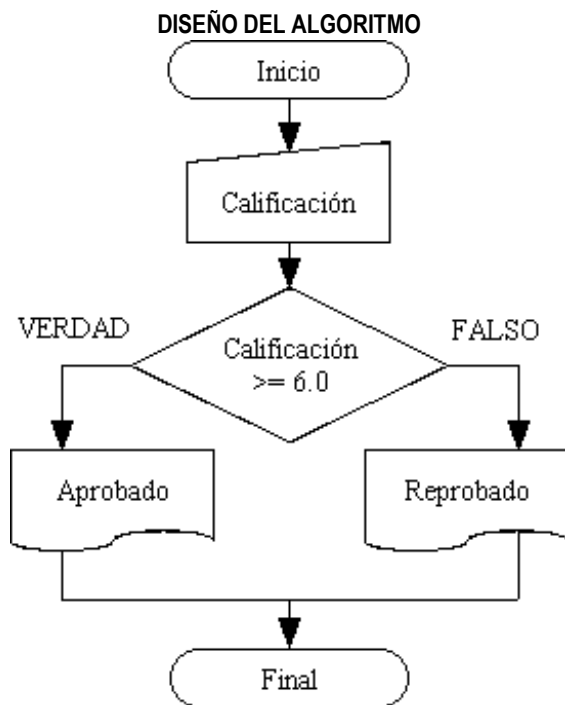
Formular el problema: Es un problema sencillo de selección doble.

Resultados esperados: Un aviso que reporte si el estudiante "Aprobó" o "Reprobó" el examen.

Datos disponibles: La calificación ingresada por el usuario. Para aprobar, la nota debe ser mayor o igual a 6.0.

Restricciones: Aplicar el criterio de aprobación.

Procesos necesarios: Solicitar al usuario que ingrese la calificación. Evaluar si la calificación es igual o superior a 6.0; en caso de ser verdadero, reportar "Aprobado"; en caso contrario, reportar "Reprobado".



TRADUCCIÓN DEL ALGORITMO

```
para aprueba1
  local "calificación"
  preguntas [Ingrese la Calificación]
  da "calificación respuesta"
  siotro (o :calificación > 6.0 :calificación = 6.0)
  [
    anuncia [Aprobado]
  ]
  [
    anuncia [Reprobado]
  ]
fin
```

Obsérvese que MicroMundos arroja un mensaje de error cuando se ejecuta el procedimiento: "No sé cómo hacer preguntas en aprueba1". Este error se produce porque el comando **pregunta** está mal escrito, en su lugar se escribió **preguntas**. El problema se soluciona al escribir correctamente el comando indicado.

Fallas de lógica

Este tipo de falla se presenta tanto en MicroMundos como en Scratch. Para identificar fallas de tipo lógico en un procedimiento que no se interrumpe en ningún momento y que arroja unos resultados, estos se deben

revisar cuidadosamente. Uno de los procedimientos más utilizados es el que se conoce como prueba de escritorio. Esta consiste seguir paso a paso cada una de las instrucciones del procedimiento, asignando valores iniciales a variables y constantes y, realizando las operaciones indicadas en cada instrucción hasta llegar al final del procedimiento. Luego, comparar los resultados que produce la prueba de escritorio con los resultados que arroja el procedimiento; ambos conjuntos de resultados deben ser iguales. Esta metodología es viable cuando los procedimientos son sencillos, con pocas instrucciones.

En caso de presentarse una falla de lógica, el estudiante debe:

- Examinar el diagrama de flujo para detectar instrucciones que faltan, sobran o que se encuentran en la posición incorrecta
- Asegurarse que las instrucciones representan rigurosamente el diagrama de flujo
- Utilizar las opciones de MicroMundos y Scratch para ver la ejecución del programa en forma lenta.

EJEMPLO 4-2

Continuamos con el mismo ejemplo: Elaborar un procedimiento que pida al usuario una calificación, aplique el criterio de aprobación e imprima "Aprobado" o "Reprobado", según sea el caso. Con el fin de ejemplificar una falla de lógica, se debe digitar en el área de procedimientos de MMP el siguiente código que representa el algoritmo del ejemplo 4-1.

TRADUCCIÓN DEL ALGORITMO

```
para aprueba2
  local "calificación"
  pregunta [Ingrese la Calificación]
  da "calificación respuesta"
  siotro (o :calificación < 6.0 :calificación = 6.0)
  [
    anuncia [Aprobado]
  ]
  [
    anuncia [Reprobado]
  ]
fin
```

Obsérvese que el procedimiento "aprueba2" se ejecuta correctamente; no aparece ningún error de sintaxis. Pero no funciona bien desde el punto de vista lógico. Si lo probamos con el valor 5, el procedimiento nos reporta que el estudiante aprobó la materia, cuando esto no es correcto. Si lo probamos con 7, nos reporta que el estudiante reprobó, cuando tampoco es exacto. En cambio, cuando lo probamos con 6 nos dice que el estudiante aprobó y este dato si es correcto. El problema radica en que en el comando **siotro** se digitó erróneamente "<" en lugar de ">", tal como aparece en el diagrama de flujo.

Adicionalmente, en este ejemplo se puede observar la sintaxis en MicroMundos de los operadores lógicos (y, o,

no), mediante los cuales se unen proposiciones sencillas para construir proposiciones compuestas. Estos deben ir en seguida del paréntesis que abre la proposición:

siotro (o :calificación > 6.0 :calificación = 6.0)

La proposición se lee así:









“calificación mayor que 6.0 o calificación igual a 6.0”.

En la detección y eliminación de fallas en un procedimiento, cuenta mucho la cantidad de fallas similares que el estudiante ha tenido oportunidad de resolver. La “experiencia” es un factor crucial; incluso, las fallas que se recuerdan con mayor precisión son aquellas cuya solución presentó mayor dificultad (Jonassen, 2003). Con la depuración, el estudiante realiza un conjunto de actividades que contribuyen a aprender de ese problema, a reinterpretarlo, a formular otros nuevos y a incrementar la comprensión de la solución hallada. Esta incita a los estudiantes a preguntarse: “¿puedo obtener el mismo resultado de una forma diferente?” y “¿puedo utilizar los métodos empleados para solucionar este problema en la solución de otros que se me han presentado antes?”

ANEXO 1

RESUMEN DE COMANDOS DE MICROMUNDOS Y SCRATCH

A continuación se ofrece un resumen de las primitivas de MicroMundos y Scratch utilizadas en esta Guía.

DESCRIPCIÓN	MICROMUNDOS	SCRATCH
ADELANTE Mueve la tortuga hacia adelante. Los valores mínimos y máximos para adelante son -9999 y 9999, respectivamente.	Adelante (ad) número Ver at, de, iz. <i>adelante control1</i> <i>repite 4 [ad 50 de 90]</i>	
ALTO Detiene el procedimiento que está activo. Alto solo puede usarse dentro de un procedimiento.	Alto Ver deténtodo, deténme y reporta. <i>para contar :número</i> <i>si :número > 100 [alto]</i> <i>muestra :número</i> <i>contar :número + 5</i> <i>fin</i>	 
ANUNCIA Muestra el mensaje en una caja de alerta. Haciendo clic en Aceptar se cierra la caja. Si mueve la caja de alerta a una nueva posición mientras está en la pantalla, esta será la posición en que la próxima caja de alerta aparecerá en el proyecto.	anuncia palabra-o-lista <i>anuncia "bienvenido</i> <i>anuncia [Hola]</i> <i>anuncia texto1</i> Ver pregunta y respuesta.	 
ATRÁS Mueve la tortuga hacia atrás. Los valores mínimos y máximos para atrás son -9999 y 9999, respectivamente.	atrás (at) número Ver ad, de, iz <i>atrás 45</i> <i>at control1</i> <i>repite 4 [at 50 iz 90]</i>	
AZAR Devuelve un número entero positivo (incluyendo el 0) menor que número. El número máximo es 9999.	azar número <i>azar control1</i> <i>azar 2</i> <i>Repite 26 [ad azar 30 de azar 60]</i>	
BNOMBRES Borra de la memoria todas las variables globales. MicroMundos no borra las variables cuando se abre o se crea un nuevo proyecto. Por lo tanto se recomienda usar bnombres cada vez que se inicie un nuevo proyecto.	bnombres Ver nombres. <i>bnombres</i>	No aplica
CON PLUMA Pone la pluma a la tortuga en uso. La tortuga dejará una marca cuando se mueva, pero no cuando sea arrastrada.	Cp Ver sp. <i>repite 6 [sp ad 10 cp ad 10]</i>	
CUMPLEVECES Activa la lista de instrucción para cada uno de los valores especificados en la serie. La primera entrada es una lista con un nombre de variable temporal y un número máximo. La segunda entrada es una lista de instrucciones que usa la variable incluida en la primera lista.	cumpleveces serie lista-de-instrucción Ver cumplelista. <i>cumpleveces [i 8][muestra :i]</i> <i>cumpleveces [i 360] [fcolor :i / 10 ad 40 at 40 de 2]</i>	No aplica
DA (ASIGNA) Crea una variable y le asigna el valor palabra-o-lista. Estas variables mantienen su valor siempre y cuando no se las borre o se cierre MicroMundos.	da vpalabra palabra-o-lista Ver nombra, cosa, bnombre, nombres y crearproyecto. <i>da "equipo [t1 t2 t3]</i> <i>da "texto texto1</i>	