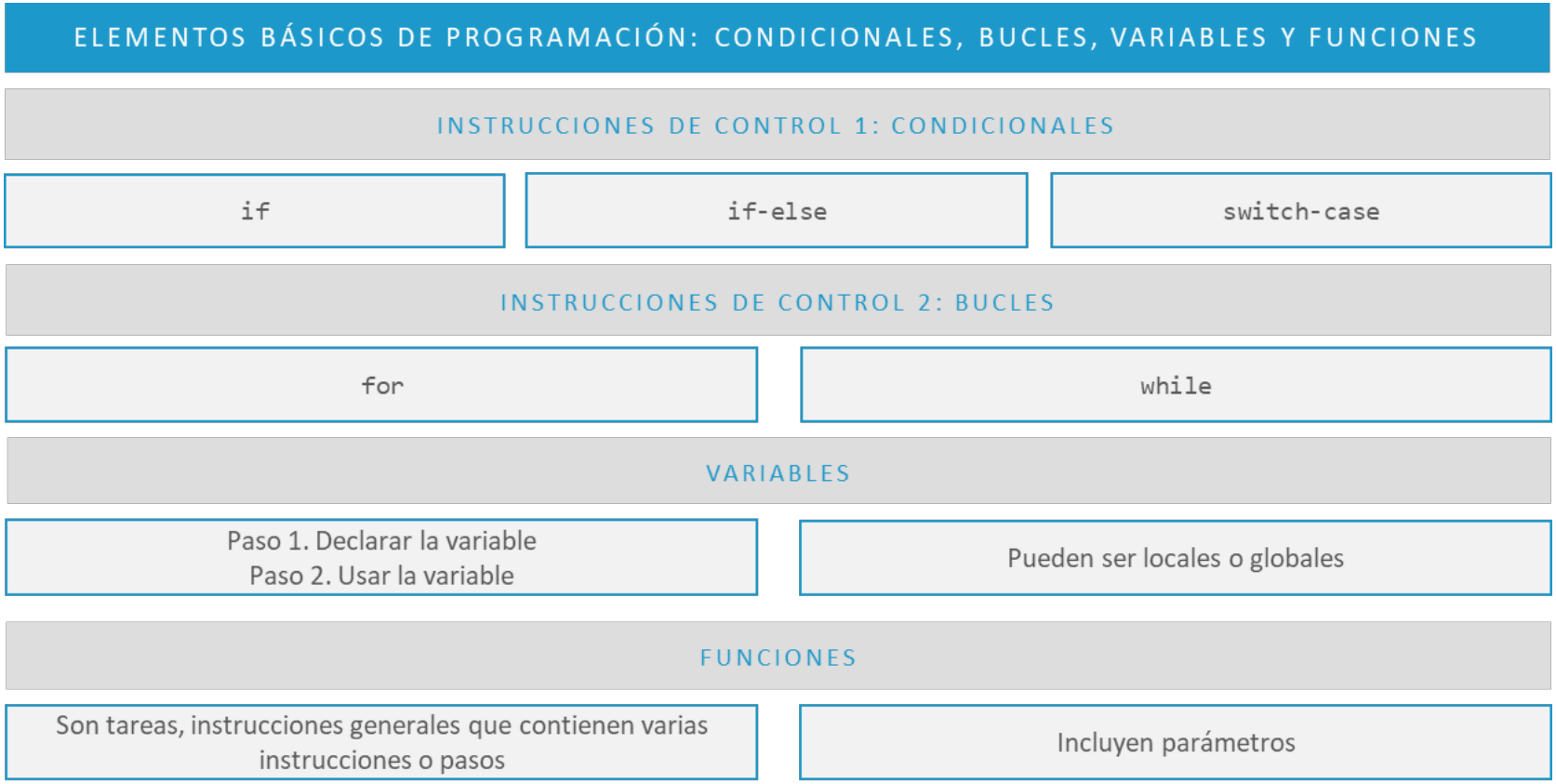
Programación y Robótica para Docentes

Condicionales, bucles, variables y funciones

Índice

Esquema



Ideas clave

3.1. Introducción y objetivos

En este tema vamos a conocer los elementos básicos que contiene cualquier programación en la mayoría de los lenguajes de programación. Partimos del conocimiento de los algoritmos para ver qué elementos los componen.

Además, abordaremos los siguientes **objetivos**:

- Conocer el concepto de *instrucciones de control* y para qué se utilizan en la programación.
- Saber diferenciar entre las distintas instrucciones de control: **condicionales** y **bucles**.
- Ser capaces de elegir la instrucción de control más adecuada para cada caso o problema.
- Comprender qué es una **variable** y cuándo se debe utilizar en programación.
- Entender la importancia de las **funciones** y ser capaces de crear una cuando sea conveniente.

Enseñar todos estos elementos de programación a nuestros alumnos es fundamental para que comprendan la lógica de programación general y sean capaces de aplicar cualquiera de ellos en cualquier lenguaje de programación que aprendan a utilizar.

Para trabajar todos estos elementos de manera lúdica, se recomienda acceder al juego *Arcadeland*, y utilizar el manual *Programación.* como complemento a cada capítulo de esta aventura gráfica. Puedes encontrar ambos recursos en la sección **A fondo**.

3.2. Condicionales

Los condicionales, o las sentencias condicionales, son un tipo de instrucciones de control. Junto con los bucles, nos sirven para controlar nuestro programa sin necesidad de intervenir, es decir, **gracias a las instrucciones de control, nuestro programa podrá actuar de una forma más o menos autónoma**. En lo que respecta a los condicionales, el programa actuará en función de qué condición se cumpla en cada caso determinado.

Existen varios tipos de condicionales, que dependerán del número de condiciones que necesitemos cubrir. Los condicionales más utilizados son `if`, `if-else` y `switch-case`.

`if`

La sentencia condicional `if` ('si') es una instrucción que se ejecuta o no en función del valor de una condición. Dependiendo de si la condición se cumple o no, se ejecutará una acción u otra. Por ejemplo:

> **Si empieza a llover, abriré el paraguas**
>
> En este caso, la condición es la lluvia, condición que puede darse en un momento determinado. Si ocurre la condición de llover, ejecutaré la acción de abrir el paraguas. Pero **si no se da** la condición de **llover**, **no ejecutaré** la acción de **abrir el paraguas**.

> **Si se hace de noche, encenderé la linterna**
>
> En este caso, la condición es que se haga de noche. Si ocurre dicha condición, ejecutaré la acción de encender la linterna. Sin embargo, **si no se hace de noche**, **no encenderé la linterna**.

`if-else`

En muchos casos, la sentencia `if` se nos queda corta. Para los casos en los que queremos hacer una acción diferente si no se cumple la condición planteada, tendremos que utilizar `if-else`.

Traducida literalmente del inglés, se la podría llamar la sentencia 'si-de lo contrario', es decir, «si se cumple la condición, haz esto; de lo contrario, haz esto otro». Por ejemplo:

> **Si nos encontramos ante una puerta, la puerta tendrá dos estados: abierta o cerrada**
>
> En caso de que esté **abierta**, haremos una cosa **(pasar)**, y, de lo contrario, en el caso de que esté **cerrada**, haremos otra **(llamar)**. Imaginemos que programamos un robot para esa situación; tendremos que plantear dos acciones: una para si se cumple la condición de que la puerta esté abierta, y otra para si no se cumple la condición de que la puerta esté abierta.

Se podría pensar que los ejemplos del apartado anterior, correspondientes al condicional simple `if`, también se podrían hacer con un `if-else`. Efectivamente. Sin embargo, vamos a ver con más profundidad cada ejemplo:

> **Si empieza a llover, abriré el paraguas**
>
> Aquí partimos de que no está lloviendo, y solo si empieza a llover ejecutaré la condición escrita: abrir el paraguas. En este caso, el «de lo contrario» sería seguir sin abrirlo, que es lo mismo que el estado inicial, por lo que no hace falta incluir esa segunda condición. Pero ¡cuidado! si no incluimos la condición de cerrar el paraguas, seguiría abierto, aunque dejase de llover.

Si se hace de noche, encenderé la linterna

Este caso es muy parecido al anterior; solo si se hace de noche ejecutaré la condición: encender la linterna. El «de lo contrario» no nos haría falta, ya que seguiríamos en el estado inicial: no encenderla. Igual que en el ejemplo anterior, si no incluimos la condición contraria, seguirá encendida, aunque salga el sol.

Figura 1. Ejemplos de condicional `if` y condicional `if-else`.

`switch-case`

En otras ocasiones, una condición puede tomar más de dos valores. Para estos casos, existe otra sentencia, conocida como `switch-case`. Por ejemplo, una persona hace diferentes actividades según qué día de la semana sea. En este caso, la condición es el día de la semana y puede tener siete valores diferentes: lunes, martes, miércoles, jueves, viernes, sábado y domingo.

Si es… (día de la semana), haré… (actividad)

1. Lunes: fútbol.
2. Martes: inglés.
3. Miércoles: fútbol.
4. Jueves: inglés.
5. Viernes: hípica.
6. Sábado: tenis.
7. Domingo: golf.

Otro ejemplo podría ser qué ropa nos ponemos según el tiempo que haga. Imaginemos que diseñamos un armario inteligente, que nos recomienda ropa según el tiempo que se espera para ese día. En este caso, tendríamos que programar diferentes condiciones: calor extremo, calor moderado, frío moderado, frío extremo, lluvia con viento, lluvia sin viento, nieve… El armario nos va a recomendar unas prendas determinadas en función del tiempo que haga, es decir, de la condición dada.

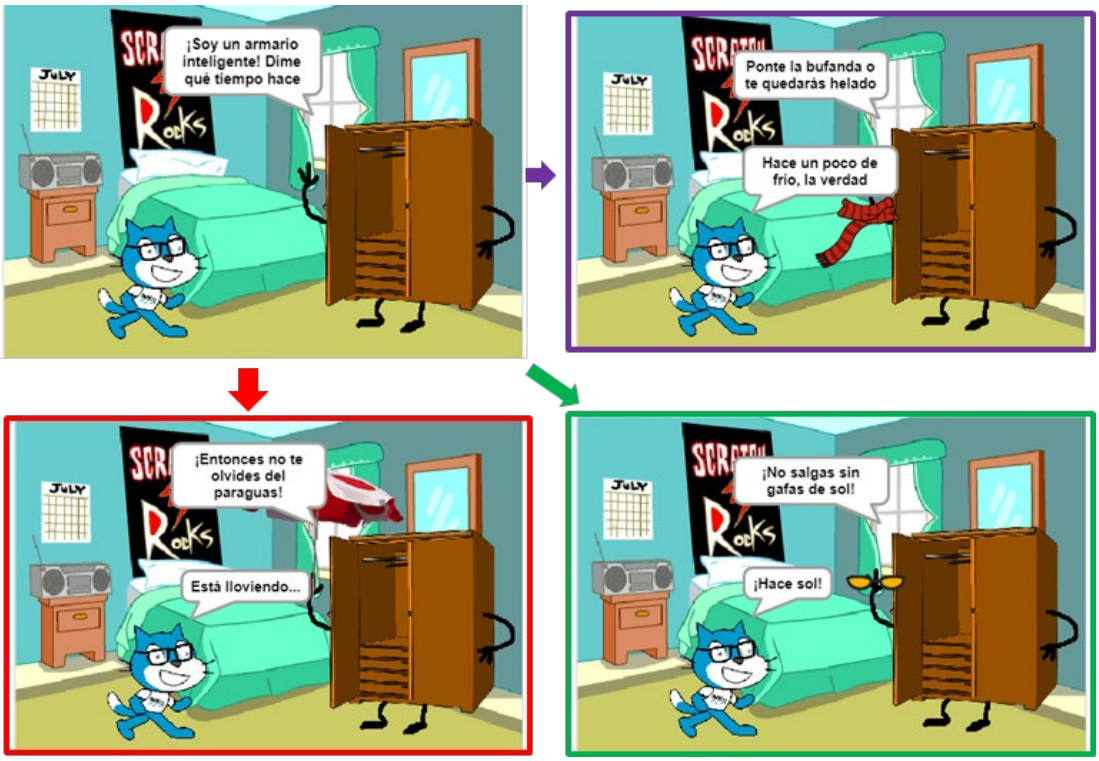


Figura 2. Ejemplo de `switch-case`.

Ejemplos aplicados a Sphero Mini

Sphero Mini tiene un sensor para detectar los cambios de velocidad, llamado **acelerómetro**. Con los datos que nos proporciona este sensor, podemos crear diferentes condicionales.

Condicional `if`

Si el acelerómetro mide más de 300 (puede alcanzar cualquier valor entre 0 y 99 999), le diremos a Sphero que encienda su LED en color azul.

Condicional `if-else`

Si el acelerómetro mide más de 300, le diremos a Sphero que encienda su LED en color azul, de lo contrario (si el acelerómetro mide menos de 300), le diremos que encienda su LED en color rojo.

Condicional `switch-case`

- Si el acelerómetro se mueve muy rápido (más de 50 000), le diremos que encienda el LED en color verde.
- Si se mueve a velocidad media (entre 300 y 50 000), le diremos que encienda el LED en color azul.
- Si no se mueve o se mueve muy lento (menos de 300), le diremos que encienda el LED en color rojo.

Sphero Mini no ofrece la posibilidad de hacer este último tipo de condicional, pero lo podemos hacer de la siguiente manera:

Dos condicionales `if-else`

- Más de 50 000: verde. De lo contrario (menos de 50 000): azul.
- Más de 300: azul. De lo contrario (menos de 300): rojo.

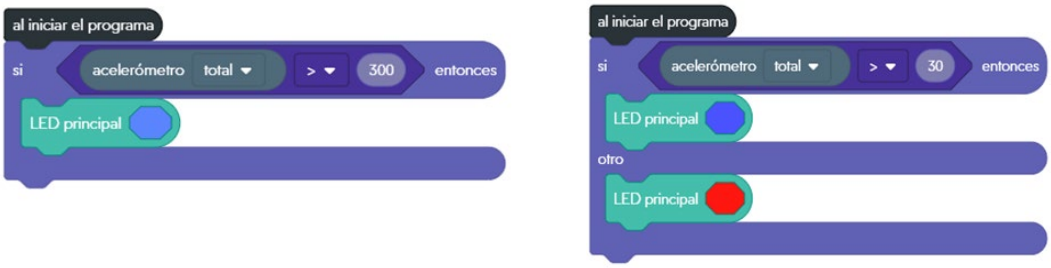


Figura 3. Programación de condicional `if` (izquierda) y programación de condicional `if-else` (derecha) en Sphero Mini.

3.3. Bucles

Los bucles, al igual que los condicionales, son instrucciones de control. Los bucles se utilizan cuando queremos repetir una acción o instrucción. Existen varios tipos de bucles, pero los dos más utilizados son el bucle `for` y el bucle `while`.

`for`

El bucle `for` ('contar') se utiliza para repetir una o más instrucciones un determinado número de veces. Por tanto, utilizamos `for` solo cuando conocemos el número de veces que queremos que se ejecute una acción. Por ejemplo, si invito a 10 amigos a merendar y quiero preparar unos bocadillos, tendría que hacer lo siguiente:

1. Preparar bocadillo.
2. Preparar bocadillo.
3. Preparar bocadillo.
4. Preparar bocadillo.
5. Preparar bocadillo.
6. Preparar bocadillo.
7. Preparar bocadillo.
8. Preparar bocadillo.
9. Preparar bocadillo.
10. Preparar bocadillo.

Para expresarlo de forma eficiente, en programación, se utilizaría el bucle `for`:

> **Bucle** `for 10`
>
> Preparar bocadillo (repite desde el 1 al 10).

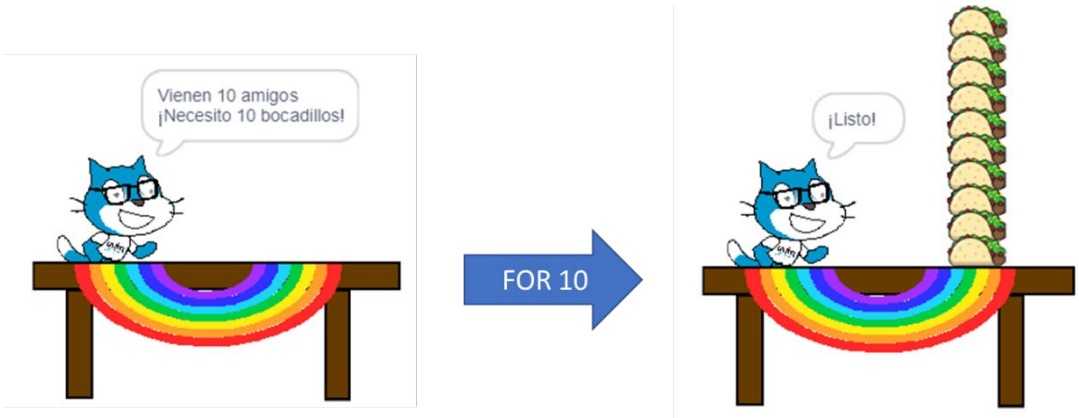


Figura 4. Ejemplo de bucle `for`.

Imaginemos que creamos un robot que tiene que subir unas escaleras. Para subir un escalón, el robot tiene que subir su pierna robótica y moverla hasta posarse en el siguiente escalón. Si la escalera tiene 32 escalones, podemos programar estos movimientos 32 veces, o podemos expresarlo con un bucle `for`:

> **Bucle** `for 32`
>
> Subir escalón (repite desde el 1 al 32).

`while`

El bucle `while` (‘mientras’) se utiliza para repetir la ejecución de unas acciones un número indefinido de veces, mientras que se cumpla una condición. Por ejemplo, mientras conduzco, miro a la carretera.

Se podría pensar que esta acción también puede realizarse mediante un condicional (si conduzco, miro a la carretera). Sin embargo, si en nuestro algoritmo tenemos órdenes tras la condición de mirar a la carretera, la acción del condicional terminaría.

Por ejemplo, si nos llama un amigo, realizaremos la acción de coger la llamada a nuestro amigo, y podríamos tener un accidente.

Esto no ocurre con el bucle `while`: mientras estemos conduciendo, solo realizaremos esas acciones y no continuaremos con nuestro programa. No cogeríamos el teléfono y, por tanto, evitaríamos la posibilidad de tener un accidente. Mucho mejor, ¿verdad?

Ejemplos aplicados a Sphero Mini

Sphero Mini también ofrece la posibilidad de realizar estos bucles. Si recordamos, el bucle `for` ejecuta una acción un número determinado de veces:

> **Bucle** `for`
>
> Repetir un sonido aleatorio 3 veces.

Por su parte, el bucle `while` ejecuta una acción mientras se cumple una condición. Con los datos del mismo sensor que hemos utilizado antes, el acelerómetro, vamos a establecer la condición para nuestro bucle `while`:

> **Bucle** `while`
>
> Repetir un sonido aleatorio hasta que el acelerómetro detecte movimiento (más de 300), es decir, repetir el sonido mientras que el acelerómetro no detecte movimiento.

Figura 5. Programación de bucle `for` (izquierda) y programación de bucle `while` (derecha) en Sphero Mini.

3.4. Variables

Una variable es un espacio de memoria del programa, como una «caja» donde guardaremos un dato, un valor que podremos ver y recuperar más adelante durante el resto del programa. Si no guardamos un dato en una variable, no podremos reutilizarlo, ya que el programa no lo recordará. Además, en cualquier momento podremos cambiar el valor guardado en la variable. Por tanto, para trabajar con variables, debemos hacer dos cosas: **declarar la variable** y **usar la variable**.

Declarar la variable

Al declarar en nuestro programa una variable, lo que hacemos es crear esa «caja» y guardar por primera vez un dato en ella. Podemos nombrar como queramos una variable, y esta cualidad nos servirá para identificar en todo momento nuestra «caja». Podemos guardar datos de muchos tipos, como de tipo número o de tipo texto. Al declarar la variable, definimos de qué tipo es. Por ejemplo, si lo primero que guardamos es un número, esa variable va a ser siempre para guardar números (¡no mezclemos tipos!).

Figura 6. Declarar la variable.

Usar la variable

Una vez que se ha definido la variable, podemos usarla siempre que queramos. Para ello, debemos nombrarla tal y como la hayamos nombrado al declararla.

Figura 7. Usar la variable.

Variables locales y variables globales

Las variables pueden ser de dos tipos, según por cuánto tiempo necesitemos reservar espacio del programa. Es importante saber en qué momento del programa usamos y declaramos las variables, ya que este se ejecuta en orden (de arriba abajo).

- Variables **locales**: solo se pueden utilizar dentro del bloque de código en el que han sido declaradas. Pero esto es una ventaja, ya que así, cuando dejan de hacer falta, se borran solas y dejan espacio en la memoria del robot para otras cosas.

- Variables **globales**: se crean al inicio del programa, lo que las hace universales dentro de dicho programa. Pueden utilizarse en cualquier momento y lugar de este.

Al ejecutar un programa en orden, necesitamos declarar las variables globales al principio del programa. También es fundamental tener en cuenta que, si declaramos una variable local, esta solo se podrá utilizar dentro del bloque de código en el que se declara. Veamos ahora unos ejemplos:

> **Variables**. Ejemplo 1
>
> Imaginemos que queremos crear una canción con un *buzzer* o zumbador, un componente que emite sonidos cuando una placa

metálica vibra. Si queremos reproducir la nota sol, el *buzzer* vibrará 392 veces; si queremos reproducir la nota mi, vibrará 329; etc. Si queremos hacer una melodía con estas notas, en lugar de escribir la frecuencia de la nota cada vez que aparezca, podemos guardar las frecuencias en variables, que llamaremos con el nombre de cada nota.

sol = 392

Cada vez que queramos usar cada nota, simplemente tendremos que llamar a la nota, y no a la frecuencia exacta.

sol,
mi,
sol,
…

Variables. Ejemplo 2

Imaginemos ahora que queremos cambiar la escala en la que suenan las notas. Si no hubiésemos creado las variables, tendríamos que localizar cuándo aparece cada frecuencia y sustituirla por la actual. Si hemos creado una variable con cada nota, solo tendremos que cambiar su valor donde se declara la variable, y todo el programa se actualizará para utilizar esos valores.

Una variable puede contener un número concreto, como hemos visto en los ejemplos anteriores, o un número que queramos ir modificando. En el segundo caso, añadiremos una instrucción para que, cuando esta se ejecute, se sume uno a la variable. Por ejemplo:

Variables. Ejemplo 3

Si creamos un contador para un partido de baloncesto, creamos la variable *Puntos* y la igualamos inicialmente a 0.

Puntos = 0

Cada vez que el balón se enceste en la canasta, la variable *Puntos* debería cambiar:

Puntos = Puntos + 1

Otro ejemplo de valor que varía es el valor que nos devuelve un sensor. En robótica, el valor que nos devuelve un sensor también lo guardamos en una variable, ya que será un número que variará en función de la magnitud detectada.

Variables. Ejemplo 4

Según la luz ambiental, un sensor de luz LDR podrá obtener un valor de 300 en un momento dado, y un valor de 800 en otro momento. El valor del sensor y, por tanto, de nuestra variable nos ayudará a ejecutar el programa si programamos en un condicional, que podría ser, por ejemplo, que si el sensor LDR detecta menos de 500, se encienda un LED rojo, y si el sensor LDR detecta lo contrario (más de 500), se encienda un LED verde.

3.5. Funciones

Cuando tengamos una tarea que se repite mucho, o cuando queramos organizar mejor nuestro proyecto, crearemos una función que realice dicha tarea cada vez que la llamemos. Por tanto, podríamos decir que **una función es una tarea, una instrucción general que contiene varias instrucciones o pasos (un algoritmo específico)** y que permite ejecutar dicha tarea con esa sola instrucción las veces que queramos.

Funciones. Ejemplo 1

Si ponemos un ejemplo relacionado con comida, diríamos que «freír un espárrago» es una función, una tarea que se compone de las instrucciones antes descritas. Esta función o tarea la podremos reutilizar en todas las recetas que lleven espárrago frito: parrillada de verduras, *risotto* de espárragos, etc.

Traducido al lenguaje que utilizamos para programar, «parrillada de verduras» y «*risotto* de espárragos» serían programas concretos. «Freír un espárrago» sería una función, una tarea que repetimos a menudo y que, si la escribimos de esta forma, nos ahorra el tener que escribir todas las veces el conjunto de instrucciones (el algoritmo «preparar la sartén», «echar el espárrago», etc.).

Funciones. Ejemplo 2

Otro ejemplo de función es «llamar al ascensor». Al solicitar esta tarea, estamos haciendo que se ejecuten una serie de instrucciones que nos permiten lograr un resultado: que el ascensor nos transporte. Cuando se pone en marcha la función «llamar al ascensor», se ejecuta un subalgoritmo con las siguientes instrucciones:

- Localizar en qué planta está el ascensor.
- Comparar si es mayor o menor que nuestra planta.
- Saber si tiene que subir o bajar para llegar.
- Poner en marcha los motores que permiten mover el ascensor.
- Comprobar en cada planta si ha llegado a su destino.
- Parar los motores cuando esté en nuestra planta.
- Abrir las puertas para que nos podamos montar.

Esta función forma parte de un programa más grande, que contendrá otras funciones: qué hacer cuando entramos al ascensor y pulsamos un botón, cuándo cerrar las puertas del ascensor, etc.

Los parámetros

Los parámetros son los datos que necesita una función para que se ejecute correctamente. Según el tipo de función, se necesitan determinados valores. Los parámetros se fijan cuando llamamos a la función. Normalmente, se especifican a continuación del nombre de la función.

Llegados a este punto, es importante distinguir un valor o parámetro de una variable: las variables contienen valores. Las funciones trabajan con valores y no con variables; si modificamos un valor en una función, la variable no se verá alterada. Por ejemplo:

Funciones. Ejemplo 3

Si para una función que expresa los minutos necesitamos el dato de una variable expresada en horas, dentro de nuestra función multiplicaremos el número de horas que hemos recibido como parámetro por 60 (horas · 60), pero la variable seguirá mostrando las horas.

Mostrarminutos (horas · 60)

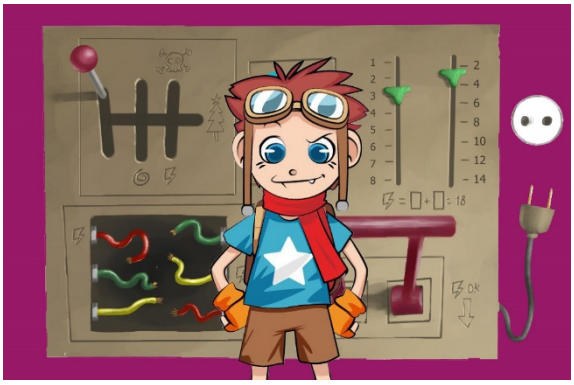
A fondo

***Arcadeland*, la aventura narrativa para aprender a programar**

Ortega, B., Alberca, J. y Arribas, A. (agosto, 2015). Bienvenidos a *Arcadeland* [Juego narrativo en línea]. DIWO: Do It With Others.

Arcadeland es una aventura narrativa en la que acompañarás a Alan, un niño al que un día le sucedió algo muy extraño y necesita tu ayuda. Alan y tú viviréis un montón de aventuras alucinantes, incluyendo tesoros, troles y ejércitos robóticos.

Con este juego aprenderás la lógica imprescindible para poder desenvolverte en cualquier lenguaje de programación. En cada uno de los seis episodios se trabaja un concepto concreto: algoritmos, condicionales, variables, etc.



Accede al juego a través del aula virtual o desde la siguiente dirección web:

http://diwo.bq.com/arcadeland/

Programación.

DIWO: Do It With Others (2015). *Programación.* [Manual en línea].

En este documento se explican los elementos principales de cualquier programación: algoritmos, variables, condicionales, bucles, funciones, diagramas de flujo, etc.

Accede al manual a través del aula virtual o desde la siguiente dirección web: http://diwo.bq.com/wp-content/uploads/2015/08/Programaci%C3%B3n.pdf

Programación y pensamiento computacional

Ortega, B. (junio, 2015). Programación y pensamiento computacional [Artículo en línea]. DIWO: Do It With Others.

En este artículo se explica por qué la programación puede ser un buen aliado en el desarrollo de un pensamiento concreto: el pensamiento computacional.

Accede al artículo a través del aula virtual o desde la siguiente dirección web: http://diwo.bq.com/programacion-y-pensamiento-computacional/

¿Qué es STEAM?

Ortega, B. (abril, 2016). ¿Qué es STEAM? [Artículo en línea]. DIWO: Do It With Others.

La educación y la sociedad actual demandan un enfoque STEAM, pero ¿qué es STEAM y por qué es importante? En este post se explica más en profundidad este concepto.

Accede al artículo a través del aula virtual o desde la siguiente dirección web: http://diwo.bq.com/que-es-steam-educacion/

Ideas para la programación en el aula

Educación 3.0 (s. f.). Ideas para la programación en el aula [Etiqueta con artículos en línea].

Mediante esta etiqueta, en la página web de la revista *Educación 3.0*, podrás acceder a multitud de experiencias educativas con programación.

EDUCACIÓN 3.0
LÍDER INFORMATIVO EN INNOVACIÓN EDUCATIVA

Accede a la página a través del aula virtual o desde la siguiente dirección web: https://www.educaciontrespuntocero.com/tag/ideas-para-la-programacion-en-el-aula

Recursos para una educación STEAM y un aprendizaje por proyectos

El profesor Javier Tourón nos introduce al mundo de las experiencias STEAM y nos ofrece recursos disponibles en Internet para trabajar el ABP (aprendizaje basado en proyectos) y diferentes áreas de las STEM. También pone a nuestro alcance recursos para relacionar STEM y STEAM.

Accede a los recursos a través del aula virtual o desde la siguiente dirección web:

https://www.javiertouron.es/recursos-para-una-educacion-steam-y-un/

Actividades

Caso práctico: Programa un cuadrado con Sphero Mini

Una vez visto el contenido del tema, deberás realizar un programa con Sphero Mini con el que al igual que en el caso de Scracth puedas «dibujar» un cuadrado. Para ello, sigue los siguientes pasos:

1. Paso 1: realiza el siguiente programa usando Sphero Edu.

Figura 1. Paso 1: crear un programa.

2. Paso 2: pulsa el botón «empezar y comprueba» el movimiento del robot en la pantalla de trayectoria.
3. Paso 3: añade las siguientes secuencias que aparecen en la imagen y vuelve a realizar el paso 2.

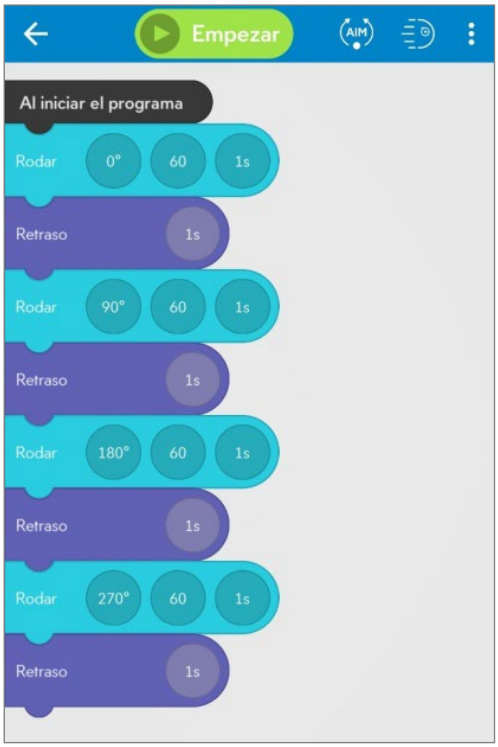


Figura 2. Paso 2: añadir secuencias.

4. Paso 4: optimiza el programa realizado incluyendo los bloques de «bucle» que puedes encontrar en Controles.

   - Ejecuta y graba las tres trayectorias, editando a tu gusto.
   - Súbelo a YouTube y entrega el enlace del vídeo.

5. Para finalizar, no olvides la importancia de trabajar en equipo, colabora con tus compañeros resolviendo las dudas que vayan planteando en el foro a la hora de realizar este trabajo.

Objetivos

- Combinar diferentes tipos de secuencias.
- Aprender a realizar diferentes programas para una misma acción.
- Comprender el uso de bucles.
- Trabajar en equipo.

Criterios de evaluación

A la hora de evaluar la actividad se tendrán en cuenta los siguientes aspectos:

- La correcta explicación del proceso seguido para lograr el perfecto funcionamiento (hasta 3 puntos).
- El funcionamiento del programa (hasta 3 puntos).
- La diferenciación de las trayectorias (hasta 3 puntos).
- La publicación del vídeo en un canal de YouTube (hasta 1 punto).

Extensión máxima: 2 páginas con el enlace del vídeo de una duración máxima de 3 minutos y la explicación del proceso seguido para su perfecto funcionamiento, fuente Georgia 11 e interlineado 1,5.

Test

1. ¿Qué condicional utilizamos si tenemos varias condiciones?

A. `switch-case`.

B. `if`.

C. `if-else`.

D. `while`.

2. ¿A qué llamamos *instrucciones de control*?

A. Variables y funciones.

B. Condicionales y variables.

C. Bucles y funciones.

D. Condicionales y bucles.

3. En un día lluvioso, para saber si tengo que abrir o cerrar mi paraguas, utilizaría:

A. `while`.

B. `switch-case`.

C. `if-else`.

D. `if`.

4. ¿Qué bucle utilizaríamos para que una luz LED se encienda exactamente 8 veces?

A. `for`.

B. `if`.

C. `while`.

D. `switch-case`.

5. Para asegurarnos de que alguien no deja de mirar a la carretera, aunque le llamen por teléfono, utilizaríamos:

   A. Condicional `if`.

   B. Bucle `while`.

   C. Una variable.

   D. Bucle `for`.

6. Si queremos utilizar durante todo el programa el número de veces que se han encendido las luces como parte de diferentes operaciones, utilizamos:

   A. Una instrucción de control.

   B. Una función.

   C. Una variable global.

   D. Una variable local.

7. Si queremos utilizar Sphero Mini para contar las veces que nuestros alumnos han contestado bien a una pregunta y que, al llegar a 10 respuestas correctas, vibre Sphero, para saber que ha llegado a 10, usaremos:

   A. Un bucle.

   B. Una función.

   C. Un condicional.

   D. Una variable.

8. Una función es:

   A. Un subalgoritmo.

   B. Un dato.

   C. Una instrucción.

   D. Una representación del programa.

9. En un robot de cocina, freír un espárrago sería:

A. Una variable.

B. Una función.

C. Un condicional.

D. Un bucle.

10. Si en una función queremos obtener un valor en segundos y el programa nos lo ofrece en minutos, multiplicaremos la variable que nos da ese valor por 60, y entonces:

A. La variable seguirá valiendo lo mismo.

B. La variable cambiará su valor.

C. No se puede trabajar con variables en una función.

D. La variable se igualará a 0.