



## Capítulo 6

# Variables. Procedimientos con argumentos

Muchas veces se necesita dibujar una misma figura varias veces, pero con distintas dimensiones. Por ejemplo, si queremos dibujar un cuadrado de lado 100, otro de lado 200 y un tercero de lado 50, con lo que sabemos hasta ahora necesitaríamos tres procedimientos distintos:

```
para cuadrado 1
  repite 4 [avanza 100 giraderecha 90]
fin
para cuadrado 2
  repite 4 [avanza 200 giraderecha 90]
fin
para cuadrado 3
  repite 4 [avanza 50 giraderecha 90]
fin
```

Es evidente que necesitamos una forma más simple de hacerlo, y que debería ser posible definir un único procedimiento que, de algún modo, permitiera cambiar el **argumento** de la primitiva `avanza`, es decir, el lado del cuadrado.

Ese es el papel de las **variables**.

### 6.1. Primitivas asociadas

Definimos ahora seis nuevas primitivas:

Descripción	Primitiva	Ejemplo
Guardar un valor en una variable	haz	haz "lado 115
Utilizar el valor de a	:	escribe :a
	cosa	escribe cosa "a
	objeto	escribe objeto "a
Enumerar todas las variables definidas.	listavars	listavars
Eliminar la variable var.	borravariabile, bov	borravariabile "lado

Por compatibilidad con otros intérpretes LOGO, se admite `imvars` (imprime todas las variables) con la misma función que `listavars`.

Fíjate en la diferencia:

- Para definir la variable, se antepone "
- Para leer la variable, se precede de :, la forma más cómoda de las tres posibles: `cosa "a`, `objeto "a` y `:a` son notaciones equivalentes.

Aunque lo detallaremos más adelante, debemos comentar que xLOGO trata de distinta forma los números, las palabras y las frases. Para distinguir cuándo una variable almacena un tipo distinto, debemos usar un *vocabulario* específico:

**Número:** Para guardar en la variable `lado` el valor 100:

```
haz "lado 100
```

**Palabra:** Para guardar en la variable `animal` la palabra GATO:

```
haz "animal "GATO
```

**Frase:** Para guardar en la variable `descripcion` la frase `El gato es gris`:

```
haz "descripcion [El gato es gris]
```

En xLOGO (y en otros lenguajes de programación) se utiliza el término *Lista* para referirse a aquellas variables que constan de varios elementos, por ejemplo:

```
haz "primitiva [ 5 9 23 26 45 48 ]
```

que contiene una posible combinación del sorteo de la *Lotería primitiva* NO es una frase, ya que no consta de palabras. Es una **Lista**.

Una lista puede constar de varias *sublistas*, por ejemplo:

```
haz "primitiva [ [5 9 23 26 45 48] [5 8 18 26 40 46] [20 24 28 31 36 39] ]
```

consta de tres sublistas, y se pueden combinar variables de cualquier tipo:

```
haz "listado [ [[Pepe Perez] 15 CuartoA] [[Lola Lopez] 16 CuartoB] ]
```

contiene dos sublistas, cada una con una lista (nombre), un número (edad) y una palabra (el grupo de clase)

Si el valor que guarda la variable es un número, puede operarse con ella igual que con un número:

```
haz "lado 100
avanza :lado
```

e incluso pueden usarse para definir otras:

```
haz "alto 100
haz "ancho 2*:alto
repite 2 [ avanza :alto giraderecha 90
           avanza :ancho giraderecha 90 ]
```

que dibuja un rectángulo de base doble que la altura.



¿Qué otras utilidades le ves al uso de variables? ¿Cómo las usarías para responder a las preguntas con las que cerrábamos el tema 4? ¿Puedes imaginar algún uso de las listas?

## 6.2. Procedimientos con variables

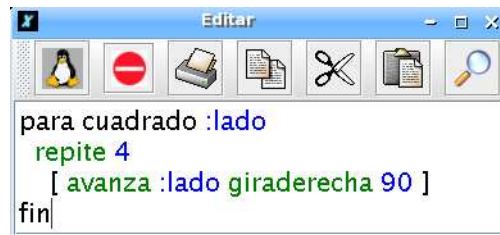
Recuperando nuestro procedimiento cuadrado:

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

introducir variables es muy simple:

- Indicamos cuál va a ser la variable, de nuevo, con un nombre adecuado: `lado`
- Sustituimos el valor numérico que nos interesa por la variable

El resultado es:



```

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

que dibuja, como ya habrás adivinado, un cuadrado. La diferencia está en que ahora el lado es desconocido, y debemos indicarle a la tortuga cuánto debe medir:

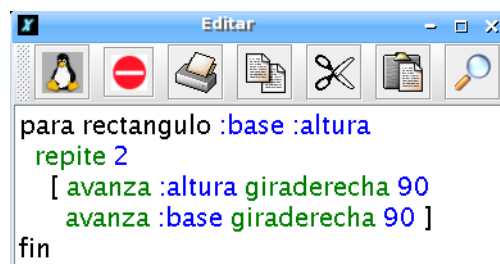
```

cuadrado 30
cuadrado 50
cuadrado 250
  
```

dibujarán cuadrados de lados 30, 50 y 250, respectivamente:



Podemos prever varios *argumentos*:



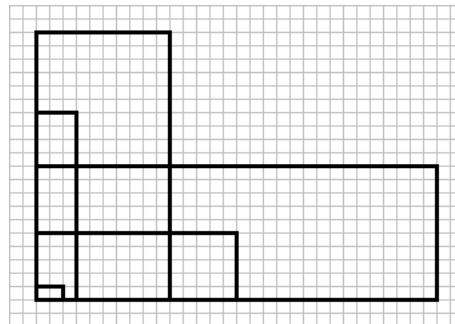
```

para rectangulo :base :altura
  repite 2
  [ avanza :altura giraderecha 90
    avanza :base giraderecha 90 ]
fin
  
```

donde vemos que depende de dos variables. Por ejemplo, `rectangulo 200 100` trazará un rectángulo de altura 200 y anchura 100.

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

genera:

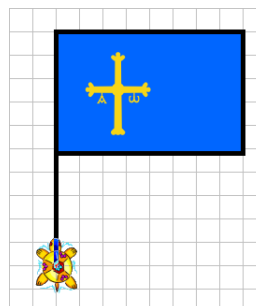


Ahora bien, si no se proporciona alguno de los argumentos al procedimiento `rectangulo`, el intérprete nos indicará con un mensaje de error que el procedimiento necesita otro argumento:

No hay suficientes datos para rectangulo

## 6.3. Ejercicios

1. Plantea un procedimiento `triangulo` que necesite una variable `lado` y que dibuje un triángulo equilátero cuyo lado sea ese valor
2. Plantea un procedimiento `rueda` que dibuje los 36 radios de longitud `largo` de una rueda
3. Plantea un procedimiento `bandera` que dibuje una bandera consistente en un mástil de longitud `mastil` y cuya tela sea un rectángulo de lados `ancho` y `alto`

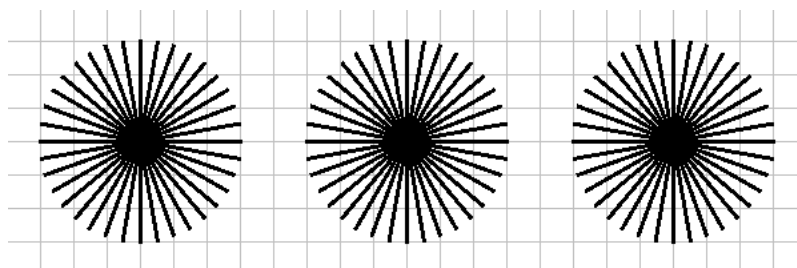


4. Plantea un procedimiento `poligono`, que reciba dos entradas: `n` y `largo`, y dibuje un polígono regular de `n` lados de longitud `largo`

**Pista:** Para hacer divisiones, xLOGO utiliza la primitiva `/`.

Por ejemplo: `escribe 256/5` devuelve `51.2`

5. Plantea un programa que dibuje una fila de `n` ruedas, cada una con 36 radios de longitud `largo`, de modo que la distancia entre los centros de dos ruedas contiguas sea `distancia`



**Prueba con distintos valores de `largo` y `distancia` y observa qué ocurre.** ¿Cuál es la relación entre `distancia` y `largo` cuando se superponen las ruedas? ¿Y cuando están separadas? ¿Qué podríamos hacer para que nunca se superpusieran?

## 6.4. Trazar una forma con distintos tamaños

Vimos como trazar un cuadrado y un rectángulo con dos tamaños distintos. Ahora volvamos al ejemplo de la casa de la página 42 y cómo modificar el código para dibujar la casa sin que importen las dimensiones. Estamos introduciendo, de este modo, el concepto de **proporcionalidad** y  **semejanza**.

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

- `casa 10` dibujará la casa de la sección 5.5.
- `casa 5` dibujará la casa a escala 0,5.
- `casa 20` dibujará una casa con las dimensiones dos veces más grandes

Según el dibujo de la sección 5.5, 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 tendrá 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, ¡sólomente 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

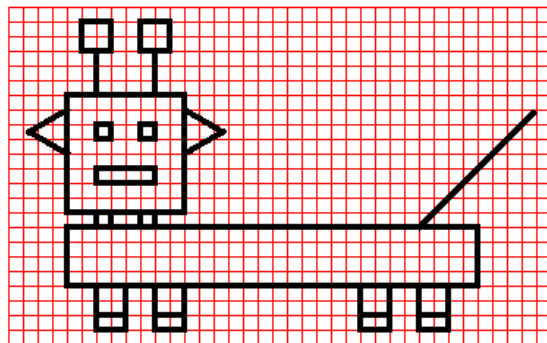
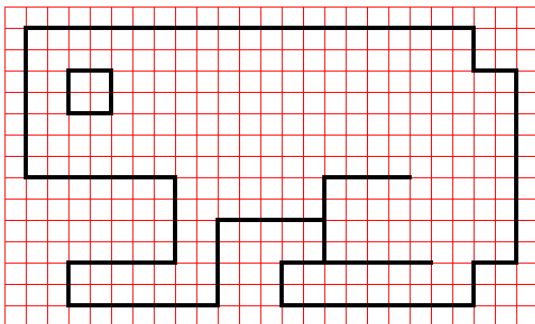
para desp3 :c
  subelapiz
  giraderecha 60 avanza 2*:c
  giraizquierda 90 avanza 3.5*:c
  bajalapiz
fin

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

```

## 6.5. Actividad avanzada

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



## 6.6. Conceptos acerca de variables

Hay dos tipos de variables:



- **Variables globales:** están siempre accesibles desde cualquier parte del programa.
- **Variables locales:** sólo son accesibles dentro del procedimiento donde fueron definidas.

En esta implementación del lenguaje LOGO, las variables locales no son accesibles desde otro sub-procedimiento. Al finalizar el procedimiento, las variables locales son eliminadas.

Las primitivas asociadas son:

Primitivas	Argumentos	Uso
haz	palabra, b	Si la variable local <b>palabra</b> existe, se le asigna el valor b (de cualquier tipo). Si no, será la variable global <b>palabra</b> la asignada con el valor b.
local	palabra	Crea una variable llamada a. Atención: la variable no es inicializada. Para asignarle un valor, hay que usar <b>haz</b> .
hazlocal	palabra b	Crea una nueva variable llamada <b>palabra</b> y le asigna el valor b.

Supongamos que en el último ejercicio de la sección anterior quisiéramos controlar la separación entre ruedas para evitar que se superpongan unas con otras. Podríamos hacer que **distancia** fuera siempre algo más del doble que **largo**, para lo que planteamos dos *sub*procedimientos distintos:

```

para ruedas :n :largo
  repite :n
    [ rueda :largo
      separa :largo ]
fin

para rueda :radio
  repite 36
    [ avanza :radio retrocede :radio giraderecha 10 ]
fin

para separa :largo
  hazlocal "distancia 2.5 * :largo
  subelapiz
  giraizquierda 90 avanza :distancia giraderecha 90
  bajalapiz
fin

```

Observa que se usan tres variables relacionadas con la longitud: **largo**, **radio** y **distancia**. Al ejecutar el programa tecleando:

```
borrapantalla ruedas 3 100
```

la tortuga lee `largo`, y le asigna el valor 100. Sin embargo, `radio` sólo “existe” mientras se está ejecutando el procedimiento `rueda` y “desaparece” al finalizar este. Puedes comprobarlo modificando el procedimiento:

```
para ruedas :n :largo
  repite :n
    [ rueda :largo separa :largo ]
  escribe :largo
  escribe :radio
fin
```

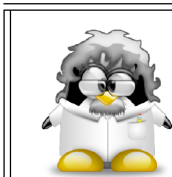
que devolverá 100 (el valor de `largo`) y un mensaje de error:

```
En ruedas, línea 4:
radio no tiene valor.
```



**Estudia qué ocurre con distancia en las dos definiciones que hemos hecho de `separa`.** Cambia `escribe :radio` por `escribe :distancia` y observa qué responde xLOGO según hayas usado `haz` o `hazlocal`.

```
para ruedas :n :largo
  repite :n
    [ rueda :largo separa :largo ]
  escribe :largo
  escribe :distancia
fin
```



**Las variables locales son muy útiles en programas largos, con varios procedimientos.** Si cada uno usa sus propias variables, no es probable que haya errores debidos a que alguna de ellas sea modificada en el procedimiento equivocado.

## 6.7. Desde la Línea de Comandos

Los procedimientos pueden ser creados y borrados desde la Línea de Comandos. Igualmente, podemos determinar cuáles han sido ya definidos y cuáles no o ejecutar una serie de órdenes sin necesidad de crear un procedimiento asociado.

### 6.7.1. La primitiva `define`

La primitiva `define` crea un nuevo procedimiento sin usar el Editor. Para ello debemos proporcionar el nombre, las variables y las instrucciones a ejecutar:

```
define nombre [variables] [instrucciones]
```

Por ejemplo:

```
define "cuadrado [lado] [repite 4 [ avanza :lado giraderecha 90]]
```

crea el procedimiento `cuadrado` con el que ya hemos trabajado antes.

### 6.7.2. Las primitivas `borra` y `borratodo`

La primitiva `borra` elimina el procedimiento indicado. La sintaxis es:

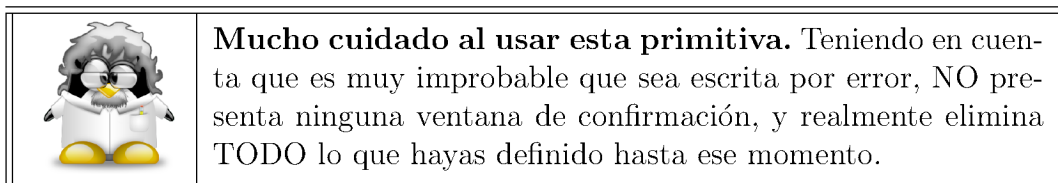
```
borra nombre
```

Por ejemplo:

```
borra "cuadrado
```

elimina el procedimiento `cuadrado` definido antes.

Por su parte, `borratodo`, sin argumentos, elimina todas las variables y procedimientos actuales.



### 6.7.3. La primitiva `texto`

Si deseamos conocer la información asociada a un procedimiento, tecleamos:

```
escribe texto nombre_proc
```

La primitiva `texto` devuelve una lista que contiene toda la información asociada al procedimiento indicado. Concretamente, devuelve una lista que contiene sub-listas:

- La primera lista contiene todas las variables fijas y opcionales del procedimiento.
- Las demás sub-listas son las líneas del procedimiento.

### 6.7.4. La primitiva listaprocs

Esta primitiva no necesita argumentos, y enumera todos los procedimientos definidos hasta ese momento en el Histórico de Comandos. Por compatibilidad con otros intérpretes LOGO, se admite `imts` (imprime todos) con la misma función.

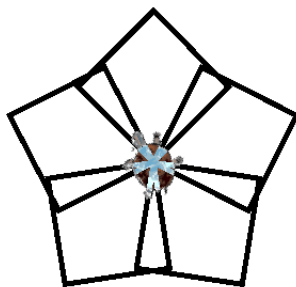
### 6.7.5. La primitiva ejecuta

Tecleando `ejecuta [lista]`, las órdenes contenidas en `lista` son ejecutadas consecutivamente.

Por ejemplo:

```
giraizquierda 27
ejecuta [ repite 5
          [ repite 4
            [ avanza 100 giraizquierda 90 ]
            giraderecha 72 ] ] ]
```

proporciona:



Un ejemplo más curioso de esta primitiva se muestra en la página de nuestro compañero Guy Walker:

<http://www.logoarts.ko.uk>

donde para dibujar un “arco iris” utiliza una lista que contiene las primitivas asociadas a seis colores (Sección 13.2.1) y con un bucle (Capítulo 11) cambia el color del lápiz (Sección 4.4) “ejecutando” su nombre:

```
...
haz "color [ rojo naranja amarillo verde azul violeta ]
repitepara [colores 1 6]
  [ poncolorlapiz ejecuta elemento :colores :color rellena
    subelapiz giraderecha 90 avanza 20 giraizquierda 90 bajalapiz ]
...
```