Diez lecciones para descubrir el lenguaje Logo

Loïc Le Coq. Traducción: Álvaro Valdés

http://xlogo.tuxfamily.org

Índice general

1. Descubramos las primitivas básicas **5**
- 1.1. Primeras primitivas que utilizaremos . . . 5
- 1.2. Dibujar un polígono regular . . . 6
  - 1.2.1. El cuadrado . . . 6
  - 1.2.2. El triángulo equilátero . . . 6
  - 1.2.3. El hexágono . . . 7
  - 1.2.4. Trazar un polígono regular en general . . . 8
- 1.3. Empezando con procedimientos . . . 8
- 1.4. Actividad . . . 9

2. Utilizando las coordenadas **10**
- 2.1. Presentación . . . 10
- 2.2. Actividad . . . 11

3. Las variables **12**
- 3.1. Papel de las variables . . . 12
- 3.2. Ejemplos de utilización . . . 12
- 3.3. Trazar un rectángulo de altura y anchura determinadas . . . 13
- 3.4. Trazar una forma con distintos tamaños . . . 14
- 3.5. Actividad . . . 16

4. La recursividad **17**
- 4.1. En el Área de Dibujo . . . 17
  - 4.1.1. Primer ejemplo . . . 17
  - 4.1.2. Tres nuevas primitivas . . . 17
  - 4.1.3. Segundo ejemplo . . . 18
- 4.2. En la zona de texto . . . 18
  - 4.2.1. Un primer ejemplo . . . 18
  - 4.2.2. Realiza un test de salida . . . 18

5. Algunas técnicas de relleno **20**
- 5.1. Primer intento . . . 20
- 5.2. Segundo intento . . . 21

5.3. Tercer intento 22

6. Actividad sobre las cifras de calculadora **23**
6.1. El programa 23
6.2. Creación de una pequeña animación 24

7. Descubrir las listas y las variables **26**
7.1. Interacción con el usuario 26
7.2. Programar un pequeño juego 27

8. Una animación: *El increible monigote creciente* **29**

9. Actividad sobre números primos dos a dos **31**
9.1. Noción de m.c.d. (máximo común divisor) 31
9.2. Algoritmo de Euclides 31
9.3. Calcular un m.c.d. en LOGO 32
9.4. Calcular una aproximación de π 32
9.5. Compliquemos un poco más: π que genera π... 34

10. Actividad sobre la suma de dos dados **37**
10.1. Simular el lanzamiento de un dado 37
10.2. El programa 37

11. Solución a las actividades **42**
11.1. Sección 1.4 42
11.2. Sección 2.2 43
11.3. Sección 3.5 43
11.3.1. El robot 43
11.3.2. La ranita 45
11.4. Sección 7.2 45

Introducción

En este manual encontrará una serie de *recetas* que le ayudarán a aprender el lenguaje Logo.

Soy profesor de matemáticas, y esta guía es una pequeña compilación de las distintas actividades realizadas en clase con los alumnos de clase de cuarto y quinto. Cada sección intenta abordar un concepto particular del lenguaje y propone actividades que deben realizarse, o bien como descubrimiento o como autoevaluación (en este caso, están disponibles algunas soluciones al final del tutorial).

Este manual no pretende ser EL tutorial ideal para comprender Logo. Se limita a proponer distintas pistas, distintos enfoques sobre algunos aspectos de la programación en Logo. Las actividades son de niveles variados y, creo, representan un buen conjunto de experiencias de lo que es capaz de realizar el lenguaje Logo. ¡Espero que las explicaciones aportadas sean lo más claras posibles! No duden en hacerme llegar sus comentarios y sus críticas respecto a este tutorial. Y ahora, ¡a iniciarse a las alegrías de la pequeña tortuga!

Capítulo 1

Descubramos las primitivas básicas

Para desplazar a la tortuga sobre la pantalla, se utilizan órdenes predefinidas llamadas ”primitivas”. En esta primera sección, se van a describir las primitivas básicas que permiten controlar a la tortuga.

1.1. Primeras primitivas que utilizaremos

- `avanza número` `av 50`

  Hace avanzar a la tortuga el número de pasos indicado

- `retrocede número` `re 100`

  Hace retroceder a la tortuga el número de pasos indicado

- `giraderecha número` `gd 90`

  La tortuga gira a la derecha el ángulo indicado

- `giraizquierda número` `gi 90`

  La tortuga gira a la izquierda el ángulo indicado

- `borrapantalla` `bp`

  Borra la pantalla y devuelve a la tortuga al centro de la pantalla (vuelve al origen)

- `muestratotuga` `mt`

  Hace visible a la tortuga en la pantalla

- `ocultatortuga` `ot`

  Oculta la tortuga. Permite una visualización mas rápida.

- `subelapiz` `sl`

  Sube el ”lápiz”. La tortuga no deja trazo tras ella (no dibuja) cuando se desplaza.

- `bajalapiz` `bl`

  Baja el lápiz. La tortuga escribe cuando se desplaza.

- `repite número lista` `repite 5 [av 50 gd 45]`

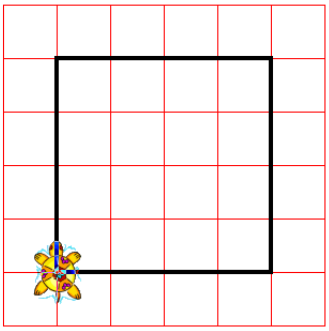
  Repite las instrucciones contenidas en la lista el número de veces indicado

1.2. Dibujar un polígono regular

Vamos a aprender a dibujar un cuadrado, un triángulo equilátero, un pentágono regular, etc.

1.2.1. El cuadrado

En el dibujo siguiente, un cuadrado representa 50 "pasos" de tortuga.



Para dibujar el cuadrado de la figura, se debe escribir:

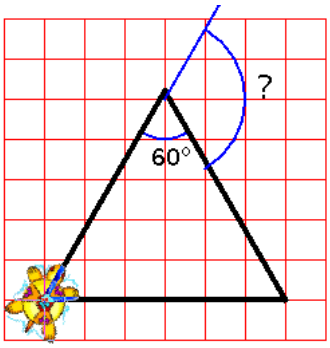
```
avanza 200 giraderecha 90
avanza 200 giraderecha 90
avanza 200 giraderecha 90
avanza 200 giraderecha 90
```

Date cuenta que se repite 4 veces la misma instrucción, así que podemos usar una sintaxis más rápida:

```
repite 4
  [ av 200 gd 90 ]
```

1.2.2. El triángulo equilátero

Vamos a ver cómo trazar este triángulo equilátero de 180 pasos de tortuga:



Aquí, un cuadrado representa 30 pasos de tortuga

Las órdenes serán algo del estilo:

```
repite 3
  [ avanza 180 giraderecha ... ]
```

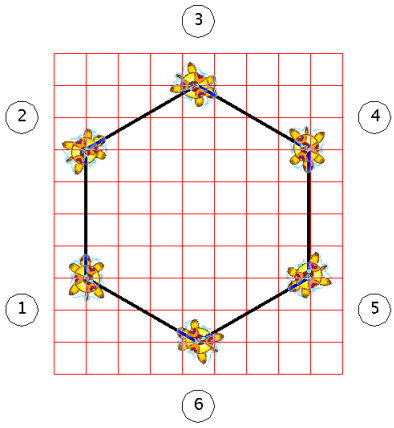
Queda por determinar el ángulo correcto. En un triángulo equilátero, los ángulos valen todos 60 grados, y como la tortuga debe volver por el exterior del triángulo, el ángulo valdrá:

```
180 - 60 = 120 grados
```

Las órdenes son, pues:

```
repite 3
  [ av 180 gd 120 ]
```

1.2.3. El hexágono



Un cuadrado = 20 pasos de tortuga.

Para un cuadrado repetíamos 4 veces, para el triángulo 3 veces, parece claro que para el hexágono será:

```
repite 6
  [ avanza 80 giraderecha ... ]
```

Date cuenta que en su desplazamiento, la tortuga realmente da una vuelta completa sobre ella misma. (Inicialmente está orientada hacia arriba y termina en esta misma posición). Esta rotación de 360 grados se efectúa en 6 etapas. Por tanto, cada vez, gira

```
360/6 = 60 grados
```

Las órdenes son, entonces:

```
repite 6
  [ av 80 gd 60 ]
```

1.2.4. Trazar un polígono regular en general

En realidad, reiterando el razonamiento anterior, puedes darte cuenta de que para trazar un polígono de `n` lados, el ángulo se obtendrá dividiendo `360` por `n`. Por ejemplo:

- Para trazar un pentágono regular de lado `100`: `(360:5=72)`

  ```
  repite 5 [ av 100 gd 72 ]
  ```

- Para trazar un eneágono regular de lado `20`: `(360:9=40)`

  ```
  repite 9 [ av 20 gd 40 ]
  ```

- Para trazar un eh ... `360`-gono[1] regular de lado `2` (que se parece mucho a un círculo):

  ```
  repite 360 [ av 2 gd 1 ]
  ```

- Para trazar un heptágono regular de lado `120`:

  ```
  repite 7 [ av 120 gd 360/7 ]
  ```

1.3. Empezando con procedimientos

Para evitar tener que repetir cada vez las instrucciones para dibujar un cuadrado, un triángulo ..., puedes definir instrucciones personalizadas llamadas "procedimientos". Un procedimiento comienza con la palabra clave `para` y termina con la palabra clave `fin`. Por ejemplo, abre el editor, escribe:

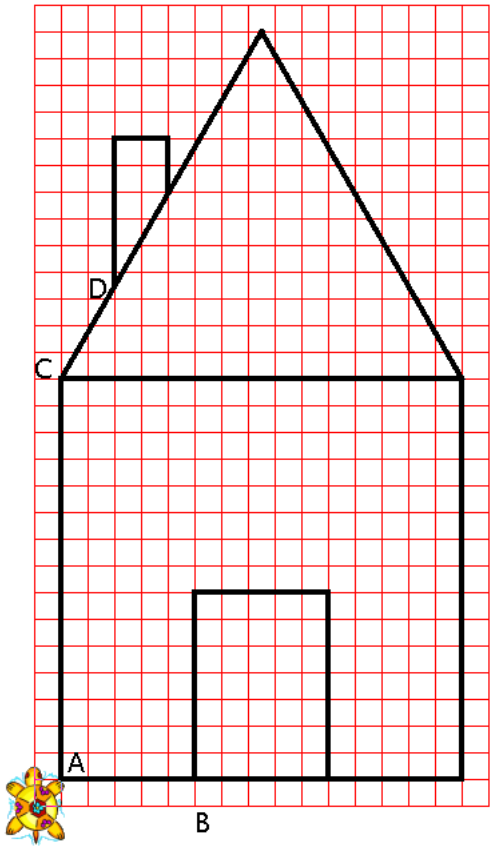
```
para cuadrado
  repite 4
    [ avanza 100 giraderecha 90 ]
fin
```

luego cierra al editor guardando los cambios haciendo *click* en el pingüino. Ahora cada vez que se escriba `cuadrado`, ¡un cuadrado aparecerá en la pantalla!

[1] Trihectahexacontágono: tri = 3, hecta = 100, hexaconta = 60, gono = ángulo

1.4. Actividad

Debes conseguir el dibujo que se muestra a continuación.



Cada cuadrado vale 10 pasos de tortuga.

Para ello, deberás definir ocho procedimientos:

- Un procedimiento "`cuadrado`" que trazará el cuadrado básico de la casa
- Un procedimiento "`tri`" que trazará el triángulo equilátero que representa el tejado
- Un procedimiento "`puerta`" que trazará el rectángulo que representa la puerta
- Un procedimiento "`chi`" que trazará la chimenea
- Un procedimiento "`desp1`" que desplazará la tortuga de la posición A a la B
- Un procedimiento "`desp2`" que llevará a la tortuga desde la posición B a la C
- Un procedimiento "`desp3`" que hará a la tortuga ir de la posición C a la D
- Un procedimiento "`casa`" que trazará la casa en su totalidad ayudándose de todos los demás procedimientos

Capítulo 2

Utilizando las coordenadas

2.1. Presentación

En esta sección, vamos a descubrir la primitiva `ponposicion`.

El **Área de Dibujo** es un sistema cartesiano cuyo origen está situado en el centro de la pantalla. De este modo, podemos alcanzar cualquiera de los puntos de dicho área ayudados por sus coordenadas.

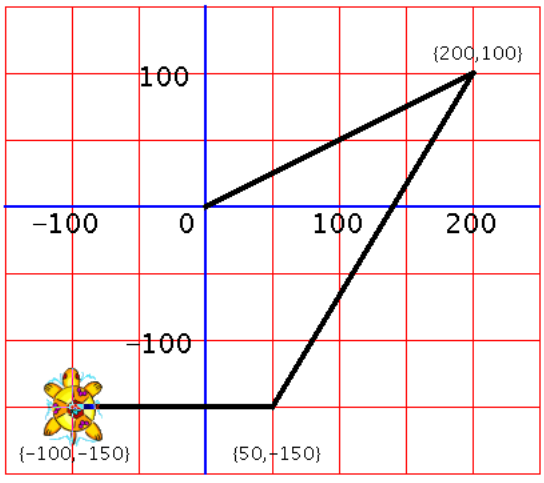
La primitiva que vamos a utilizar es:

`ponposicion lista` `ponpos [100 -250]`

que desplaza la tortuga al punto cuyas coordenadas son las definidas en la lista.

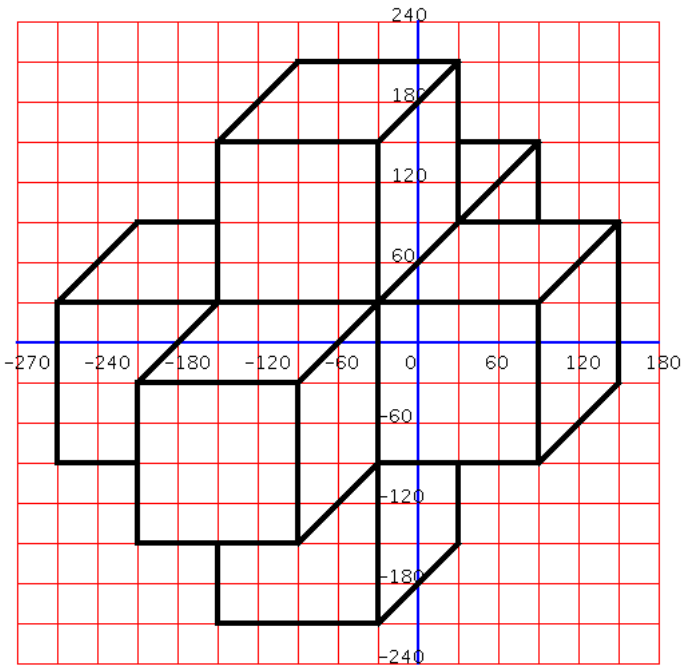
Un ejemplo sencillo de su utilización:

```
borrapantalla
ponpos [200 100]
ponpos [50 -150]
ponpos [-100 -150]
```



2.2. Actividad

En esta actividad debes realizar la siguiente figura.



Sólo puedes utilizar las primitivas

- `ponposicion` (`ponpos`)
- `borrapantalla` (`bp`)
- `subelapiz` (`sl`)
- `bajalapiz` (`bl`)

Capítulo 3

Las variables

3.1. Papel de las variables

A veces, se desearía dibujar una misma forma pero con dimensiones diferentes. Por ejemplo, si se desea dibujar un cuadrado de lado 100, un cuadrado de lado 200 y un cuadrado de lado 50, con lo que sabemos hasta ahora deberíamos definir tres procedimientos diferentes, uno para cada uno de estos cuadrados. Sería más simple definir un único procedimiento al que se pasara como parámetro la longitud deseada del lado. Por ejemplo, `cuadrado 200` trazaría el cuadrado de lado 200, `cuadrado 100` trazaría el cuadrado de lado 100, etc. Eso es precisamente lo que las variables nos van a permitir realizar.

3.2. Ejemplos de utilización

Para dibujar un cuadrado de lado `100`, habíamos escrito:

```
para cuadrado
  repite 4
    [ avanza 100 giraderecha 90]
fin
```

Vamos a modificar este procedimiento para que reciba un parámetro (o dicho de forma elegante, un "argumento") que indique la longitud del lado del cuadrado a dibujar.
Una variable siempre va precedida del símbolo ":". Si queremos indicar al procedimiento `cuadrado` que dependa de la variable `:c`, se añade a la línea de definición la ruta `:c`.

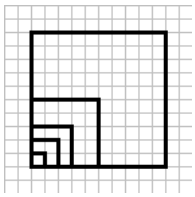
En consecuencia, no debemos indicar a la tortuga que avance `100` pasos, sino `:c` pasos. El procedimiento resultante es:

```
para cuadrado :c
  repite 4
    [ avanza :c giraderecha 90]
fin
```

Así, escribiendo:

```
bp ot cuadrado 100 cuadrado 50 cuadrado 30 cuadrado 20 cuadrado 10
```

Obtenemos la figura siguiente:



3.3. Trazar un rectángulo de altura y anchura determinadas

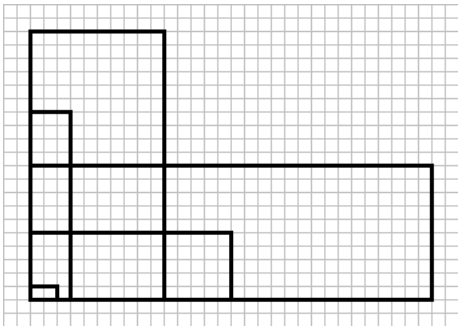
Definamos ahora un procedimiento llamado `rectangulo` que dependerá de dos variables. Por ejemplo, `rec 200 100` trazará un rectángulo de altura `200` y anchura `100`. Para ello:

```
para rectangulo :alto :ancho
  repite 2
    [ avanza :alto giraderecha 90 avanza :ancho giraderecha 90 ]
fin
```

Hechos los cambios:

```
borrapantalla ocultatortuga
rectangulo 200 100 rectangulo 100 300
rectangulo  50 150 rectangulo  10  20
rectangulo 140  30
```

genera:



Ahora bien, si no se proporciona alguno de los argumentos al procedimiento `rectangulo`, el intérprete nos indicará con un mensaje de error que el procedimiento necesita otro argumento:

```
No hay suficientes datos para rectangulo
```

3.4. Trazar una forma con distintos tamaños

Veamos como trazar un cuadrado y un rectángulo con dos tamaños distintos. Ahora volvamos al ejemplo de la casa de la sección 1.4 y cómo modificar el código para dibujar la casa sin que importen las dimensiones.

El objetivo es pasar un argumento al procedimiento `casa` para que, según el parámetro, la casa sea más o menos grande. Has de notar que:

- `casa 10` dibujará la casa de la sección 1.4.
- `casa 5` dibujará la casa a escala 0,5.
- `casa 20` dibujará una casa con las dimensiones dos veces más grandes

La noción de proporcionalidad se da por conocida. Según el dibujo de la sección 1.4, un cuadrado representa 10 pasos. El procedimiento `cuadrado` era el siguiente:

```
para cuadrado
  repite 4
    [ avanza 150 giraderecha 90 ]
fin
```

que ahora se va a convertir en:

```
para cuadrado :c
  repite 4
    [ avanza :c giraderecha 90 ]
fin
```

Así, cuando se escriba `cuadrado 10`, el cuadrado tendá un lado igual a `15 * 10 = 150`. ¡Las proporciones se mantienen correctamente! De hecho, hay que darse cuenta de que va a ser necesario reescribir todos los procedimientos y cambiar las longitudes de desplazamiento de la siguiente manera.

- `70` se convertirá en `7 * :c`
- `av 45` se convertirá en `av 4.5 * :c`
- etc.

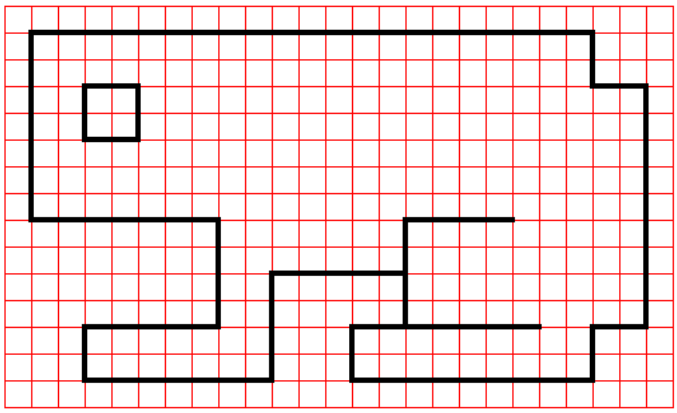
Eso hace que, en realidad, ¡sólamente haya que contar el número de cuadrados para cada longitud! Se obtiene:

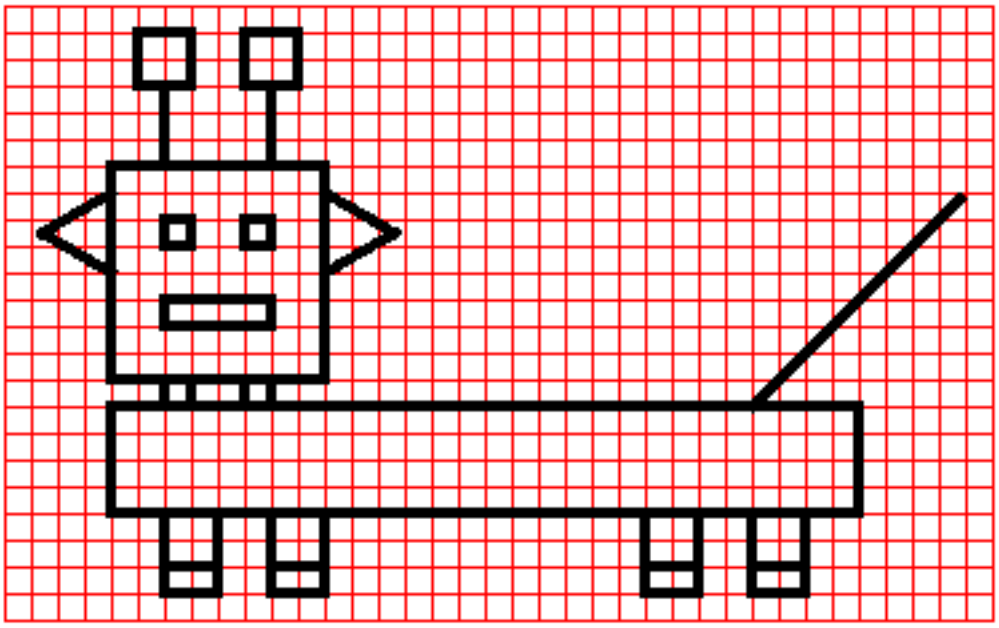
```
para cuadrado :c
  repite 4
    [ avanza 15*:c giraderecha 90 ]
fin
```

```
para tri :c
  repite 3
    [ avanza 15*:c giraderecha 120 ]
fin

para puerta :c
  repite 2
    [ avanza 7*:c giraderecha 90
      avanza 5*:c giraderecha 90 ]
fin

para chi :c
  avanza 5.5*:c giraderecha 90
  avanza 2*:c giraderecha 90
  avanza 2*:c
fin

para desp1 :c
  subelapiz
  giraderecha 90 avanza 5*:c
  giraizquierda 90
  bajalapiz
fin

para desp2 :c
  subelapiz
  giraizquierda 90 avanza 5*:c
  giraderecha 90 avanza 15*:c
  giraderecha 30
  bajalapiz
fin

para desp3 :c
  subelapiz
  giraderecha 60 avanza 2*:c
  giraizquierda 90 avanza 3.5*:c
  bajalapiz
fin

para casa :c
  cuadrado :c desp1 :c puerta :c desp2 :c tri :c desp3 :c chi :c
fin
```

3.5. Actividad

Realiza los siguientes dibujos con dos variables de modo que sea posible obtenerlos a distintos tamaños:





Capítulo 4

La recursividad

Se dice que un procedimiento es recursivo si se llama a sí mismo. Veamos algunos ejemplos que utilizan esta propiedad.

4.1. En el Área de Dibujo

4.1.1. Primer ejemplo

Escribe el siguiente procedimiento en el Editor:

```
para ejemplo1
  giraderecha 1
  ejemplo1
fin
```

Este procedimiento es recurrente puesto que se ejecuta la orden `ejemplo1` en la última linea. En la ejecución, se comprueba que la tortuga no deja de girar sobre sí misma. Para parar el programa, nos vemos obligados a hacer click en el boton `Alto`.

4.1.2. Tres nuevas primitivas

- `espera número` `espera 60`

  Hace una pausa igual a la 60.ª parte del número indicado.

  Por ejemplo, `espera 120` deja el programa parado durante dos segundos

- `goma` `go`

  Cuando la tortuga se desplaza, borra todo a su paso en vez de dejar un rastro tras ella

- `ponlapiz` `pla`

  Devuelve al modo de dibujo habitual: la tortuga deja un rastro tras ella en su desplazamiento

4.1.3. Segundo ejemplo

Piensa qué hace este programa:

```
para ejemplo2
  av 200 goma espera 60
  re 200 ponlapiz gd 6
  ejemplo2
fin
```

Inícialo con la orden: `bp ejemplo2` ¡La aguja segundera de un reloj!

4.2. En la zona de texto

4.2.1. Un primer ejemplo

Escribe sucesivamente

```
escribe "Hola, escribe "Hola, escribe [Yo escribo lo que yo veo]
```

Espero que a estas alturas ya conozcas y controles la primitiva `escribe` o `es`. No olvides las comillas "" cuando quieras escribir una palabra.

Estudia este ejemplo:

```
para ejemplo3 :n
  escribe :n
  ejemplo3 :n+1
fin
```

- Inicíalo con la orden `ejemplo3 0`
- Haz los cambios necesarios en este programa para que las cifras aparezcan de dos en dos
- Páralo con el botón `Alto`

Quiero ahora mostrar todas los números mayores que 100 y que son múltiplos de cinco. ¿Qué debo hacer en el programa? ¿Qué debo escribir para iniciarlo?

4.2.2. Realiza un test de salida

Escribe las órdenes siguientes:

```
si 2+1=3 [escribe [Esto es verdad] ]
si 2+1=4 [escribe [Esto es verdad] ] [escribe [El calculo es mentira] ]
si 2+5=7 [escribe "Verdad] [escribe "Mentira]
```

Si todavía no entiendes la sintaxis de la primitiva `si`, echa un vistazo al manual de referencia de XLogo.

```
para ejemplo4 :n
   si :n = 100 [alto]
   escribe :n
   ejemplo4 :n+1
fin
```

- Lanza el programa `ejemplo4 0`
- Haz los cambios necesarios en el programa para que aparezcan los números comprendidos entre 55 y 350 que son múltiplos de 11.

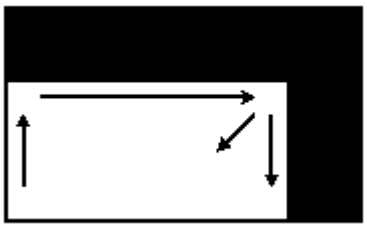
Capítulo 5

Algunas técnicas de relleno

En esta lección, vamos a ver como se puede proceder para rellenar un rectángulo de longitud y anchura determinada. En los ejemplos siguientes trabajaremos con un rectángulo de 100 por 200.

5.1. Primer intento

Si, por ejemplo, se desea trazar un rectángulo sólido de 100 por 200, una primera idea puede ser dibujar el rectángulo de 100 por 200: luego trazar un rectángulo de 99 por 199; luego un rectángulo de 98 por 198 ... hasta que el rectángulo quede coloreado completamente.



Empecemos por definir un rectángulo cuya altura y anchura dependan de dos variables:

```
para rectangulo :alto :ancho
  repite 2
    [ avanza :alto  giraderecha 90
      avanza :ancho giraderecha 90 ]
fin
```

Para rellenar nuestro rectángulo, vamos a ejecutar:

```
rectangulo 100 200 rectangulo 99 199 rectangulo 98 198 ... rectangulo 1 101
```

algo que conseguimos de forma recursiva haciendo:

```
para rectangulo :alto :ancho
   [ avanza :alto  giraderecha 90
     avanza :ancho giraderecha 90 ]
   rectangulo :alto-1 :ancho-1
fin
```

Probando `rectangulo 100 200` nos encontramos con un problema: el procedimiento no se detiene cuando el rectángulo se ha rellenado por completo, sino que ¡continúa dibujando rectángulos!

Vamos a añadir un condicional que permita detectar cuando la altura o anchura son nulas. En ese momento, pediremos al programa que se detenga con la primitiva `alto`:

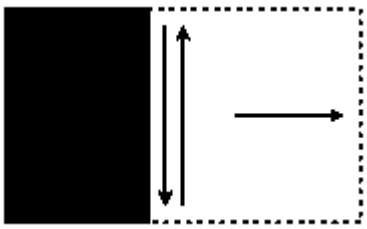
```
para rectangulo :alto :ancho
  si o :alto=0 :ancho=0 [alto]
  [ avanza :alto  giraderecha 90
    avanza :ancho giraderecha 90 ]
  rectangulo :alto-1 :ancho-1
fin
```

Nota: En vez de utilizar la primitiva `o`, se puede utilizar el símbolo "|", y escribiríamos:

```
si :alto=0 | :ancho=0 [alto]
```

5.2. Segundo intento

La idea aquí va a ser que la tortuga empiece avanzando `100` pasos, despues retroceda esos mismos 100 pasos; se desplace un poco a la derecha y repita este movimiento, que llamaremos "elemental" hasta que el rectángulo esté completamente coloreado.



Si la altura del rectángulo está representada por la variable `:alto`, ese será el movimiento elemental a repetir:

```
avanza :alto retrocede :alto giraderecha 90 avanza 1 giraizquierda 90
```

Este movimiento debe repetirse `:ancho` veces. El procedimiento final será:

```
para rectangulo :alto :ancho
  repite :ancho
    [ avanza :alto retrocede :alto
      giraderecha 90 avanza 1 giraizquierda 90 ]
fin
```

Nota: si no se indica el primer trazo vertical en el bucle, hará un pequeño trazo de longitud un paso en la base del rectángulo.

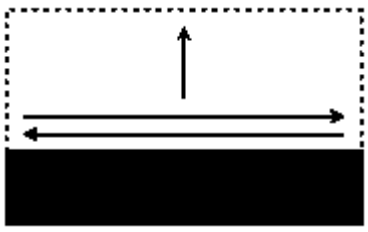
Otro intento habría podido utilizar la recursividad y un condicional como prueba de alcanzar el final.

```
para rectangulo :alto :ancho
 si :ancho = 0 [alto]
 repite :ancho
   [ avanza :alto retrocede :alto
     giraderecha 90 avanza 1 giraizquierda 90 ]
fin
```

Nota: A cada trazo vertical dibujado, disminuye la variable `:ancho` en una unidad. Así pues, cuando vale `0`, se habrá dibujado el rectángulo completo.

5.3. Tercer intento

Efectuaremos los mismos movimientos que en la sección anterior, pero trazando consecutivamente líneas horizontales:

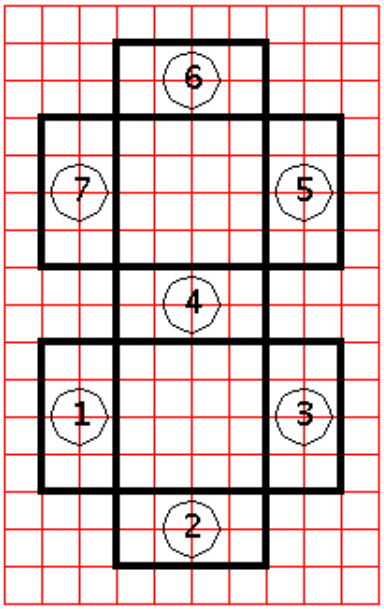


```
para rectangulo :alto :ancho
  repite :alto
    [ avanza :ancho retrocede :ancho
      giraderecha 90 avanza 1 giraizquierda 9 ]
fin
```

Capítulo 6

Actividad sobre las cifras de calculadora

Esta actividad se basa en el hecho de que todos los números de calculadora pueden obtenerse con ayuda de un patrón sí–no:



- para dibujar un ”4”, ”encenderemos” los rectángulos 3, 4, 5 y 7, pero no los 1, 2 y 6
- para dibujar un ”8”, ”encenderemos” los rectángulos 1, 2, 3, 4, 5, 6 y 7.
- para dibujar un ”3”, encenderemos los rectángulos 2, 3, 4, 5 y 6, pero no los 1 y 7

6.1. El programa

Nos basaremos en el rectángulo sólido de la sección anterior:

```
para rectangulo :alto :ancho
  si :ancho = 0 | :alto =0 [alto]
  repite 2
    [ avanza :alto giraderecha 90
      avanza :ancho giraderecha 90]
  rectangulo :alto-1 :ancho-1
fin
```

Supondremos aquí que la tortuga parte de la esquina inferior izquierda. Vamos a definir un procedimiento llamado `cifra` que lee 7 argumentos `:a`, `:b`, `:c`, `:d`, `:e`, `:f` y `:g`.

- Cuando `:a` vale `1`, dibuja el rectángulo 1. Si `:a` vale `0`, no se dibuja.
- Cuando `:b` vale `1`, dibuja el rectángulo 2. Si `:b` vale `0`, no se dibuja.
- Cuando `:c` vale `1`, dibuja el rectángulo 1. Si `:c` vale `0`, no se dibuja.
- ...

Se obtiene el siguiente procedimiento:

```
para cifra :a :b :c :d :e :f :g
# Dibujamos el rectangulo 1
  si :a=1 [rectangulo 160 40]
# Dibujamos el rectangulo 2
  si :b=1 [rectangulo 40 160]
  sl gd 90 av 120 gi 90 bl
# Dibujamos el rectangulo 3
  si :c=1 [rectangulo 160 40]
  sl av 120 bl
# Dibujamos el rectangulo 5
  si :e=1 [rectangulo 160 40]
# Dibujamos el rectangulo 4
  gi 90 sl re 40 bl
  si :d=1 [rectangulo 160 40]
# Dibujamos el rectangulo 6
  gd 90 sl av 120 gi 90 bl
  si :f=1 [rectangulo 160 40]
# Dibujamos el rectangulo 7
  sl av 120 gi 90 re 40 bl
  si :g=1 [rectangulo 160 40]
fin
```

6.2. Creación de una pequeña animación

Vamos a simular una cuenta atrás, que consiste en hacer aparecer sucesivamente las cifras de `9` a `0` en orden decreciente.

```
para cuentatras
   bp ot cifra 0 1 1 1 1 1 1 espera 60
   bp ot cifra 1 1 1 1 1 1 1 espera 60
   bp ot cifra 0 0 1 0 1 1 0 espera 60
   bp ot cifra 1 1 1 1 0 1 1 espera 60
   bp ot cifra 0 1 1 1 0 1 1 espera 60
   bp ot cifra 0 0 1 1 1 0 1 espera 60
```

```
   bp ot cifra 0 1 1 1 1 1 0 espera 60
   bp ot cifra 1 1 0 1 1 1 0 espera 60
   bp ot cifra 0 0 1 0 1 0 0 espera 60
   bp ot cifra 1 1 1 0 1 1 1 espera 60
fin
```

Pequeño problema: hay un efecto de parpadeo desagradable durante la creación de cada cifra. Para suavizarlo se van a utilizar las primitivas `animacion` y `refrescar`. `animacion` hace que la tortuga dibuje pero no lo muestre, es decir, a nuestros ojos no hace nada; al recibir la orden `refrescar` muestra todo el trabajo almacenado en memoria.

Para salir del modo *animación* y volver al modo *normal* tenemos dos opciones:

- Usar la primitiva `detieneanimacion`
- Hacer *click* en la cámara de cine que aparece a la izquierda del Histórico de Comandos.

Se obtiene el siguiente programa modificado:

```
para cuentatras
# Entramos en modo animacion
  animacion
  bp ot cifra 0 1 1 1 1 1 1 refrescar espera 60
  bp ot cifra 1 1 1 1 1 1 1 refrescar espera 60
  bp ot cifra 0 0 1 0 1 1 0 refrescar espera 60
  bp ot cifra 1 1 1 1 0 1 1 refrescar espera 60
  bp ot cifra 0 1 1 1 0 1 1 refrescar espera 60
  bp ot cifra 0 0 1 1 1 0 1 refrescar espera 60
  bp ot cifra 0 1 1 1 1 1 0 refrescar espera 60
  bp ot cifra 1 1 0 1 1 1 0 refrescar espera 60
  bp ot cifra 0 0 1 0 1 0 0 refrescar espera 60
  bp ot cifra 1 1 1 0 1 1 1 refrescar espera 60
# Volvemos al modo de dibujo habitual
  detieneanimacion
fin
```

Capítulo 7

Descubrir las listas y las variables

7.1. Interacción con el usuario

Vamos a realizar un pequeño programa que pide al usuario su nombre, sus apellidos y su edad. Al final del cuestionario, el programa responde con una lista del estilo:

```
Tu nombre es: ........
Tus apellidos son: ........
Tu edad es: ........
Tu eres mayor o menor
```

Para eso, vamos a utilizar las primitivas siguientes:

- `leelista:` `leelista [¿Cuántos años tienes?] "a`

  Presenta una caja de diálogo con el título indicado en la lista (en este caso, titulada `¿Cuántos años tienes?`). El usuario da la respuesta y se guarda en forma de lista en la variable `:a`. Por ejemplo, si introducimos 20, la variable `:a` contendrá `[20]`.

- `haz:` `haz "a 30`

  Asigna el valor 30 a la variable `:a`

- `frase, fr:` `frase [30 k] "a`

  Incorpora un valor a una lista. Si este valor es una lista, acopla las dos listas.

  ```
  frase [30 k] "a              —>  [30 k a]
  frase [1 2 3] 4              —>  [1 2 3 4]
  frase frase [1 2 3] [4 5 6]  —>  [1 2 3 4 5 6]
  ```

- `primero,` `pr:`

  Devuelve el primer valor de una lista

  ```
  escribe primero [4 5 6]          —>  4
  escribe primero [Como te va 8]   —>  Como
  ```

Obtenemos el código siguiente:

```
para cuestion
leelista [¿Cuantos años tienes?] "edad
    # :edad contiene ahora una lista con un elemento;
    # extraemos ese elemento y lo guardamos de nuevo en :edad
haz "edad primero :edad
leelista [¿Cómo te llamas?] "nombre
leelista [¿Cómo te apellidas?] "apellidos
escribe frase [Te llamas: ] :nombre
escribe frase [Te apellidas: ] :apellidos
escribe frase [Tu edad es: ] :edad
si o :edad>18 :edad=18
[escribe [Eres mayor de edad]] [escribe [Eres menor de edad]]
fin
```

7.2. Programar un pequeño juego

El objetivo de este apartado consiste en crear el siguiente juego:

- El programa elige un número al azar entre 0 y 32 y lo memoriza.
- Se abre una caja de diálogo y pide un número al usuario.
- Si el número introducido es igual al número memorizado, devuelve "`¡Acertaste!`" en la zona de texto.
- En el caso contrario, el programa indica si el número memorizado es `más grande` o `más pequeño` que el número propuesto por el usuario
- Luego abre de nuevo la caja de diálogo.
- El programa se termina cuando el usuario ha encontrado el número memorizado.

Necesitarás utilizar la primitiva siguiente:

`azar`: `azar 20`
Devuelve un número aleatorio entre 0 y 19.

VEAMOS ALGUNAS REGLAS QUE DEBES RESPETAR PARA REALIZAR ESTE JUEGO:

1. El número se guardará en la memoria del ordenador en una variable llamada `numero`.
2. La caja de diálogo tendra por título: "`Propón un número:`".
3. El número introducido por el usuario será guardado en una variable llamada `prueba`.

4. El procedimiento que inicia el juego se lanzará con la palabra `juego`.

ALGUNAS POSIBLES MEJORAS:

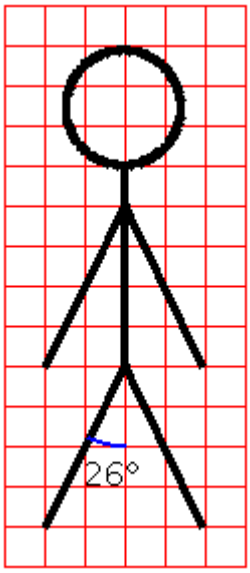
- Limitar el número de intentos
- El número a adivinar debe estar comprendido entre 0 y 2000.
- Comprobar si el usuario introduce realmente un número o está introduciendo una lista o una palabra. Para ello, utilizar la primitiva `numero?`

  Ejemplos:

```
numero? 8                 es cierto.
numero? [5 6 7]           es falso. ([5 6 7] es una lista)
numero? "abcde            es falso ("abcde es una palabra)
```

Capítulo 8

Una animación: *El increible monigote creciente*



En primer lugar, vamos a definir un procedimiento `monigote` que dibujará el monigote representado arriba, con un tamaño de nuestra elección.

```
para monigote :c
   gi 154 av 2.2*:c re :c*2.2
   gi 52 av 2.2*:c re :c*2.2
   gi 154 av :c*2
   gi 154 av 2.2*:c re :c*2.2
   gi 52 av 2.2*:c re :c*2.2
   gi 154 av :c/2
   gi 90 repite 180[av :c/40 gd 2] gd 90
fin
```

Vamos ahora con la animación que creará la ilusión de que el monigote crece poco a poco. Para ello, escribimos `monigote 1`, despues `monigote 2` `monigote 3` ... hasta `monigote 75`. Entre cada trazado, se borrará la pantalla. Se obtienen los dos procedimientos siguientes:

```
para monigote :c
  si :c=75 [alto]
  gi 154 av 2.2*:c re :c*2.2
  gi 52 av 2.2*:c re :c*2.2
  gi 154 av :c*2
  gi 154 av 2.2*:c re :c*2.2
  gi 52 av 2.2*:c re :c*2.2
  gi 154 av :c/2
  gi 90 repite 180 [av :c/40 gd 2] gd 90
  bp ot monigote :c+1
fin

para empezar
   bp ot monigote 0
fin
```

Por último, para suavizar todo el proceso, vamos a servirnos del modo `animacion` y de la primitiva `refrescar`.

```
para monigote :c
  si :c=75 [alto]
  gi 154 av 2.2*:c re :c*2.2
  gi 52 av 2.2*:c re :c*2.2
  gi 154 av :c*2
  gi 154 av 2.2*:c re :c*2.2
  gi 52 av 2.2*:c re :c*2.2
  gi 154 av :c/2
  gi 90 repite 180 [av :c/40 gd 2] gd 90
  refrescar
  bp ot monigote :c+1
fin

para empezar
   bp ot animacion
   monigote 0
   detieneanimacion
fin
```

Capítulo 9

Actividad sobre números primos dos a dos

AVISO: Son necesarios algunos conceptos de matemáticas para comprender bien esta sección.

9.1. Noción de m.c.d. (máximo común divisor)

Dados dos números enteros, el máximo común divisor define al mayor divisor común de ambos. Por ejemplo:

- 42 y 28 tienen como m.c.d. 14, ya que es el número más grande por el que es posible dividir a la vez 28 y 42
- el m.c.d. de 25 y 55 es 5
- 42 y 23 tienen m.c.d. igual a 1

Cuando dos números tienen m.c.d. 1, se dice que son primos entre sí. Así en el ejemplo anterior, 42 y 23 son primos entre sí. Eso significa que no tienen ningún divisor común excepto 1 (¡por supuesto, hablamos de división entera!).

9.2. Algoritmo de Euclides

Para determinar del m.c.d. de dos números, se puede utilizar un método llamado **algoritmo de Euclides**: (Aquí no se demostrará la validez de este algoritmo, se admite que funciona)

He aquí el principio: "dados dos enteros positivos `a` y `b`, se comienza por probar si `b` es nulo. En caso afirmativo, entonces el m.c.d. es igual a `a`. Si no, se calcula `r`, el resto de la división de `a` por `b`. Se sustituye `a` por `b`, y `b` por `r`, y se reinicia el método.

Calculemos por ejemplo, el m.c.d. de 2160 y 888 por este algoritmo con las siguientes etapas:

a	b	r
2160	888	384
888	384	120
384	120	24
120	24	0
24	0	

El m.c.d. de 2160 y 888 es, por tanto, 24. 24 es el mayor entero que divide simultáneamente a los dos números. De hecho, `2160 = 24 * 90` y `888 = 24 * 37`. El m.c.d. es, por tanto, el último resto no nulo.

9.3. Calcular un m.c.d. en Logo

Un pequeño algoritmo recursivo permite calcular el m.c.d. de dos números, `:a` y `:b`.

```
para mcd :a :b
   si (resto :a :b) = 0
      [devuelve :b]
      [devuelve mcd :b resto :a :b]
fin

escribe mcd 2160 888
```

proporciona como resultado `24`

Nota: Nos vemos obligados a poner paréntesis en `resto :a :b`, si no el intérprete intentará evaluar `b = 0`. Para evitar este problema de paréntesis, podemos escribir: `si 0 = resto :a :b`

9.4. Calcular una aproximación de π

Un resultado conocido de teoría de los números pone de manifiesto que la probabilidad que dos números tomados aleatoriamente sean primos entre ellos es de $\frac{6}{\pi^2} \simeq 0{,}6079$. Para intentar encontrar este resultado, vamos a:

- Tomar dos números al azar entre 0 y 1 000 000.
- Calcular su m.c.d.
- Si su m.c.d. vale 1, anadir 1 a una variable contador.
- Repetir 1000 veces
- la frecuencia de los pares de números primos entre ellos se obtendrá dividiendo la variable contador por 1000 (el número de pruebas)

```
para prueba
# Inicializamos la variable contador a 0
  haz "contador 0
  repite 1000
    [ si (mcd azar 1000000 azar 1000000) = 1
       [haz "contador :contador+1] ]
  escribe [Frecuencia:]
  escribe :contador/1000
fin
```

Nota: Igual que antes, nos vemos obligados a poner paréntesis en
`mcd azar 1000000 azar 1000000`
si no, el interprete va a intentar evaluar `1 000 000 = 1`. Para evitar este problema de paréntesis, escribe: `si 1 = mcd azar 1000000 azar 1000000`

```
prueba
0.609
prueba
0.626
prueba
0.597
```

Se obtienen valores próximos al valor teórico de 0,6097. Lo que es notable es que esta frecuencia es un valor aproximado de $\frac{6}{\pi^2} \simeq 0,6079$. Si tenemos en cuenta `f`, la frecuencia encontrada, se tiene pues: $f \simeq \frac{6}{\pi^2}$. Despejando:

$$\pi^2 \simeq \frac{6}{f} \quad \text{y así:} \quad \pi \simeq \sqrt{\frac{6}{f}}$$

Añadimos esta aproximación al programa, y transformamos el final del procedimiento `prueba`:

```
para prueba
# Inicializamos la variable contador a 0
  haz "contador 0
  repite 1000
    [ si (mcd azar 1000000 azar 1000000) = 1
       [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
  haz "f :contador/1000
# Mostramos el valor aproximado de pi
  escribe frase [Aproximacion de pi:] raizcuadrada (6/:f) fin
fin
```

que proporciona:

```
prueba
Aproximacion de pi: 3.164916190172819
prueba
Aproximacion de pi: 3.1675613357997525
prueba
Aproximacion de pi: 3.1008683647302115
```

Bien, modifiquemos el programa de modo que cuando lo lancemos, podamos indicar el número de pruebas deseadas.

```
para prueba :repeticiones
# Inicializamos la variable contador a 0
  haz "contador 0
  repite :repeticiones
    [ si (mcd azar 1000000 azar 1000000) = 1
       [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
  haz "f :contador/:repeticiones
# Mostramos el valor aproximado de pi
  escribe frase [Aproximacion de pi:] raizcuadrada (6/:f)
fin
```

Probamos con 10000 repeticiones, y obtenemos en las tres primeras tentativas:

```
prueba 10000
Aproximacion de pi: 3.1300987144363774
prueba 10000
Aproximacion de pi: 3.1517891481565017
prueba 10000
Aproximacion de pi: 3.1416626832299914
```

No está mal, ¿verdad?

9.5. Compliquemos un poco más: π que genera π...

¿Qué es un número aleatorio? ¿Es que un número tomado aleatoriamente entre 1 y 1.000.000 es un número realmente aleatorio?

Deberías darte cuenta rápidamente que nuestro modelo no hace más que aproximarse al modelo ideal. Bien, es precisamente sobre el modo de generar el número aleatorio sobre el que vamos a efectuar algunos cambios .... No vamos utilizar más la primitiva `azar`, sino que utilizaremos la secuencia de los decimales de π.

Me explico: los decimales de π siempre han intrigado a los matemáticos por su falta de regularidad, las cifras de 0 a 9 parecen aparecer en cantidades aproximadamente iguales

y de manera aleatoria. No se pueden predecir los decimales siguientes basándonos en los anteriores.

Vamos a ver a continuación como generar un número aleatorio con ayuda de los decimales de π. En primer lugar, debes obtener los primeros decimales de π (por ejemplo un millón). Existen dos programas que los calculan bastante bien. Aconsejamos `PiFast` para Windows y `ScnhellPi` para Linux.

Puedes acceder a Internet para conseguir un fichero de texto:

`http://3.14159265358979323846264338327950288419716939937510582097494459 2.com`

También en Internet, en la web de XLogo:

`http://xlogo.tuxfamily.org/common/millionpi.txt`

Para crear los números aleatorios, agrupamos de 8 en 8 cifras la serie de decimales de π. Es decir, el fichero empieza así:

$$\underbrace{3{,}1415926}_{1^{\text{er}}\ \text{numero}}\underbrace{53589793}_{2^{\text{o}}\ \text{numero}}\underbrace{23846264}_{3^{\text{er}}\ \text{numero}}\underbrace{33832795}_{\ldots}0288419716939\ldots$$

Retiro la coma "," del `3.14...` que podría equivocarnos al extraer los decimales. Bien, todo está preparado, creamos un nuevo procedimiento llamado `azarpi` y modificamos ligeramente el procedimiento `prueba`

```
para mcd :a :b
  si (resto :a :b) = 0
   [ devuelve :b ]
   [ devuelve mcd :b resto :a :b ]
fin

para prueba :repeticiones
# Abrimos el flujo numero 1 hacia el fichero millionpi.txt
# (supongamos que se encuentra en el directorio de trabajo actual
# si no estuviera, utilizar la ruta absoluta)
# pondirectorio "/home/usuario/ruta_directorio
# pondirectorio "c:\\Mis\ Documentos\ruta\ directorio
  abreflujo 1 "millionpi.txt
# Asignemos a la variable "linea la primera linea del fichero millionpi.txt
  haz "linea primero leelineaflujo 1
# Inicializamos la variable contador a 0
  haz "contador 0
  repite :repeticiones
    [ si 1 = mcd azarpi 7 azarpi 7
       [ haz "contador :contador+1 ] ]
# Calculamos la frecuencia
```

```
   haz "f :contador/:repeticiones
 #  Mostramos el valor aproximado de pi
   escribe frase [Aproximacion de pi:] raizcuadrada (6/:f)
   cierraflujo 1
 fin

 para azarpi :n
   hazlocal "numero "
   repite :n [
 # Si no hay ningun caracter en la linea
   si 0=cuenta :linea
    [haz "linea primero leelineaflujo 1]
 # Asignamos a la variable :caracter el valor de primer caracter de la linea
   haz "caracter primero :linea
 # despues eliminamos el primer caracter de la linea
   haz "linea menosprimero :linea
   haz "numero palabra :numero :caracter ]
   devuelve :numero
fin
```

El resultado es:

```
prueba 10
Aproximacion de pi: 3.4641016151377544
prueba 100
Aproximacion de pi: 3.1108550841912757
prueba 1000
Aproximacion de pi: 3.081180112566604
prueba 10000
Aproximacion de pi: 3.1403714651066386
```

Encontramos pues una aproximación del número π ¡con ayuda de sus propios decimales!

Aún es posible mejorar este programa indicando por ejemplo el tiempo usado para el cálculo. Se añade, entonces, en la primera línea del procedimiento prueba:

```
 para prueba :repeticiones
   haz "principio tiempo
   ...
```

y se modifica el final del procedimiento

```
   ...
   cierraflujo 1
   escribe frase [ Tiempo empleado: ] frase tiempo - :principio "segundos
 fin
```

Capítulo 10

Actividad sobre la suma de dos dados

Cuando se lanzan dos dados y se calcula la suma de los puntos de cada uno de ellos, se obtiene un resultado comprendido entre 2 y 12. En esta actividad vamos ver la distribución de frecuencias de las distintas tiradas y a representarla en un sencillo gráfico.

10.1. Simular el lanzamiento de un dado

Para simular el lanzamiento de un dado, vamos a utilizar la primitiva `azar`. Veamos cómo.

> `azar 6` —>devuelve un número aleatorio comprendido entre 0 y 5.
>
> Por tanto, `(azar 6) + 1` devuelve una cantidad elegida aleatoriamente del conjunto `{1,2,3,4,5,6}`.
>
> Tener en cuenta la utilización de los paréntesis; si no, el intérprete Logo leería `azar 7`. Para evitar los paréntesis, se puede escribir también `1 + azar 6`.

Se define así el procedimiento `lanzar`, que simula el lanzamiento de un dado.

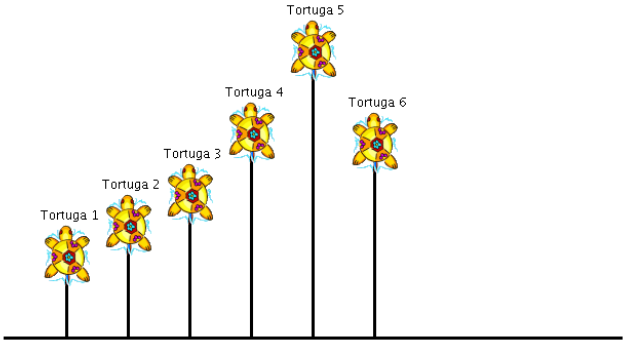
```
para lanzar
  devuelve 1 + azar 6
fin
```

10.2. El programa

Vamos a utilizar el modo *multitortuga*, para así disponer de varias tortugas sobre la pantalla.

Se utiliza la primitiva `pontortuga`, seguida del número de tortuga que se quiere seleccionar.

Un buen esquema que vale más que mil explicaciones . . . :



El objetivo es que cada tortuga, numerada de 2 a 12, avance un *paso de tortuga* cuando el resultado de la suma de la tirada de los dos dados coincida con su número. Por ejemplo, si la tirada de dados suma 8, la tortuga número 8 avanzará un paso.

La separación horizontal entre las tortugas es de 30 pasos de tortuga, y se colocarán a las tortugas con ayuda de los datos.

- Se colocará a la tortuga número 2 en `(-150 ; 0)`
- Se colocará a la tortuga número 3 en `(-120 ; 0)`
- Se colocará a la tortuga número 4 en `(-90 ; 0)`
- Se colocará a la tortuga número 5 en `(-60 ; 0)`
- ...

es decir:

```
pontortuga 2 ponpos [-150 0]
pontortuga 3 ponpos [-120 0]
pontortuga 4 ponpos [-90 0]
pontortuga 5 ponpos [-60 0]
pontortuga 6 ponpos [-30 0]
......
```

En lugar de copiar 11 veces prácticamente la misma línea de órdenes, se puede automatizar utilizando la primitiva `repitepara`. Con ayuda de esta primitiva, se puede asignar a una variable una serie de valores distribuidos en un intervalo a espacios regulares. Aquí, queremos que la variable `:i` tome sucesivamente los valores `2, 3, 4, ..., 12`. Su uso es:

```
repitepara [i 2 12]
  [lista de las instrucciones que deben realizarse]
```

Para colocar a las tortugas, se crea pues el procedimiento `inicia`

```
 para inicia
   borrapantalla
   ocultatortuga
   repitepara [i 2 12]
    [ # coloca la tortuga
     pontortuga :i ponpos lista -150 + (:i - 2) * 30 0
      # escribe el numero de la tortuga justo debajo
     subelapiz
     retrocede 15
     rotula :i
     avanza 15
     bajalapiz ]
fin
```

Observa la expresión `-150 + (:i - 2) * 30`. Con ello hacemos que el primer valor para la abscisa sea `-150`, y a cada nueva tortuga se añaden 30 (probar con distintos valores de `:i` si no se ve bien).

Finalmente se obtiene el siguiente programa:

```
para lanzar
  devuelve 1 + azar 6
fin

para inicia
  borrapantalla
  ocultatortuga
  repitepara [i 2 12]
    [ # coloca la tortuga
     pontortuga :i ponpos lista -150 + (:i - 2)*30 0
      # escribe el numero de la tortuga justo debajo
     subelapiz
     retrocede 15
     rotula :i
     avanza 15
     bajalapiz ]
fin

para empezar
  inicia
# Hacemos 1000 intentos
  repite 1000
    [ haz "suma lanzar+lanzar
      pontortuga :suma
```

```
      avanza 1 ]
 # indicamos las frecuencias de tirada
  repitepara [i 2 12]
    [ pontortuga :i
  # la ordenada de la tortuga representa el numero de tiradas
      hazlocal "frecuencia ultimo pos
      subelapiz
      avanza 10 giraizquierda 90
      avanza 10 giraderecha 90
      bajalapiz
      rotula :frecuencia/1000*100 ]
fin
```

Veamos ahora una generalización de este programa. Aquí, se pedirán al usuario el número de dados deseados así como el número de lanzamientos a efectuar.

```
para lanzar
  hazlocal "suma 0
  repite :dados
    [ hazlocal "suma :suma + 1 + azar 6 ]
  devuelve :suma
fin

para inicia
  borrapantalla ocultatortuga
  ponmaximastortugas :max + 1
  repitepara frase lista "i :min :max
   [ # coloca la tortuga
     pontortuga :i
     ponpos lista (:min - :max)/2*30 + (:i - :min)*30 0
     # escribe el numero de la tortuga justo debajo
     subelapiz retrocede 15
     rotula :i
     avanza 15 bajalapiz ]
fin

para empezar
  leeteclado [Numero de dados:] "dados
  si no numero? :dados
   [ es [largoetiqueta El numero introducido no es valido!]
     alto ]
  haz "min :dados
  haz "max 6*:dados
  leeteclado [Numero de lanzamientos a realizar] "tiradas
```

```
  si no numero? :tiradas
   [ es [largoetiqueta El numero introducido no es valido!]
     alto ]
  inicia
# Hacemos un numero de intentos igual :tiradas
  repite :tiradas
    [ pontortuga lanzar
      avanza 1 ]
# indicamos las frecuencias de tirada
  repitepara frase lista "i :min :max
    [ pontortuga :i
  # la ordenada de la tortuga representa el numero de tiradas
      hazlocal "frecuencia ultimo pos
  # normalizamos entre 0,1
      subelapiz
      avanza 10 giraizquierda 90
      avanza 10 giraderecha 90 bajalapiz
      rotula (redondea :frecuencia/:tiradas)*100 ]
fin
```

Capítulo 11

Solución a las actividades

11.1. Sección 1.4

```
para cuadrado
  repite 4 [av 150 gd 90]
fin

para tri
  repite 3 [av 150 gd 120]
fin

para puerta
  repite 2 [av 70 gd 90 av 50 gd 90]
fin

para chi
  av 55 gd 90 av 20 gd 90 av 20
fin

para desp1
  gd 90 av 50 gi 90
fin

para desp2
  gi 90 av 50 gd 90 av 150 gd 30
fin

para desp3
  sl gd 60 av 20 gi 90 av 35 bl
fin
```

```
para casa
  cuadrado desp1 puerta desp2 tri desp3 chi
fin
```

11.2. Sección 2.2

```
para supercubo
  bp sl ponpos [-30 150] bl
  ponpos [-150 150]  ponpos [-90 210]  ponpos [30 210]  ponpos [-30 150]
  ponpos [-30 -210]  ponpos [30 -150]  ponpos [30 -90]  ponpos [-30 -90]
  ponpos [90 -90]    ponpos [90 30]    ponpos [-270 30] ponpos [-270 -90]
  ponpos [-210 -90]  ponpos [-210 -30] ponpos [-90 -30] ponpos [-90 -150]
  ponpos [-210 -150] ponpos [-210 -30] ponpos [-150 30] ponpos [-30 30]
  ponpos [-90 -30]   ponpos [90 150]   ponpos[30 150]   ponpos[30 210]
  ponpos [30 90]     ponpos [90 90]    ponpos [90 150]  ponpos[90 90]
  ponpos [150 90]    ponpos [150 -30]  ponpos[90 -90]   ponpos[90 30]
  ponpos [150 90]
  sl ponpos [-150 30] bl
  ponpos [-150 150]  ponpos [-150 90]  ponpos [-210 90] ponpos [-270 30]
  sl ponpos [-90 -150] bl
  ponpos [-30 -90]
  sl ponpos [-150 -150] bl
  ponpos [-150 -210] ponpos [-30 -210]
fin
```

11.3. Sección 3.5

11.3.1. El robot

El primer dibujo está formado exclusivamente por un motivo elemental basado en un rectángulo, cuadrado y triángulo. He aquí el código asociado a este dibujo:

```
para rec :alto :ancho
# dibuja un rectangulo de altura :alto y anchura :ancho
  repite 2 [av :alto gd 90 av :ancho gd 90]
fin

para cuadrado :c
# dibuja una cuadrado de lado :c
  repite 4 [av :c gd 90]
fin
```

```
para tri :c
# dibuja un triangulo equilatero de lado :c
  repite 3 [av :c gd 120]
fin

para piernas :c
  rec 2*:c 3*:c cuadrado 2*:c
fin

para antenas :c
  av 3*:c gi 90 av :c gd 90 cuadrado 2*:c
  sl re 3 *:c gd 90 av :c gi 90 bl
fin

para robot :c
  bp ot
# El cuerpo
  rec 4*:c 28*:c
# Las piernas
  gd 90 av 2*:c
  piernas :c av 4*:c
  piernas :c av 14*:c
  piernas :c av 4*:c
  piernas :c
# La cola
  sl gi 90 av 4* :c bl
  gd 45 av 11*:c re 11*:c gi 135
# el cuello y la cabeza
  av 18*:c cuadrado :c
  av 3*:c cuadrado :c
  gd 90 av :c gi 90 av 3*:c gd 90
  cuadrado 8*:c
# Orejas
  av 4*:c gi 60 tri 3*:c
  sl gd 150 av 8 *:c gi 90 bl tri 3*:c
# Las antenas
  av 4 *:c gi 90 av 2*:c gd 90 antenas :c
  gi 90 av 4*:c gd 90 antenas :c
# los ojos
  sl re 3 *:c bl cuadrado :c
  gd 90 sl av 3*:c bl gi 90 cuadrado :c
# La boca
```

```
  sl re 3*:c gi 90 av 3*:c gd 90 bl rec :c 4*:c
fin
```

11.3.2. La ranita

```
para rana :c
   bp ot
   av 2 *:c gd 90 av 5*:c gi 90
   av 4*:c gi 90 av 7 *:c gd 90
   av 7*:c gd 90 av 21 *:c gd 90
   av 2*:c gi 90 av 2*:c gd 90
   av 9*:c gd 90 av 2*:c gi 90
   av 2*:c gd 90 av 9*:c gd 90
   av 2*:c gd 90 av 7*:c re 5*:c gi 90
   av 4*:c gd 90 av 4*:c re 4*:c gi 90
   re 2*:c gi 90 av 5*:c gi 90 av 4*:c gd 90
   av 7*:c gd 90 sl av 9*:c bl
   repite 4 [av 2*:c gd 90]
fin
```

11.4. Sección 7.2

```
para juego
# Inicializamos el numero, contador y el programa
  haz "numero azar 32
  haz "contador 0
  bucle
fin

para bucle
  leelista [Propon un numero: ] "prueba
  si numero? :prueba
# Si el valor introducido es un numero
   [ si :numero = :prueba
      [es fr fr [Has ganado en ] :contador+1 [intento(s)!] ]
      [ si :prueba > :numero
         [es [Mas pequeno] ]
         [es [Mas grande] ]
      haz "contador :contador+1
      bucle ]  ]
  [ escribe [Debes introducir un numero valido!]
    bucle ]
```

```
fin
```