eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Departamento de Lenguajes y Sistemas Informáticos
Escuela Universitaria de Ingeniería de Vitoria-Gasteiz

MANUAL DE SCRATCH 2

Manual para la asignatura de

Fundamentos de Informática, editado por

Pablo González Nalda

versión de

22 de agosto de 2014

MANUAL DE SCRATCH 2

22 de agosto de 2014

"Si no hay motivación, no hay nada que hacer"

Carlos Sainz, campeón del mundo de rallyes.

 "Como todo gran plan, mi estrategia es tan simple que la podría haber ideado un idiota"

Zapp Brannigan, General en el ODP y Capitán de la nave Nimbus de "Futurama", sobre planes y objetivos.

Prefacio

¿Por qué? ¿Para qué?

En este manual se busca un acercamiento intuitivo a la programación para estudiantes de primero de ingenierías no informáticas. Muchas veces el problema de aprender a programar consiste en la falta de motivación. Scratch es entretenido y visual.

Cuestiones legales

uso que hace de su obra).

- Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Agradecimientos

Agradezco al gato de la portada y a toda su familia la ayuda para hacer este manual.

Quiero agradecer a Sergio Mendoza y otros muchos integrantes de la comunidad del Software Libre por colaborar con su trabajo para el beneficio mutuo. En concreto Sergio preparó muy bien un esquema de LaTeXpara escribir tesis doctorales, que es el que he usado y completado. También gracias a los desarrolladores de GNU/Linux, Ubuntu, KDE, Kile, LaTeX, Gimp y otras muchas herramientas que he usado en este trabajo.

Índice general

Índice de figuras

Resumen y organización

"Lo bueno, si breve, dos veces bueno"

Baltasar Gracián, escritor español. No es necesario alargarse para explicarse.

"Seré breve, porque cuanto menos hablo, menos me equivoco"

El meteorólogo de La Sexta.

"Ante todo, mucha calma"

Título de un disco en directo de Siniestro Total. La calma y serenidad permite una mejor organización.

Resumen

En este manual se presenta un aprendizaje de la programación en Scratch para personas adultas. Se presentan acciones incrementales para acceder a los conceptos. No se busca explicar todas las funcionalidades de Scratch 2 sino utilizarlo para introducir los conceptos de programación básica necesarios en una ingeniería.

Este lenguaje de programación no está diseñado para ello, por lo que algunas cosas *"están un poco traídas por los pelos"*, como la operación de devolver valor de las funciones. Sin

embargo, estos detalles se esperan pulir en los LP más habituales.

Organización

El manual contiene los capítulos habituales en el aprendizaje de un lenguaje de programación desde cero en una asignatura de primer curso de ingeniería: descripción del entorno de programación y primeros pasos, el concepto de variable y operadores, entrada y salida, estructuras de control de flujo, subprogramas y tipos de datos compuestos.

CAPÍTULO

1 Primeros pasos

"Intentar algo es el primer paso para el fracaso"

Homer Jay Simpson, pesimista.

Este capítulo muestra lo que hay que hacer para dar los primeros pasos en Scratch 2 una vez se tiene correctamente instalado el programa.

1.1. Instalación y entorno

Scratch 2 se puede descargar desde la web del MIT o desde el servidor de LSI:

http://lsi.vc.ehu.es/pablogn/docencia/FdI/Scratch.html

El fichero Scratch.air se instala con Adobe Air (http://get.adobe.com/air/).

El funcionamiento de Scratch es muy sencillo. Se pueden seguir vídeos de Youtube para aprender su manejo, como los que están enlazados en la primera página referenciada. Ya que el objetivo de la asignatura para la que se prepara este manual no es la parte gráfica de Scratch

sino la visualización de conceptos de programación, parte de los vídeos no tienen que ver con la asignatura.

Scratch 2 es un avance desde la primera versión, e incluye nuevas posibilidades interesantes para esta asignatura, para los objetivos de este manual. Sin embargo, para una parte del curso podrá usarse cualquiera de las dos versiones. Como las dos versiones tienen un entorno distinto al cambiar muchos elementos de posición, este manual dará indicaciones para el entorno de la versión 2.

1.2. Primeros pasos

Este manual "lleva de la mano", es decir, está escrito de forma que se vaya haciendo lo que indica mientras se lee. De todas formas, en cualquier momento se puede probar lo que se te ocurra porque la forma de aprender es probar, observar, suponer el resultado de lo que hacemos, equivocarnos en esa suposición y volver a probar hasta que el programa haga lo que queremos.

Dicho esto, vamos a empezar.

Nada más abrir Scratch hay tres zonas, una con un gato (arriba una bandera verde y un octógono (señal de *Stop*)), otra con bloques de color en unas pestañas y una zona vacía.

Por ahora sólo vamos a usar bloques con muescas tanto arriba como abajo.

Arriba, pulsa la pestaña *Scripts* y elige *Motion*, movimiento.

Arrastra el elemento o *bloque* "mover 10 pasos" (*move 10 steps*) a la zona de la derecha y haz doble clic sobre él. El gato se mueve hacia la derecha.

Escribe el número 100 en vez del 10 que está en el bloque y haz doble clic sobre el bloque. El gato se mueve más hacia la derecha. Puedes arrastrar al gato hacia el centro.

Vamos a añadir un giro, arrastra el bloque "gira 15 grados" (*turn 15 degrees*) y sustituye por 90. Elige hacia dónde debe girar. Encaja un bloque con otro. Verás que una zona se pone de color blanco, en ese momento suelta el bloque y se unirán. Haz clic sobre uno de los dos bloques y se realizará la acción. El resultado es el de la figura 1.1. Si haces clic cuatro veces seguidas, el gato se moverá haciendo un cuadrado.


```
Mueve 10 pasos
Gira 90 grados
```

Figura 1.1: Muévete y gira. Si se clica 4 veces, hace un cuadrado. A la derecha, una representación del programa en Scratch.

Para que dibuje, pulsa *Pen*, de color verde, y clica *pen down* (baja lápiz). Puedes hacer la receta anterior y dibujará el cuadrado. *Clear* borrará la zona del gato. Mira la figura 1.3.

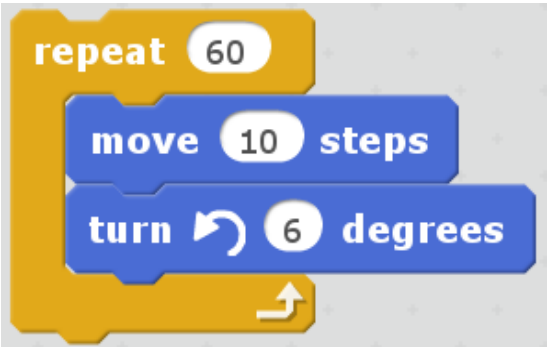
1.3. Repetición

¿Qué deberíamos hacer para que dibuje el cuadrado con un solo clic? La forma más intuitiva es repetir cuatro veces los dos bloques: `mover, girar, mover, girar, mover, girar, mover, girar`. Comprueba que funciona. Una ayuda: pulsa botón derecho sobre el bloque que está más arriba y elige *duplicar*. Puedes soltarlo cuando se ilumina la zona de unión.

Podemos dibujar un círculo (algo parecido, un polígono regular de más de 20 lados ya parece una circunferencia) con este método si giramos, por ejemplo, sólo 6 grados 60 veces. El movimiento debe ser pequeño, 10 pasos, o se saldrá de la pantalla. Si se sale, para "recuperarlo" le ordenaremos que vaya a las coordenadas (0,0) clicando en el bloque "set x to 0" y de forma equivalente en el "set y to 0".

Si eres "prisas", habrás pulsado el botón derecho sobre el primer bloque para "duplicar". Eso facilita el trabajo, pero está claro que tiene que haber un método mejor que repetir lo mismo 60 veces.

Pulsa "Control", en naranja, y verás unos bloques con forma normal, y otros con forma de C e incluso de E. Escoge la llamada "Repite 10" e intenta abrazar con ella los dos bloques (mover,girar). Cambia el 10 por 60. Debería quedar como en la figura 1.2.

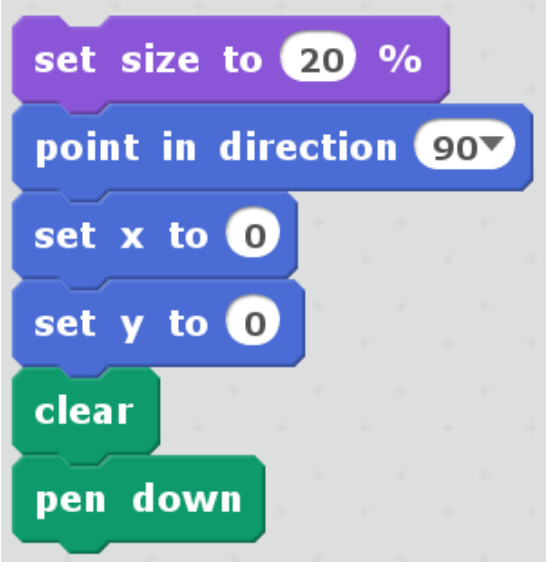



```
Repite 60
        Mueve 10 pasos
        Gira 6 grados
```

Figura 1.2: Polígono de 60 lados, casi un círculo. A la derecha, una representación del programa en Scratch.

Observa que el bloque naranja hace repetir lo que se encuentra dentro las veces que indica.

Puedes hacer más pruebas, y para facilitar las cosas haz otro grupo de bloques o *programa* con los bloques de limpiar, bajar lápiz, ir a las coordenadas (0,0). También podemos hacer el gato más pequeño para que no moleste al ver el resultado. El programa sería algo así como el de la figura 1.3.




```
Tamaño 20%
Apunta en dirección 90
Ir a x 0
Ir a y 0
Limpia
 Bajar lápiz
```

Figura 1.3: Programa que inicia y permite probar repetidamente.

Vamos a hacer un dodecágono regular, repitiendo 12 veces y girando 30 grados. El movimiento puede ser de 30 pasos.

```
Repite 12
       Mueve 30 pasos
       Gira 30 grados
```

Si añadimos una instrucción de Girar 60 después del Repite, pegado debajo y fuera de la forma de C que recoge los dos bloques de dentro:

```
Repite 12
       Mueve 30 pasos
       Gira 30 grados
Gira 60 grados
```

Haz clic sobre este programa seis veces. Hace un curioso dibujo. Esto nos tiene que recordar que podemos repetir algo con el bloque “Repite”. Vamos a poner otro bloque “Repite” que abarque todo lo que hemos hecho:

```
Repite 6
       Repite 12
              Mueve 30 pasos
              Gira 30 grados
       Gira 60 grados
```

Observa que el “Gira 30” está dentro de todos los Repite (se va a repetir 6 x 12 = 72 veces), mientras que el “Gira 60” está fuera del “Repite 12” y dentro del “Repite 6”, así que se repetirá seis veces.

Si al final del ciclo interior del anterior programa ponemos el bloque “cambiar el color del lápiz en 30” podremos ver que el color cambia para cada lado de los dodecágonos. Sin embargo, si se pone junto al “Gira 60” habrá seis dodecágonos de color uniforme.

El resultado final será el de la figura 1.4:

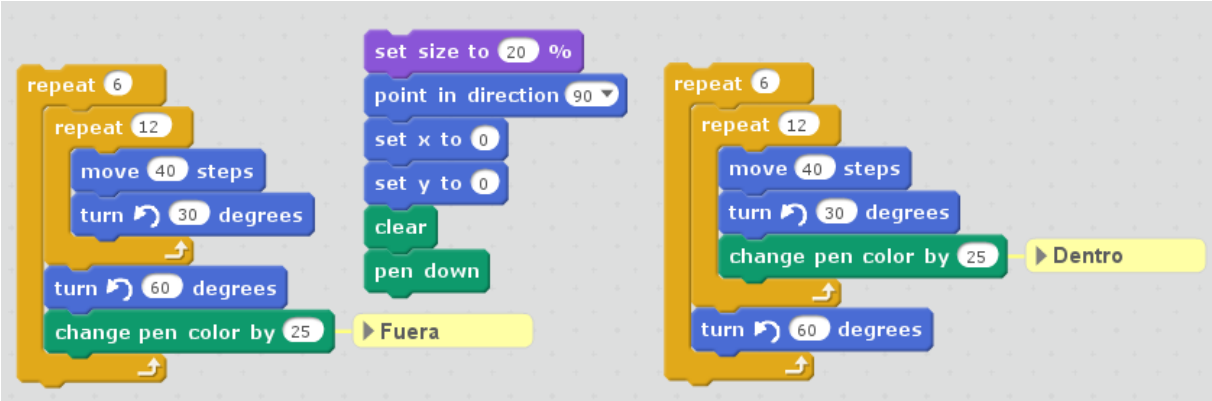


Figura 1.4: Repeticiones *anidadas*, una dentro de la otra. Diferentes efectos si un bloque está dentro o fuera.

CAPÍTULO

2 Variables y Entrada/Salida de datos

Vamos a ver qué es una variable y cómo se pueden usar. También veremos cómo se introducen y muestran datos en Scratch.

2.1. Variables propias de Scratch

En el ejemplo anterior vemos que el color va cambiando “de 30 en 30”, es decir, está determinado por un número y que el programa se encarga de almacenar. Una variable es eso, un lugar en la memoria del ordenador donde se almacena un tipo de dato. Según el lenguaje de programación se puede diferenciar entre diferentes tipos de datos, como números enteros, números reales, caracteres (que incluyen letras, cifras y símbolos del teclado), cadenas de caracteres (textos) y combinaciones de los anteriores. Cada lenguaje de programación pone los límites en cómo se mezclan o distinguen los tipos de datos, el número máximo que se puede almacenar, y otros detalles en los que no nos vamos a fijar ahora.

En este caso es un entero. Podemos cambiarlo con el bloque que hemos usado, o podemos usar el bloque “asignar al lápiz el color 0” o cualquier otro valor.

Aunque nos ha servido para ver la necesidad de las variables, el color es un caso especial porque en realidad los colores en la pantalla son más complicados que un solo número.

Vamos a usar otras variables también muy interesantes: las coordenadas del gato. Probamos el siguiente programa:

```
Repite 6
       Cambia x en 10
       Cambia y en 10
```

Y nos dibuja unos escalones. En cada uno de esos bloques es como si tecleáramos en una calculadora "+10 =" Incrementa el valor de la pantalla de la calculadora en 10. Ese valor de la pantalla de la calculadora es también una variable, como el de sus memorias.

Si cambiamos el signo de uno de los "10" y ejecutamos (clic sobre el Repite) tenemos otras escaleras en otro sentido. Son números enteros. ¿O no? Prueba a introducir un número como 30.4567 en vez del 10 en el programa anterior. Cuidado, depende del idioma no dejará escribir la coma como separador de decimales.

Copia este programa. Para los comentarios, pulsa botón derecho sobre el bloque al que se enlazan. Observa que el bloque con los extremos redondeados que contiene la expresión "posición de x" representa a la variable o coordenada *x*. Este bloque se coloca sobre el hueco blanco a la derecha de "Piensa". Así aparece en el *bocadillo* del gato el contenido de la variable *x*.

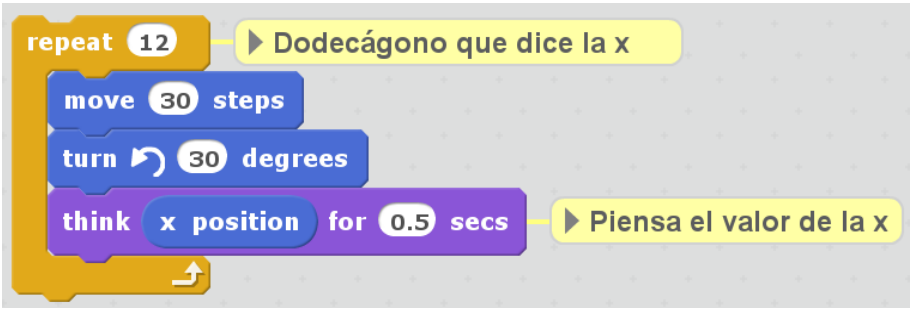


Figura 2.1: Uso de variables y comentarios.

Como ves, un comentario puede aclarar detalles de lo que hace un bloque, varios, o un programa entero.

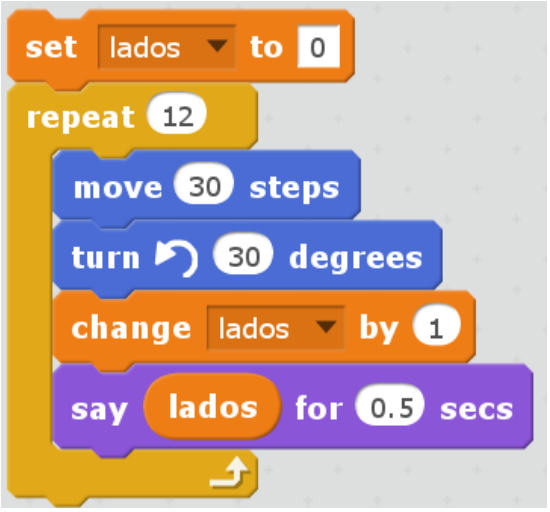
2.2. Declaración de variables

Con las variables propias del Scratch (color, coordenadas y alguna más que no nos importa mucho) no se pueden hacer muchas cosas. Por ejemplo, si queremos contar los lados del polígono que llevamos, necesitamos una variable que contenga ese valor. Para ello:

1. Pulsa *Data*
2. Pulsa *Haz una variable*
3. Teclea "lados" como nombre de la variable.

Aparecen varios bloques. El primero tiene una marca para que aparezca una caja con su contenido. Los demás son iguales que los que hemos usado para las coordenadas.

Prepara este programa, que cuenta los lados hechos. Evidentemente, si repetimos 12 veces y cambia al final de cada vuelta, contará 12.




```
Asigna a "lados" el 0
Repite 12
      Mueve 30 pasos
      Gira 30 grados
      Cambia "lados" en 1
      Piensa "lados" por 0.5 segundos
```

Figura 2.2: Programa que cuenta los lados de un dodecágono.

Cuidado: en el último bloque es diferente escribir el texto “lados” después del “Piensa”, que poner el óvalo naranja de la variable *lados*. Ésta es la diferencia entre una variable y una constante, ya que una variable tiene un texto como nombre, para representarla, y una constante es un texto entre comillas o un número.

La primera instrucción (bloque Asigna) se llama *inicialización*. Observa qué pasa si la pones dentro del Repite, de esta forma:

```
Repite 12
      Asigna a "lados" el 0
      Mueve 30 pasos
      Gira 30 grados
      Cambia "lados" en 1
      Piensa "lados" por 0.5 segundos
```

2.3. Introducción de datos y operadores

Para introducir datos al programa (lo que se llama en informática *entrada/salida*) usamos la instrucción o bloque “pedir y esperar” (*ask and wait*) junto con la variable “respuesta” (*answer*). También vemos en la figura 2.3 los operadores, en este caso la multiplicación.

Prueba qué hace un programa en el que tengas un valor en una variable “cantidad” y ponga un bloque parecido al anterior: *set cantidad to cantidad*2*

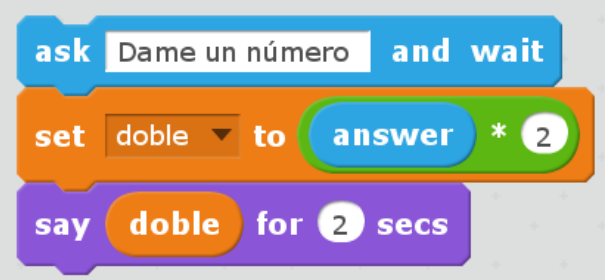


Figura 2.3: Entrada y salida de datos y operaciones básicas.

2.4. Ejercicios

Documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa. Grábalos en disco para poder modificarlos posteriormente.

1. Haz una calculadora €- $ (toma como cambio 1€= 1,3$)
2. Convierte temperaturas Fahrenheit a Celsius (grados centígrados).
$$^{\circ}C = (^{\circ}F - 32) \cdot \frac{5}{9}$$
3. Crea un programa que calcule potencias de exponente natural como una iteración o repetición de multiplicaciones de la base: $a^b = a \cdot a \cdot a \cdot \ldots \cdot a$ (b veces)
4. Pregunta el número de lados y dibuja el polígono con esos lados. Gira $(360/lados)$
5. Pregunta un número de días y conviértelos a años (de 365 días), meses (de 30 días) y días.
6. Crea un programa que haga tiradas de dados, es decir, que el gato dé números del 1 al 6.
7. Haz que el gatete pregunte cuántos dados quieres tirar y que dé la suma de las n tiradas.
8. El gato es rolero, así que tiene dados de todo tipo. Haz que pregunte cuántas caras tienen los dados que tira (4, 8, 10, 12, 20...).
9. Haz que pregunte en qué rango debe estar la tirada, por ejemplo, entre 37 y 85.
10. Vamos a simular una hormiga (en concreto, el *movimiento browniano*). Baja el lápiz, coloca un bucle infinito (en control, el bloque *forever*) y dentro de él, la instrucción para mover un paso hacia adelante, y gira los grados que diga el bloque "dame aleatorio" (*pick random*) de -50 a 50. Prueba con otros valores.
11. Haz un programa que diga los 10 primeros elementos de la *Secuencia de Fibonacci*, en la que los 2 primeros números son 1, y los siguientes son la suma de los dos anteriores.
$$F_0 = 1,\ F_1 = 1,$$
$$F_i = F_{i-1} + F_{i-2}\ para\ i > 1$$

CAPÍTULO

3 Condicionales y condiciones

 "– ¿Puede traducirlo?

 – ¡Por supuesto, pero sólo a β-código3! ¡Se trata de una lengua tan compleja que es más difícil de entender todavía!'

 – No le pido explicaciones totalmente irrebatibles. Sólo le pido que lo haga."

Fry al Dr. Farnsworth, el gran científico de Futurama

En este capítulo probaremos las estructuras condicionales y las condiciones que se pueden usar.

3.1. Condicionales

Vamos a hacer que el dodecágono tenga lados de colores alternados, es decir, que los lados pares sean de un color y los impares de otro. Para ello, vamos a usar la variable que numeraba

los lados del último programa.

Pero primero vamos a ver cómo detectamos los pares. Queremos algo parecido a:

Si "lados" es par entonces di ¡Par!

¿Pero cómo sabemos si el número es par? Hay una operación que consiste en quedarnos con el resto de la división entera, y se llama *mod*. Por eso, 7 *mod* 2 nos da 1 y 6 *mod* 2 nos da 0. Es un poco laborioso crear la instrucción-bloque compuesta de bloques, pero es fácil. Debe quedar aproximadamente como en la figura 3.1.

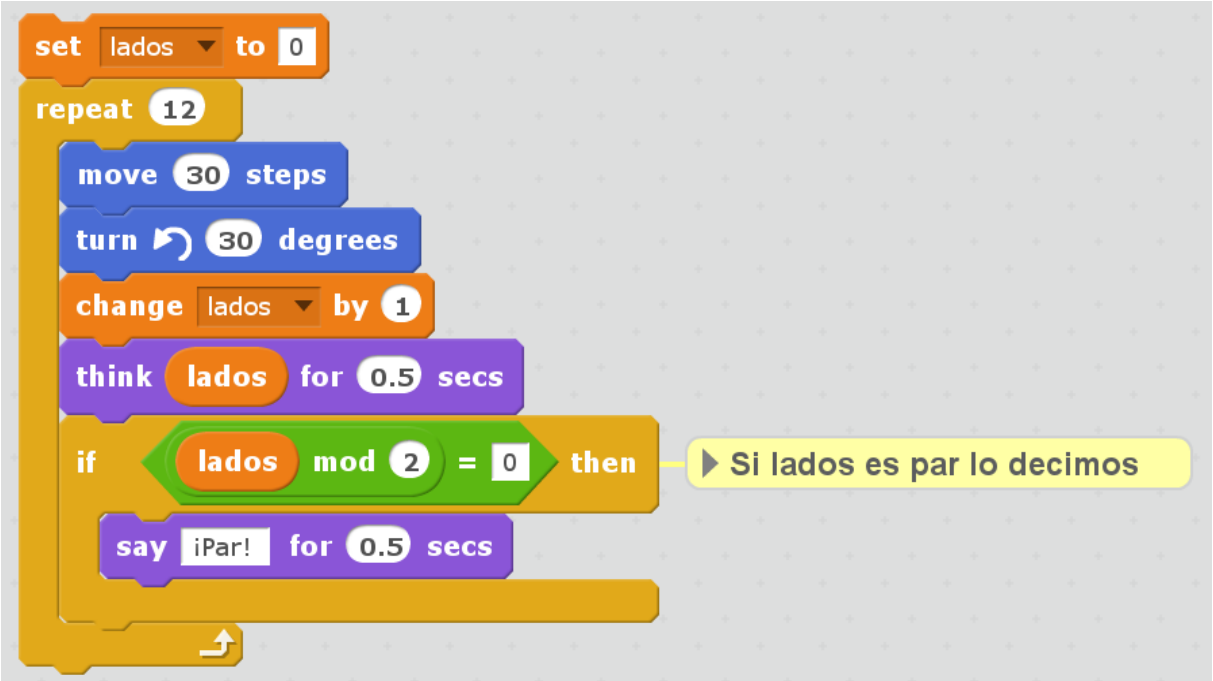


Figura 3.1: Condicional simple. Si el resto de dividir lados entre 2 da 0, entonces di "*Par*"

Y si queremos que diga impar, usamos la estructura condicional con forma de E *if-then-else*. En la parte de abajo pondremos que diga *Impar*.

En vez de decir par o impar, podemos cambiar el color con el que se dibuja. El *if* quedaría así:

```
Si ("lados" mod 2) = 0 entonces
        Asigna a color del lápiz el rojo
si no,
        Asigna a color del lápiz el verde
```

Observa que no es lo mismo cambiar el color antes de moverse (cuando se usa) o después de dibujar pero antes de cambiar el valor de la *variable* lados. El orden de las instrucciones

condiciona el resultado. Además, la variable lados empieza con el valor 0 y cambia a 1 cuando ya se ha dibujado el primer lado.

En la figura 1.4 hemos visto que se puede colocar un *Repite* dentro de otro *Repite* y en la figura 3.1 hemos insertado un bloque condicional dentro de uno repetitivo. De la misma forma podemos combinar los condicionales y los repetitivos con los demás de la forma que queramos y necesitemos para crear un programa con sentido. Todos estos bloques o instrucciones se denominan de control de flujo (de la ejecución).

También hemos visto que tenemos diferentes tipos de bloques: los que se pueden enlazar como ladrillos y con muescas arriba y abajo, las variables con extremos redondeados, y las comparaciones con extremos en ángulo recto.

3.2. Condiciones

Las condiciones tienen forma de bloque de extremos en ángulo recto. Por una parte tenemos los comparadores $>, <, =$ que se pueden usar entre dos números o bloques de extremos redondeados. En los cuadrados se pueden escribir números o cadenas de caracteres (textos), además de bloques redondeados o "angulosos".

Por otra parte, tenemos las operaciones *booleanas*: conjunción, disyunción y negación, que se escriben *and, or, not* (*y,o,no*, respectivamente). *And* necesita que se cumplan las dos condiciones, *or* una por lo menos, y *not* hace lo contrario.

$$3 < 5 \; and \; 7 < 4 \; es \; falso$$

$$3 < 5 \; or \; 7 < 4 \; es \; verdadero$$

$$not(7 < 4) \; es \; verdadero$$

Una comparación da verdadero o falso (*true, false*). Si estamos comparando cadenas (texto) se usa el alfabeto siendo menor *a* que *b* y *aa* que *ab*, pero mayor *ba* que *ab*.

La respuesta "true" o "false" se da a su vez en forma de cadena. Por ello hay que tener cuidado al comparar tres números. Para ver si 3 <x <5 tendremos que construir

$$3 < x \; and \; x < 5$$

o de forma equivalente,

$$(x > 3) \; and \; (x < 5)$$

También tendremos que usar *and* para $\geqslant$ o $\leqslant$ y el *not* para $\neq$. Ante todo, hay que **pensar qué se quiere hacer** y qué equivalencias entre operaciones se pueden aplicar para simplificar la expresión.

De la misma forma que matemáticas se define el orden de evaluación o *precedencia* con paréntesis y corchetes, en Scratch se define con el orden en el que se introducen unos bloques en otros. Se empieza a evaluar por el más interno y se acaba por el más externo.

Los bloques redondeados en el apartado "Operadores" son operaciones matemáticas y otros elementos, y producen números y textos, es decir, pueden sustituir a cualquier valor constante (como los grados que hay que girar o lo que tiene que pensar el gatete) y pueden usarse también como si fueran una variable. La ventaja del Scratch en este punto es clara, puesto que sólo deja usar un bloque donde tiene sentido.

3.3. Ejercicios

Graba y documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa.

1. Pide dos números y el programa tiene que decir si son múltipos entre sí: 8 y 7 no son múltiplos, pero 3 y 6 sí, y también 6 y 3.

2. Haz una calculadora € $\leftrightarrow$ \$ (toma como cambio $1€ = 1,3\$$) preguntando en qué sentido es la conversión, es decir, cuál es la moneda origen.

3. Convierte temperaturas Fahrenheit a Celsius (grados centígrados) o en sentido contrario, pregunta el sentido de la conversión.

$$^\circ C = (^\circ F - 32) \cdot \frac{5}{9}$$

4. Pregunta el número de lados y dibuja el polígono con esos lados. Gira $(360/lados)$. Ahora, ya que sabemos hacer condiciones, que sólo haga el polígono si el número de lados es mayor que 2.

5. Pregunta el número de días y conviértelos a años (de 365 días), meses (de 30 días) y días, comprobando que es un número de días mayor que cero, y que diga los resultados **si no son** cero, es decir, que **no** diga *1 años, 0 meses, 0 días*.

6. Haz un programa que pida tres números y diga si los números han sido introducidos en orden ascendente, descendente o desordenados.

CAPÍTULO

4 Iterativas

Hemos visto ya un bloque de control iterativo en la sección 1.3: el bloque *Repite*. Con él podemos ejecutar un subprograma un número determinado de veces. Como ejemplo, podemos preparar un juego como el que se presenta a continuación.

4.1. Juego con “Repite”

Con lo que sabemos, casi se puede hacer el juego de la figura 4.1. No voy a explicar qué hace en detalle cada parte, porque ése es tu objetivo, copiarlo, probarlo y deducir qué hace cada bloque.

Lo único que nos falta para entenderlo es saber dos cosas. La primera es cómo hacer que nos tecleen algo, ya sea un número o una cadena, para entender el juego. Vemos los bloques azules, “Pregunta y espera”. Lo que hace es que el gato presenta el texto incluido y se para hasta que tecleamos algo y damos al *tick* de aceptar. Lo tecleado se almacena en la variable especial “Respuesta” (*answer*).

Hay otro bloque que no habíamos usado, y es *join*, “unir”. Lo que hace es *concatenar*, la operación de pegar un texto con otro, y es muy útil para que aparezca un texto antes de una variable. En este lenguaje de programación se convierte automáticamente el contenido de la variable a texto para concatenarlo.

Es muy importante que entiendas bien cada uno de los pasos que da el programa de la figu-

ra 4.1 antes de continuar, puesto que de lo contrario no se puede entender nada de lo siguiente. Un truco: pártelo en trozos y ejecútalos de uno en uno, haciendo clic en cada uno y viendo mientras qué valor reciben las variables.

He colocado algún comentario como pista, pero es necesario ejecutarlo en el ordenador para comprenderlo. Precisamente es una buena práctica la de decir qué significa cada variable, qué sentido tiene el valor que contiene, si es la cantidad de veces que se hace algo, si es el nombre de un jugador, si sirve para saber si se ha encontrado o no la respuesta ...

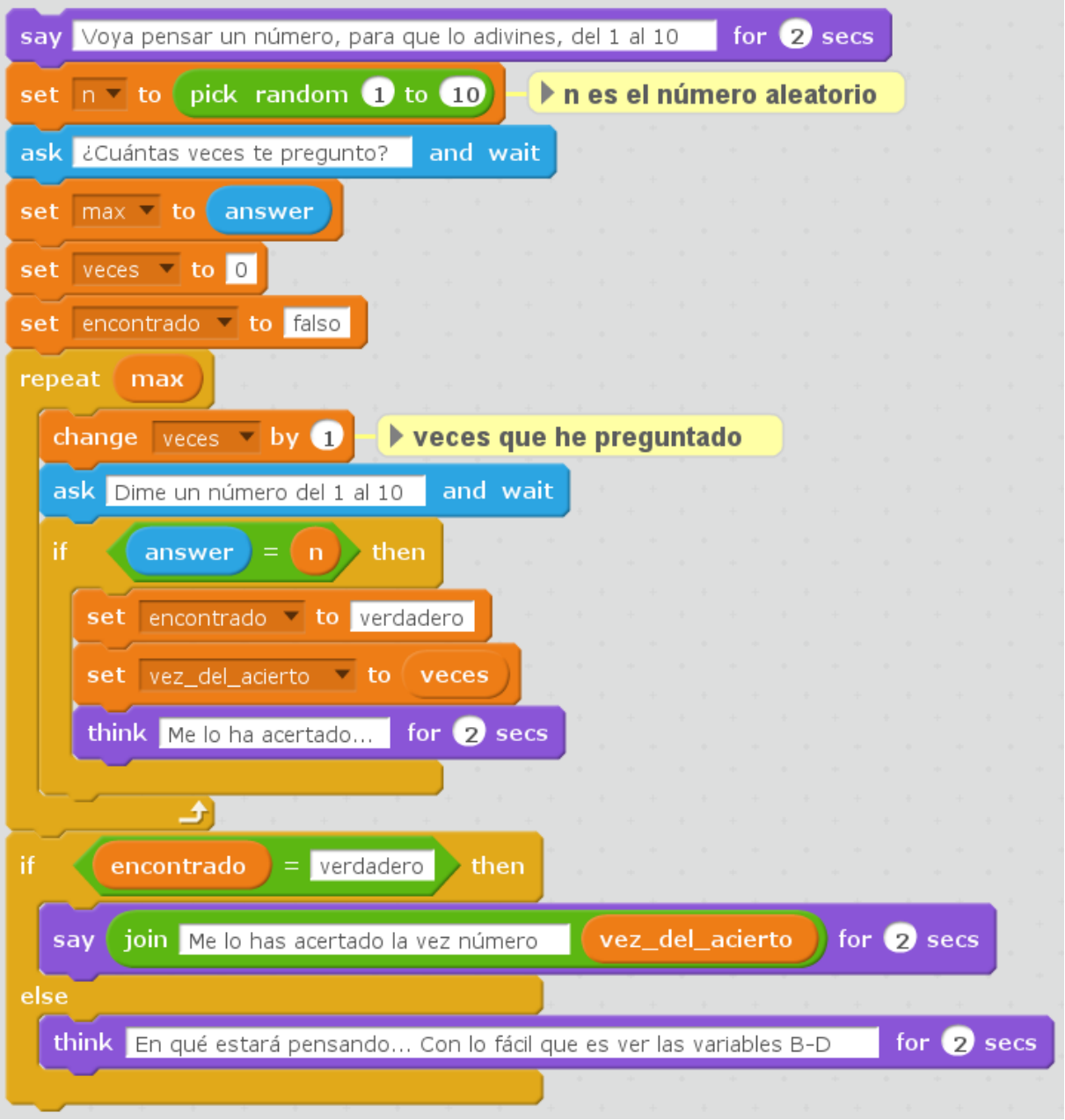


Figura 4.1: Juego de adivinar números. Analiza cómo funciona.

4.2. Juego con "Repite Hasta"

De todas formas, el juego que se ha programado en la figura 4.1 es un poco extraño, porque se suele acabar de jugar cuando se acierta el número. En este caso el programa sigue preguntando cuando se ha acertado. Para ello vamos a usar una nueva estructura iterativa que repite hasta que se cumple una cierta condición. La instrucción es *"repite hasta"* <*condición*> . En nuestro caso, la condición que se debe cumplir para dejar de preguntar es que se haya acertado el número pensado por el gato (almacenado en la variable *n*).

Vamos a centrarnos en la parte del programa que se repite un número *determinado* de veces con la instrucción *repite*. Queremos que se repita un número ***in****determinado* de repeticiones con el *repite hasta*. Es importante distinguir el número de veces que se necesita repetir para solucionar el problema. En este caso quedaría así:

```
Repite hasta que encontrado=verdadero
      Cambia veces en 1
      Pide "Dime un número del 1 al 10" y espera
      Si respuesta=n entonces
              Asigna a encontrado el valor verdadero
              Asigna a vez_del_acierto el valor de veces
```

Observa que la condicional posterior a este trozo, que compara encontrado con verdadero, siempre se cumplirá puesto que sólo se sale del bucle o ciclo repetitivo cuando se produce esta situación, que el número haya sido acertado.

Ahora vamos a restringir el número de veces que se pregunta al valor que nos dan al preguntar, y que introducimos en la variable `max`.

```
Repite hasta que encontrado=verdadero o veces=max
      Cambia veces en 1
      Pide "Dime un número del 1 al 10" y espera
      Si respuesta=n entonces
              Asigna a encontrado el valor verdadero
              Asigna a vez_del_acierto el valor de veces
```

En este caso es necesaria la condición que explica al usuario si se ha salido del bucle porque se ha acertado el número o porque se ha agotado el número de intentos. Además pueden ocurrir las dos razones a la vez, ya que se puede adivinar el número en el último intento válido, con lo que la siguiente vuelta se tendría que salir por ambas situaciones.

Con este análisis llegamos a la conclusión de que hace falta primero entender el problema y después formalizarlo con un programa.

Lo anterior lo podríamos simplificar así:

```
Pide "Dime un número del 1 al 10" y espera
Repite hasta que respuesta=n o veces=max
      Cambia veces en 1
      Pide "Dime un número del 1 al 10" y  espera
Asigna a vez_del_acierto el valor de veces
```

La primera vez que se pide es una instrucción que se ejecuta una sola vez y prepara la situación para que se pueda hacer la primera comparación al entrar en el bucle. En este caso (esta instrucción y con este lenguaje de programación), la condición se comprueba **antes** de empezar la primera instrucción de cada vuelta del ciclo.

4.3. Ejercicios

Graba y documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa.

1. Haz un programa que pida 10 números y los vaya sumando.
2. Haz un programa que vaya pidiendo números y los vaya sumando, hasta que nos escriban el número cero.
3. Añade al programa anterior el código para que guarde el mayor número que nos hayan dado hasta el momento.
4. Añade al programa anterior el código para que guarde también cuándo nos han dado el mayor número hasta el momento, es decir, si ha sido el quinto, el sexto o el duodécimo.
5. Haz un programa que vaya pidiendo números y los vaya sumando, hasta que nos escriban el número cero o la suma sobrepase una cierta cantidad (por ejemplo 15).
6. Haz un programa que pida un número, y si es menor que cero nos lo vuelva a pedir, y **que repita este proceso** hasta que nos dé un valor igual o mayor que cero.
7. Pedir un número real. Hacer un ciclo que añada el 35 % a dicho número y lo diga (*say*) hasta que supere el séxtuplo del número inicial. Para ello introduciremos el valor inicial en una nueva variable que iremos multiplicando por 1.35 (añadir el 35 %) mientras no supere el séxtuplo del número inicial.
8. Pide un número y di si es primo, es decir, cuenta el número de divisores (números que lo divicen con resto 0) y si son más de dos entonces no es primo.

9. Modifica el programa anterior para que se pare cuando encuentre el primer divisor empezando a buscar desde el 2. Si ese primer divisor es el propio número, entonces es primo.

10. Calcula el máximo común divisor de dos números.

11. Crea un programa que invierta el orden de las cifras de un número a través de divisiones sucesivas. Si se halla el resto de dividir un número entre 10 tenemos la cifra de las unidades. Si dividimos el número entre 10 obtenemos todas las cifras excepto la de unidades (quizás haya que redondear el número con *round*). Ahora sólo faltaría ir sumando las cifras de unidades a otra variable y multiplicando por 10.

CAPÍTULO

5 Procedimientos

En el capítulo anterior hemos podido comprobar que las cosas se complican cuando los programas tienden a ser más grandes. De la misma forma que este manual tiene una estructura como puedes ver en el índice, un programa también puede hacerse dividiéndolo en unidades funcionales, en partes que desarrollan una utilidad. El programa del juego pregunta datos primero, sigue con el proceso de adivinación, y por último acaba dando una respuesta. Normalmente los programas siguen estos tres pasos: entrada de datos, proceso y salida de resultados.

5.1. Procedimientos como un agrupamiento de instrucciones

Un procedimiento puede ser simplemente una forma de agrupar instrucciones o bloques bajo un único nombre.

Vamos a crear subprogramas o procedimientos basándonos en la derecha de la figura 1.4. Recuerda que los procedimientos son una característica propia de Scratch 2, que no existe en la primera versión. Los pasos son los siguientes:

- Pulsa “Más bloques” (*more blocks*) y haz un nuevo bloque (*make a block*).
- Escribe “cuadrado” y *OK*. Aparecerá un bloque de forma nueva, con la parte de arriba curva, que empieza con la palabra “define” y contiene un bloque con la cadena de caracteres o texto que has escrito, y que equivale a todo el procedimiento. También estará

disponible un bloque con el mismo texto. Ese bloque simple es equivalente a todos los bloques que componen el cuerpo del procedimiento.

- Añade el siguiente código al bloque curvo que encabeza la definición del procedimiento *cuadrado*.

```
Define cuadrado
      Repite 4
            Mueve 50 pasos
            Gira 90 grados
```

Una vez definido el procedimiento *cuadrado*, haremos lo mismo con *iniciaDibujo*, en el que limpiaremos la pantalla, cambiaremos el tamaño del gato al 20 %, iremos a la coordenada de inicio (-100,-100), nos orientaremos hacia la derecha y bajaremos el lápiz. Si haces el programa principal de la siguiente forma, se ejecutarán los dos procedimientos secuencialmente.

```
iniciaDibujo
cuadrado
```

5.2. Procedimientos anidados

Un paso más lejos es crear otro procedimiento que use el ya definido de “cuadrado”. Para eso definimos el subprograma “figura” que repite 8 veces los tres bloques siguientes: cuadrado, mueve 100 pasos y gira 45 grados.

Terminaremos el programa creando un procedimiento llamado “saluda” que levante el lápiz, se mueva 200 pasos y diga algo durante dos segundos.

Prueba a hacerlo hasta conseguir el dibujo siguiente, de la figura 5.2. Apunta qué cambios haces para conseguir el resultado.

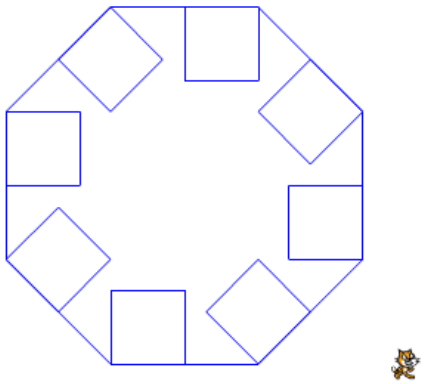


Figura 5.1: Resultado de los procedimientos anidados.

Si no consigues la misma figura, compara con el código de la figura 5.2.

5.3. Procedimientos con parámetros

Los procedimientos son una herramienta muy potente, pero lo que hemos visto hasta ahora tiene el problema de que no podemos cambiar los valores. Sería absurdo tener que definir un procedimiento para los cuadrados, otro para pentágonos, hexágonos... Veamos cómo hacemos polígonos de número variable de lados.

Separa el bloque "define cuadrado" y el bloque "cuadrado" y arrástralos hasta el recuadro de definición de bloques, para destruirlos. Mantenemos el cuerpo de la definición para transformarlo de cuatro lados y 90 grados de giro a x lados y $360/x$ grados.

Figura 5.2: Procedimientos anidados.

Crea un nuevo procedimiento llamado “polígono de” pero no pulses todavía *OK*. Abre el menú de Opciones que tiene el diálogo. Pulsa “Añade una entrada numérica” (*Add number input*) y sustituye “number1” por “lados”. Será el nombre de la variable del número de lados dentro del procedimiento “polígono”. Añade también una etiqueta con *add label text* y escribe “lados”. Nos va a servir como comentario de forma que la llamada al procedimiento va a ser casi una frase: *polígono de 5 lados*.

Arrastra el bloque *define polígono de* lados *lados* y pégale los bloques que antes servían para dibujar el cuadrado. Sustituye el valor 4 por la variable “lados” que aparece en la definición de procedimiento. El giro debe ser el resultado de dividir 360 entre “lados”. Cuando lo hayas terminado compáralo con la figura 5.3.

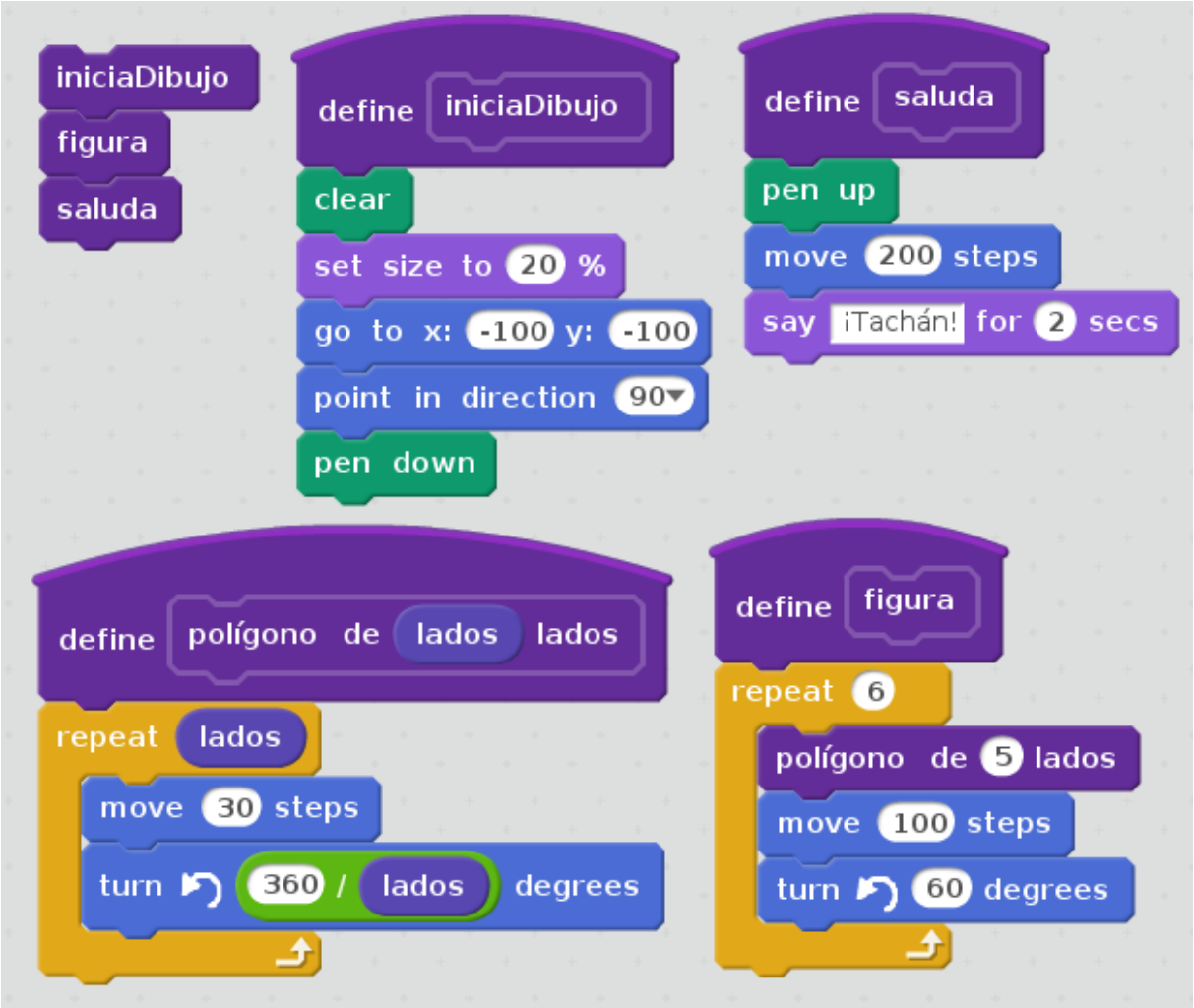


Figura 5.3: Procedimientos anidados con parámetros y etiquetas.

5.4. Funciones

Las funciones son subprogramas que también realizan un conjunto de instrucciones o bloques al igual que los procedimientos, pero en este caso crean un nuevo valor a partir de otros. El ejemplo más sencillo es el de la figura5.4, en el que la función devuelve la media de dos números a través de la variable *media*.

Scratch no tiene un mecanismo propio para crear funciones, por lo que debemos simularlas con una variable que se llame como la propia función. A esta variable le asignaremos el resultado que debe devolver la función. Como hemos dicho, *media* recibirá un 3.

Si usáramos cada vez una variable diferente para dar el resultado, debería cambiar la función. Es más fácil usar la misma variable e ir repartiendo los valores salida a diferentes variables.

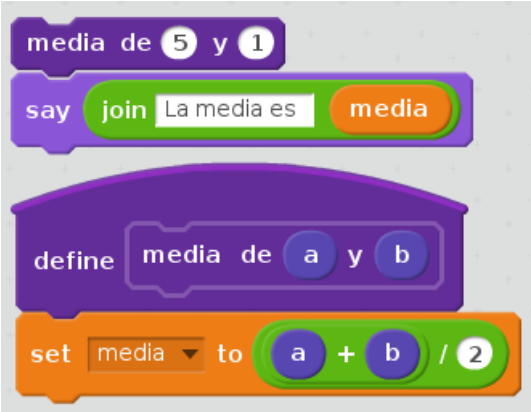


Figura 5.4: Función simple en Scratch 2.

En la figura 5.5 tenemos un ejemplo más complicado, el de calcular el factorial. En este ejemplo además añadimos un bloque nuevo: atiende a un *evento*, cuando se hace clic en la bandera verde. De esa forma, al pulsar la bandera el programa arranca.

5.5. Parámetros por valor y parámetros por referencia

Éste es un concepto clásico en programación. Un *parámetro por valor* es la variable *n* sobre la que calculamos su factorial. No se modifica en el interior de la operación de cálculo del factorial.

Pero piensa en el proceso de intercambiar los valores de dos variables. Si *a* vale 5 y *b* vale 3, queremos que después de ejecutar el procedimiento Si *a* valga 3 y *b* valga 5. Como el procedimiento no sólo queremos usarlo para agrupar bloques y usarlos una vez, tendremos que usar dos variables para hacer el paso de variables o *parámetros por referencia*. En la figura 5.6

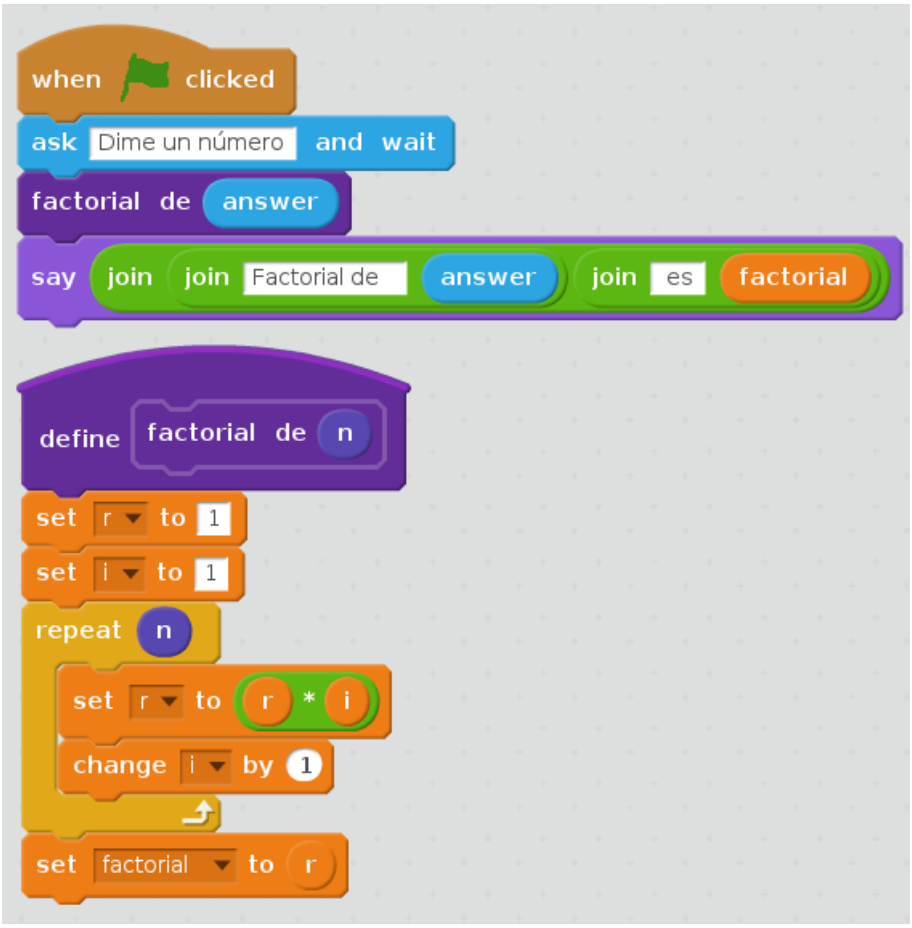


Figura 5.5: Programa que calcula el factorial de un número.

son las variables *j* y *k*. Además, usamos dos variables auxiliares para cambiar los valores dentro del procedimiento. [1]

5.6. Ejercicios

Graba y documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa.

1. Crea un procedimiento que sustituya a “mueve pasos” para dibujar con líneas intermitentes.

2. Crea un procedimiento con el ejercicio que invierte las cifras de un número, y úsalo para saber si es capicúa, si el invertido y el número original son iguales.

[1] La cuestión es que en Scratch las variables son globales a todo el *sprite* (o a todo el programa, según eliges) por lo que no se pueden definir variables locales a un procedimiento, aparte de los parámetros por valor. Es más, Scratch funciona como si cada *sprite* fuera una instancia de un objeto.

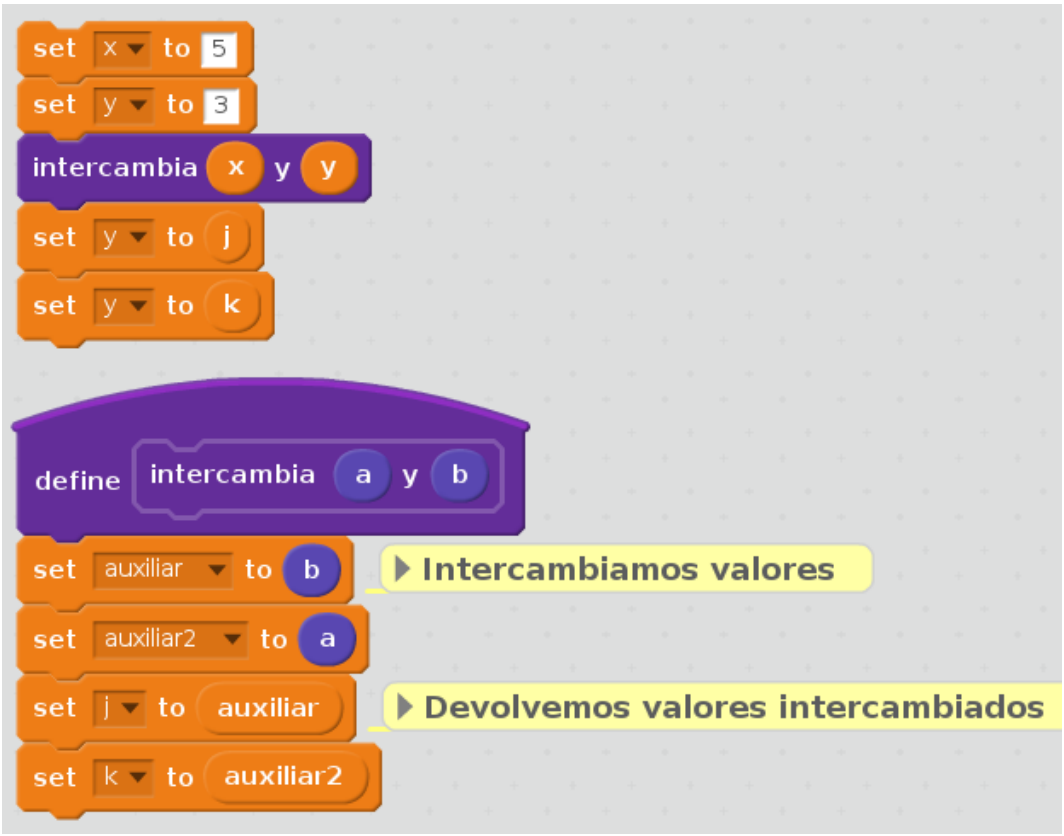


Figura 5.6: Programa que intercambia el valor de las variables x e y.

3. Haz un procedimiento que calcule el valor del número combinatorio n sobre m de la fórmula 5.1.

$$\binom{n}{m} = \frac{n!}{m!\,(n-m)!} \tag{5.1}$$

CAPÍTULO

6 Cadenas de caracteres

Como hemos ido diciendo, las cadenas de caracteres son trozos de texto, y se llaman así porque son en realidad letras, números y símbolos consecutivos. Normalmente, en los lenguajes de programación cada carácter es accesible a través de su lugar en la cadena, es decir, es como el dorsal de cada integrante de un equipo deportivo: podemos referirnos a todo el equipo o a un miembro concreto del grupo.

Para manejar cadenas de caracteres en Scratch hay tres operadores. Ya hemos visto *join*, que une un texto a otro. Además, hay un operador que da la longitud de la cadena (*length of*). Por último, podemos obtener el carácter que se encuentra en una determinada posición de la cadena (*letter of*).

La figura 6.1 es una captura de programa que pide una cadena de caracteres, le da la vuelta de derecha a izquierda y la imprime. Usa las tres instrucciones de manipulación de cadenas.

Con estas instrucciones no se puede modificar una cadena, sólo crear una nueva a partir de otra. Tienes como ejercicio hacer un procedimiento para asignar a una variable el trozo de cadena entre dos posiciones de otra. Por ejemplo, “subcadena de varcad entre 6 y 8” daría “osa” si *varcad* es “ sigilosamente”.

A partir de la anterior también podrías hacer otro procedimiento para copiar una cadena cambiando una única letra de una determinada posición.

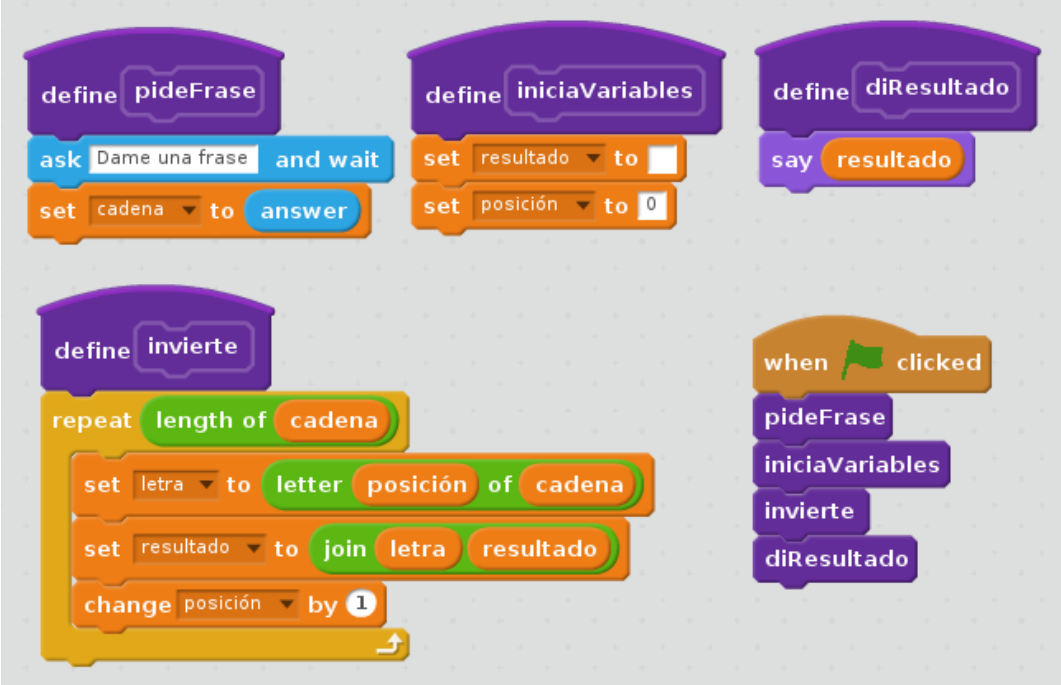


Figura 6.1: Programa que invierte una cadena.

6.1. Ejercicios

Graba y documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa.

1. Cuenta el número de veces que aparece un cierto carácter en una cadena.
2. Comprueba que tiene el número correcto de paréntesis en una operación matemática almacenada en una cadena, tantos abiertos como cerrados.
3. Crea un procedimiento con el ejercicio que invierte la cadena, y úsalo para saber si la cadena es *palíndroma* (el mismo concepto que capicúa pero para texto), si la invertido y la cadena original son iguales.
4. Haz un procedimiento que calcule directamente si es un palíndromo o no, que vaya comparando los caracteres de la posición i e $n - i$ hasta que llegue al centro de la cadena o sean distintos.
5. Haz un procedimiento para asignar a una variable el trozo de cadena entre dos posiciones de otra. Por ejemplo, “subcadena de varcad entre 6 y 8” daría “osa” si *varcad* es “ sigilosamente”.
6. A partir de la anterior también podrías hacer otro procedimiento para copiar una cadena cambiando una única letra de una determinada posición.

CAPÍTULO

7 Vectores

Los vectores son agrupaciones ordenadas de variables de un tipo (aunque en Scratch las variables pueden contener cualquier tipo de dato). Los vectores más sencillos son las tablas unidimensionales de un solo tipo de datos, por ejemplo *n* números enteros. Aunque en Scratch 2 lo que hay disponible es una lista como conjunto numerado, es decir, grupos de datos ordenados que son accesibles por su valor, no vamos a entrar en sus posibilidades.

Para usar las listas de la misma forma que los vectores unidimensionales propios de otros lenguajes de programación, vamos a usar las siguientes instrucciones o bloques de entre los disponibles:

el operador "elemento *i* del vector" (*item i of*) para obtener un valor, y

"reemplazar el elemento *i* del vector con" (*replace item i of vector with*) para cambiar el valor de un elemento del vector.

Además con el bloque *length of* podremos saber el número de elementos del vector.

Con estas operaciones podemos aprender todo tipo de "recetas" para enfrentarnos con problemas reales en matrices.

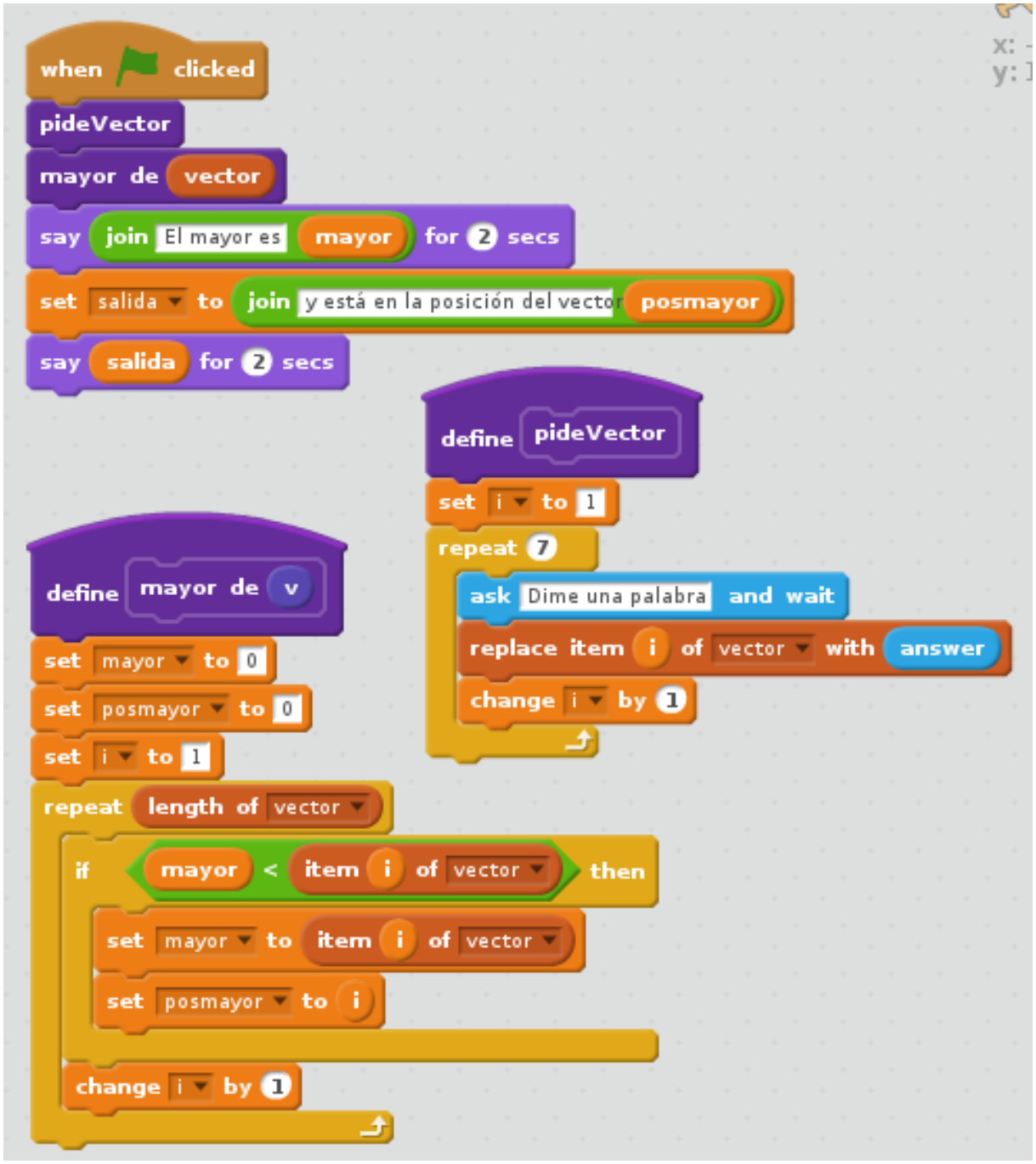


Figura 7.1: Programa que pide un vector y busca el mayor.

7.1. Ejercicios

Graba y documenta los ejercicios haciendo capturas de pantalla o de sólo el código, del programa.

1. Pide un vector y devuélvelo en orden inverso.

2. Haz un procedimiento que duplique el valor de cada posición de un vector.

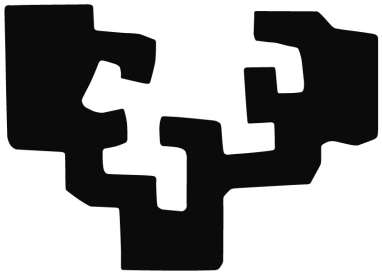
3. Pide dos vectores *distancia* y *giro*, de forma que la posición *i* de ambos definan un paso en el dibujo de una figura. Un cuadrado tendría un vector de longitud 4 con 40 en cada posición (distancia) y otro vector con 4 valores de giro (90).

4. Crea un procedimiento que pide un vector y calcula la media.

5. Modifica el procedimiento anterior para que calcule la media sólo de los números pares.

6. Comprueba que el generador de números aleatorios de Scratch funciona correctamente. Para ello crea un vector que cuente el número de veces que nos da cada valor, almacenando en la posición *i* las veces que nos da el valor *i*.

eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Este manual está en continua modificación