# MANUAL ABREVIADO PARA EL APRENDIZAJE DE MSWLOGO BÁSICO



PRINCIPALES PRIMITIVAS RELATIVAS A LOS GRÁFICOS DE LA TORTUGA

# MANUAL ABREVIADO DE MSWLOGO BÁSICO.

## 1. INTRODUCCIÓN.

LOGO es un lenguaje de programación tan poderoso como cualquier otro, pero con la ventaja para el aprendiz de que posee unas instrucciones que permiten que un cursor dibuje en la pantalla. El cursor es conocido como "la tortuga", pero en la versión MSWLogo que vamos a usar nosotros tiene forma de triángulo.

## 2. LA INTERFAZ DE MSWLOGO.

La interfaz de MSWLogo consta de dos ventanas: la ventana **Pantalla de MSWLogo**, donde se dibujan los gráficos, y la **ventana Trabajo**, situada en la parte inferior, que será donde nos comunicaremos con el programa. Con el ratón puede cambiarse el tamaño de estas ventanas. Para activar una ventana basta pinchar con el ratón sobre ella.

Dentro de la ventana Trabajo hay varias partes:

### La caja de entrada

Es un único renglón situado a la izquierda del botón *ejecutar*. En la caja de entrada podemos escribir instrucciones, las cuales se ejecutarán al pulsar **INTRO** o bien haciendo clis en el botón **Ejecutar**. También podemos modificar en ella el texto de instrucciones anteriores, para lo cual situamos el cursor sobre la instrucción en la caja de lista (ver a continuación) y dicha instrucción aparece en la caja de entrada, en la cual podemos modificarla y volver a ejecutarla.

En la caja de entrada se puede copiar con **Control+C**, pegar con **Control+V** y cortar con Control+X.

### La caja de lista

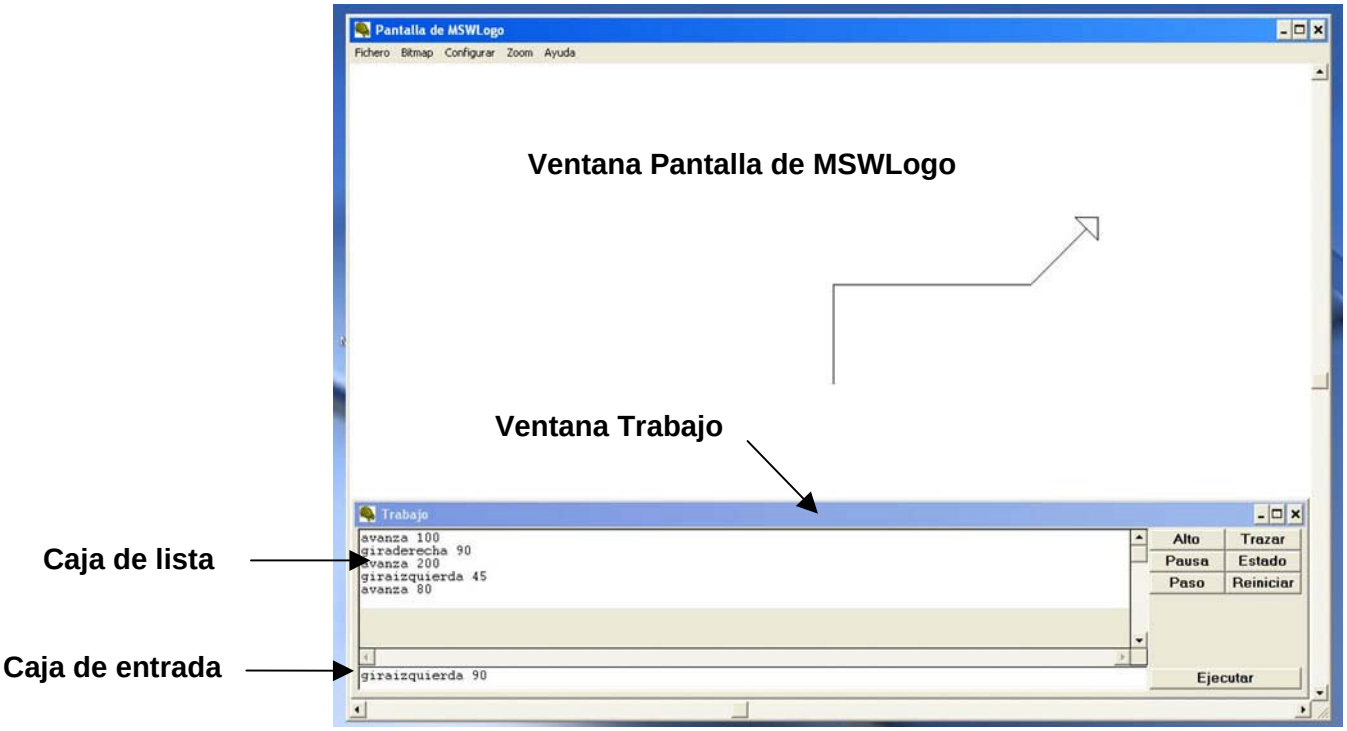
Está situada encima de la caja de entrada. En ella aparecen todas las instrucciones que se ejecutan en la caja de entrada y las respuestas que MSWLogo genera al ejecutar determinadas instrucciones.

Podemos volver a ejecutar cualquier instrucción de la caja de lista situando el cursor en la línea y pulsando INTRO, o bien haciendo doble clic en la línea.

En la caja de lista también se puede usar **Control+C** y **Control+V**, pero no Control+X.

### El botón Ejecutar

Al hacer clic sobre él se ejecutan las órdenes que haya escritas en la caja de entrada. Hace el mismo efecto que pulsar la tecla **INTRO**.



## El botón Alto

Detiene inmediatamente la ejecución de lo que esté haciendo y se dispone a esperar una nueva orden.

## El botón Reiniciar

Hacer clic sobre él es semejante a ejecutar la orden BORRAPANTALLA (BP), es decir, borra la pantalla y sitúa a la tortuga en el centro de la misma (posición (0,0)) con rumbo 0 (mirando hacia arriba).

## 3. EL MODO DIRECTO.

Las palabras clave que se usan para comunicarse con el entorno de MSWLogo se llaman **PRIMITIVAS**. Las primitivas pueden escribirse y ejecutarse directamente en la caja de entrada. Este modo de trabajar se denomina "***modo directo***". (nota: puedes escribir las primitivas de MSWLogo en mayúsculas o minúsculas).

Con los ejemplos siguientes quedará más claro el funcionamiento del modo directo:

Introduce y ejecuta sucesivamente en la caja de entrada las siguientes primitivas:

```
AVANZA 100
GIRADERECHA  90
AVANZA 100
GIRADERECHA  90
AVANZA 100
GIRADERECHA  90
AVANZA 100
GIRADERECHA  90
```

Observarás que la tortuga ha dibujado un cuadrado en la pantalla de gráficos.

Prueba ahora los siguientes ejemplos:

```
ESCRIBE "Hola
ESCRIBE [Hola amigo ¿cómo estás?]
```

Observarás que las palabras o frases escritas aparecen en la caja de lista, mezcladas con las instrucciones.

Veamos ahora qué ocurre cuando a MSWLogo se le manda ejecutar una instrucción que no conoce o en la que faltan datos, etc.

Ejecuta las siguientes instrucciones:

```
BAILAR
AVANZA
20
PONGROSOR 4
```

Observarás que MSWLogo avisa del problema:

- de que no sabe hacer lo que se le está pidiendo.
- de que le faltan datos o de que no sabe qué quieres que haga con los datos.
- de que la entrada de datos no es como la espera, etc.

Muchas primitivas de MSWLogo tienen abreviaturas para reducir el tiempo de escritura. Prueba con el siguiente ejemplo:

```
BP
AV 100
GD  90
AV 100
GD  90
AV 100
GD  90
AV 100
GD  90
```

Observarás que has borrado la pantalla (BP es la abreviatura de la primitiva BORRAPANTALLA) y has vuelto a dibujar el mismo cuadrado de antes, pero escribiendo menos texto.

Vamos a conocer las abreviaturas de algunas primitivas:

| PRIMITIVA | ABREVIATURA |
|---|---|
| AVANZA n | AV n |
| RETROCEDE n | RE n |
| BORRAPANTALLA | BP |
| BORRATEXTO | BT |
| GIRADERECHA n | GD n |
| GIRAIZQUIERDA n | GI n |
| SUBELAPIZ | SL |
| BAJALAPIZ | BJ |
| OCULTATORTUGA | OT |
| MUESTRATORTUGA | MT |
| ESCRIBE "palabra | ES "palabra |

Ten en cuenta también que podemos escribir varias órdenes en la misma línea; práctica, por ejemplo, con:

AV 100 GD 90 AV 30 GD 90 AV 100 GI 90

Puedes ejecutar cuantas veces quieras la misma orden haciendo doble clic sobre la instrucción en la caja de lista. Prueba a ejecutar cuatro veces la línea anterior y verás como dibujas una cruz.

Pero aún con las abreviaturas parece que hay que repetir mucho código. No te preocupes, MSWLogo dispone de primitivas que nos ahorrarán muchas veces esta tediosa tarea, como la primitiva REPITE.

Merece la pena dedicarle un momento a la primitiva REPITE debido a su gran utilidad. Esta primitiva no tiene abreviatura; su sintaxis es:

REPITE n [acciones]

Borra la pantalla con BP y prueba lo siguiente

REPITE 4 [ AV 100 GD 90]

En efecto, obtenemos el mismo cuadrado que ya hemos hecho varias veces, pero esta vez hemos escrito muy poco texto.

Haz estas otras pruebas, borrando la pantalla cuando lo necesites.

REPITE 6 [AV 40 GD 60]

REPITE 7 [AV 30 GD 360 / 7 ]

Nota: Cambia el 7 por otros números y pulsa [INTRO].

## 4. EL MODO PROCEDIMENTAL.

Programar en modo procedimental consiste en enseñar *procedimientos* al ordenador que, una vez asimilados, interpreta como si fueran nuevas primitivas.

Para crear un procedimiento entraremos en el editor de MSWLogo haciendo clic con el ratón en la barra de menú de la Pantalla de MSWLogo:

**Fichero > Editar....**

Nos aparece la ventana de diálogo **Editar procedimiento**. Escribimos el nombre del procedimiento que queremos crear y hacemos clic en el botón OK (o pulsamos la tecla INTRO), con lo que entramos en la pantalla del editor, donde ya nos aparecen los comandos ***para*** y ***fin*** que, como veremos después, inician y terminan, respectivamente, todos los procedimientos.

En esta ventana de diálogo nos aparecen todos los nombres de los procedimientos que tengamos definidos. Si queremos editar alguno de ellos para modificarlo, hacemos clic sobre su nombre y luego hacemos clic en OK, o bien hacemos doble clic sobre el nombre.

Si queremos editar todos los procedimientos hacemos clic en el botón Todo.

Existen dos primitivas asociadas a cualquier procedimiento, PARA y FIN. Todos los procedimientos comienzan por la primitiva PARA seguida del nombre del procedimiento y terminan con la primitiva FIN.

Veamos un ejemplo. Entra en el Editor de MSWLogo y escribe el siguiente procedimiento:

PARA CUADRADO

REPITE 4 [ AV 60 GD 90]

FIN

Salimos del editor a través de su menú haciendo clic en:

**Fichero > Guardar y salir**

Ya tenemos definido el procedimiento CUADRADO y podemos ejecutarlo desde la caja de entrada.

Todos los procedimientos que definamos pueden ejecutarse mientras no se salga de MSWLogo. Sin embargo, si en este momento salimos del programa, se perderán.

Para guardar todos los procedimientos definidos hacemos clic en **Fichero > Guardar como** en la barra de menú de MSWLogo (a partir de la primera grabación, puedo hacer **Fichero > Guardar**) y le damos un nombre al archivo (los nombres de archivo deben tener como máximo 8 letras más la extensión ".log").

Una vez tenemos definidos unos procedimientos, éstos pueden ser llamados por otros procedimientos como si fuera una primitiva más.

Esta característica permite realizar programas de forma *modular*, es decir, descomponiéndolos en partes más simples que pueden resolverse por

separado. Esta modularización puede hacerse a varios niveles ya que cada módulo puede descomponerse a su vez en partes más sencillas.

Por ejemplo, una vez que tenemos definido el procedimiento **cuadrado** de antes, podemos definir otro que lo use, como, por ejemplo, el siguiente, que dibujará tres cuadrados sucesivos.

PARA CUADROS

REPITE 3 [CUADRADO AV 70]

FIN

Cuando queramos **volver a utilizar los procedimientos que hemos definido en otra sesión** de MSWLogo, cargaremos el archivo donde los hemos guardado a través de la barra de menús:

**Fichero > Cargar ...**

Aparece una ventana donde podemos elegir la carpeta donde está y el archivo.

## 5. NOTACIÓN BÁSICA Y USO DE ALGUNOS SIGNOS EN MSWLOGO.

Llamamos ***palabras*** en MSWLogo a los conjuntos de caracteres entre los cuales no hay signos delimitadores.

Llamamos ***delimitadores*** a los signos que separan las palabras. Pueden ser espacios en blanco, corchetes y paréntesis. Las comillas y comas no son delimitadores.

Cuando se introduzca una palabra como entrada (argumento) a una primitiva o procedimiento irá precedida de unas comillas (“).

Ejemplo:

Escribe “Hola
**Hola**

Ejemplo:

(Escribe “Hola “Juan, “¿Cómo “estás?)
**Hola Juan, ¿Cómo estás?**

Una **lista** es un conjunto de palabras encerrado entre corchetes. Dentro de las listas las palabras no van precedidas de comillas.

Ejemplo:

Escribe [Hola Juan, ¿Cómo estás?]
**Hola Juan, ¿Cómo estás?**

Los números, aunque también se pueden considerar palabras, no tienen por qué ir precedidos de comillas (aunque pueden llevarlas).

Ejemplo:

Escribe 27
**27**
(Escribe 34 7 “56 20)
**34 7 56 20**

Una **línea** puede continuar en la línea siguiente si su último carácter es **~** (que se consigue tecleando 126 con el teclado numérico mientras se mantiene pulsada la tecla Alt).

Ejemplo:

(Escribe “hola “Juan, “me “alegra ~
“volver “a “verte)
**hola Juan, me alegra volver a verte**

Un punto y coma (**;**) inicia un **comentario** en una línea de instrucciones. MSWLogo ignora todos los caracteres que siguen al punto y coma hasta el final de la línea. Un carácter ~ como último carácter de la línea indica una continuación de la línea, pero no del comentario.

Ejemplo:

Para saludo
;en la siguiente línea saludamos
(es “hola “amigo) ;es un saludo breve
;en la siguiente línea nos despedimos
es “adiós
fin

saludo
**hola amigo**
**adiós**

## 6. LOS GRÁFICOS DE LA TORTUGA

Los “gráficos de la tortuga”, tienen las misión de facilitar la comprensión de la programación

El centro de la ventana de gráficos es la posición [0 0] de la tortuga. Las posiciones X positivas están hacia la derecha y las posiciones Y positivas hacia arriba. Tanto X como Y pueden variar entre – 500 y 500. Si se sobrepasan estos valores se pasa al lado opuesto de la ventana (como si fuera una esfera). El rumbo (ángulo) se mide en grados sexagesimales (360º es una vuelta

completa) en el sentido de las agujas del reloj a partir del eje Y positivo.

La tortuga en MSWLogo se representa por un triángulo isósceles, siendo la posición exacta de la tortuga la que viene dada por el punto medio de la base del triángulo (el lado mayor). La punta del triángulo indica el rumbo de la tortuga.

La tortuga lleva un lápiz para poder dibujar, pero este lápiz puede estar alzado (con lo que si la tortuga se mueve no dibuja) o bajado (al moverse la tortuga va dibujando). Además, cuando el lápiz está bajado, puede dibujar o borrar:

Modo PONLAPIZ: dibuja al moverse la tortuga.

Modo GOMA: borra por donde pasa la tortuga.

Los colores vienen definidos en LOGO por una lista con tres números en el rango 0-255. Cada uno de estos números especifican la intensidad de los colores rojo, verde y azul. Para más facilidad podemos referirnos a los colores más habituales por unos “índices de color”:

| Índice | Color | Lista |
|---|---|---|
| 0 | negro | [ 0 0 0] |
| 1 | azul | [ 0 0 255] |
| 2 | verde claro | [ 0 255 0] |
| 3 | azul claro | [ 0 255 255] |
| 4 | rojo | [255 0 0] |
| 5 | magenta | [255 0 255] |
| 6 | amarillo | [255 255 0] |
| 7 | blanco | [255 255 255] |
| 8 | marrón oscuro | [155 96 59] |
| 9 | marrón claro | [197 136 18] |
| 10 | verde oscuro | [100 162 64] |
| 11 | azul turquesa | [120 187 187] |
| 12 | naranja | [255 149 119] |
| 13 | violeta | [144 113 208] |
| 14 | naranja claro | [255 163 0] |
| 15 | gris (intermedio) | [183 183 183] |

## PRINCIPALES PRIMITIVAS RELATIVAS A LOS GRÁFICOS DE LA TORTUGA

### AVANZA *distancia*

La orden AVANZA se puede abreviar por **AV**. *distancia* es un número que indica los pasos que debe avanzar la tortuga.

Mueve la tortuga hacia delante en la dirección de su rumbo la distancia especificada.

### RETROCEDE *distancia*

La orden RETROCEDE se puede abreviar por **RE**. *distancia* es un número que indica los pasos que debe retroceder la tortuga.

Mueve la tortuga hacia atrás, es decir, en la dirección opuesta a su rumbo la distancia especificada.

### GIRAIZQUIERDA *ángulo*

La orden GIRAIZQUIERDA se puede abreviar por **GI**. *ángulo* es un número que indica los grados que girará el rumbo de la tortuga en sentido contrario a las agujas del reloj. Cada unidad es 1/360 de la circunferencia.

### GIRADERECHA *ángulo*

La orden GIRADERECHA se puede abreviar por **GD**. *ángulo* es un número que indica los grados que girará el rumbo de la tortuga en el sentido de las agujas del reloj. Cada unidad es 1/360 de la circunferencia.

### BORRAPANTALLA

Puede abreviarse por **BP**. Borra los gráficos de la ventana y envía la tortuga al centro de la pantalla y con rumbo 0.

### BAJALAPIZ

Se puede abreviar por **BL**. Pone el lápiz en posición “bajado”.

### SUBELAPIZ

Se puede abreviar por **SL**. Pone el lápiz en posición “subido”.

### PONLAPIZ

Se puede abreviar por **PLA**. Baja el lápiz y lo pone en modo PONLAPIZ.

### GOMA

Baja el lápiz y lo pone en modo GOMA.

**PONCOLORLAPIZ *color***

Se puede abreviar por **PONCL**. Pone el color del lápiz según especifica el argumento *color*, que puede ser una lista de tres números o un índice de color.

**PONGROSOR *tamaño***

Se puede abreviar por **PONG**. Pone el grosor del lápiz de la anchura dada por el argumento *tamaño*. Este argumento es una lista de dos números enteros que corresponden con la altura y la anchura, pero como MSWLogo sólo usa la anchura, sólo el segundo número es relevante (lo que se hace es que se suelen poner los dos iguales).

Ejemplo:

Pongrosor [5 5]

**MUESTRATORTUGA**

Se puede abreviar por **MT**. Hace visible la tortuga activa.

**OCULTATORTUGA**

Se puede abreviar por **OT**. Hace invisible la tortuga activa.

**PONPOS *posición***

Mueve la tortuga a las coordenadas absolutas X, Y dadas por *posición*.

*posición* es una lista (debe ir entre corchetes) de dos números, que representan las coordenadas X e Y del punto donde queremos situar la tortuga. El rumbo se mantiene.

Ejemplo: (dibuja un cuadrado)
```
bp
ponpos [0 100]
ponpos [100 100]
ponpos [100 0]
ponpos [0 0]
```

**PONXY *xcoor ycoor***

Mueve la tortuga a las coordenadas absolutas X, Y dadas por los número *xcoor* e *ycoor*.

La diferencia con PONPOS es que su argumento son dos números en vez de una lista.

Ejemplo 1:

```
ponxy 50 100
```

Ejemplo 2: (utiliza variables)
```
haz "x 50
haz "y 100
ponxy :x :y
```

**Nota:** ponpos [:x :y] sería incorrecto.

**PONX *xcoor***

Mueve la tortuga al punto dado por la coordenada X especificada por el número *xcoor* y la misma coordenada Y actual. Es decir, se mueve en dirección horizontal.

**PONY *ycoor***

Mueve la tortuga al punto dado por la coordenada Y especificada por el número *ycoor* y la misma coordenada X actual. Es decir, se mueve en dirección vertical.

**PONRUMBO *ángulo***

La orden PONRUMBO se puede abreviar por **PONR**.

Cambia el rumbo absoluto de la tortuga al especificado por *ángulo*.

**POS**

Devuelve una lista de dos números que son las coordenadas X e Y de la posición actual de la tortuga.

**COORX**

Devuelve un número que es la coordenada X de la posición actual de la tortuga.

**COORY**

Devuelve un número que es la coordenada Y de la posición actual de la tortuga.

**CIRCULO *radio***

Dibuja un círculo (si está el lápiz bajado) dado por el *radio* especificado y cuyo centro es la posición actual de la tortuga. La tortuga no se mueve.

**RELLENA**

Rellena de color una región de la ventana de gráficos en la que se encuentra la tortuga y que está limitada por líneas previamente dibujadas.

Para que una línea pueda ser detectada como límite debe estar dibujada con el mismo color de lápiz que el actual.

**PONCOLORRELLENO *color***

Se puede abreviar por **POCCR**. Pone el color para rellenos según indica el argumento *color*, que puede ser una lista de tres números o un índice de color.

**PIXEL**

Devuelve una lista de tres números que representan la intensidad de los colores rojo, verde y azul (en un rango 0-255) del píxel sobre el que se encuentra la tortuga.

## 7. EL USO DE LAS VARIABLES.

Las variables pueden ser simples o múltiples.

La más sencilla, la variable simple, puede ser comparada con una caja a la que se le pone una etiqueta (que es el nombre) y en cuyo interior guarda un contenido.

Una variable, por ejemplo, puede llamarse **X** y contener el número **34**; otra puede llamarse **Y** y contener la palabra **chocolate**.

El nombre que se da a la variable debe ser un conjunto de uno o más caracteres sin espacios en blanco y debe ir precedido por comillas. El contenido de la variable puede ser un número, el resultado de una operación, una palabra (que irá precedida de unas comillas), una lista (que irá entre corchetes) o datos introducidos desde teclado.

Las principales primitivas para trabajar con variables son HAZ y VALOR:

**HAZ *nombrevar contenido***

Asigna el valor indicado por el argumento *contenido* a la variable llamada *nombrevar*, que debe ser una palabra precedida por unas comillas. Si la variable ya existe modifica su valor y si no existe la crea. El argumento *contenido* puede ser un número, una palabra (precedida por comillas) o una lista (entre corchetes).

La variable creada es global, salvo que esta orden se ejecute dentro de un procedimiento en el que la variable se haya definido como local (inmediatamente veremos los conceptos de variable local y global).

Ejemplo:

haz “saludo “Hola
escribe :saludo
**Hola**

Con la orden haz hemos creado una variable llamada *saludo* cuyo contenido es la palabra *Hola*. Después hemos ordenado escribir el contenido de la variable *saludo*.

**VALOR *nombrevar***

La primitiva VALOR se puede abreviar por **:**.

Devuelve el valor de una variable cuyo nombre viene dado por el argumento *nombrevar*.

Ejemplo:

Haz “saludo [Hola, cómo estás]
escribe valor “saludo
**Hola, cómo estás**
escribe :saludo
**Hola, cómo estás**

Observa que cuando nos referimos al nombre de la variable precedemos éste de unas comillas (**“**) mientras que cuando nos referimos a su contenido precedemos el nombre de dos puntos (**:**).

Ejecuta las siguientes órdenes:

**Haz “edad 25**

**Escribe :edad + 9**

Observarás que MSWLogo escribe 34 que es igual a la suma del contenido de la variable edad (25) más 9.

A una variable múltiple se le llama **matriz** y puede compararse también con una caja a la que se le pone una etiqueta (nombre) pero con varios compartimentos en su interior, en cada uno de los cuales guarda un contenido.

La caja tiene un único nombre, pero cada compartimento tiene un índice (el compartimento 1, el 2, etc), lo que permite acceder individualmente a cada uno de ellos, bien para ver su contenido o para modificarlo.

Veamos un ejemplo; ejecuta las siguientes órdenes:

**Haz "mimatriz {gato lápiz 25 niño}**

**Escribe elemento 2 :mimatriz**

Observarás que aparece en pantalla la palabra **lápiz**.

A continuación ejecuta las órdenes:

**Ponelemento 4 :mimatriz "hombre**

**Escribe :mimatriz**

Aparecerá en pantalla **{gato lápiz 25 hombre}**

## 7.1. Las variables locales en los procedimientos

Antes definimos el procedimiento **cuadrado** que nos permitía dibujar un cuadrado de tamaño fijo. Pero esto no es muy eficaz porque para dibujar cuadrados de diferente tamaño tendríamos que crear otro procedimiento. Las variables locales permiten resolver este problema. Edita el siguiente procedimiento:

```
PARA CUADRADOVAR :L
BP
REPITE 4 [AV :L GD 90 ]
FIN
```

Ejecuta las siguientes órdenes:

**Cuadradovar 100**

**Cuadradovar 150**

**Haz "lado 30**

**Cuadradovar :lado**

Observarás que en cada caso se dibujan cuadrados de distintos lados en función del valor que introduzcamos después del nombre del procedimiento (que pueden ser en este caso números o contenidos de variables que contengan números).

Los valores que introducimos pasan a ocupar el valor de la variable **L**.

A **L** se le llama *variable local*, porque sólo existe dentro del procedimiento en el que se crea. Una vez terminado el procedimiento desaparece. En efecto, si se ejecuta la orden:

**Escribe :L**

MSWLogo responde que L no tiene valor.

Un procedimiento puede tener varias entradas. Edita, por ejemplo, el siguiente procedimiento para realizar un rectángulo

```
PARA RECTANGULO :B :H
REPITE 2 [AV :H GD 90 AV :B GD 90]
FIN
```

A continuación ejecuta la orden:

**Rectangulo 100 80**

Observarás que dibuja un rectángulo de base 100 unidades y altura 80 unidades. O sea, 100 ocupa el lugar de B y 80 el lugar de H.

Dentro de un procedimiento también se pueden definir variables locales con la primitiva LOCAL..

**LOCAL *nombrevar***

Esta primitiva puede adoptar diversas formas:

LOCAL *nombrevar*

LOCAL *listanombrevar*

(LOCAL *nombrevar1 nombrevar2* ...)

O sea, que podemos definir como local una sola variable o varias al mismo tiempo.

Con esta función se crea una o varias variables locales con los nombres dados por nombrevar, nombrevar1, etc (precedidos de comillas) o incluidos en la lista *listanombrevar* (nombres de las variables sin comillas pero encerrados todos entre corchetes).

Estas variables sólo existen en el procedimiento en el que se definen. Fuera de dicho procedimiento no tienen valor.

Ejemplo: el siguiente procedimiento calcula potencias de 2 (2 elevado a un exponente):

```
PARA POTEN2 :EXP
LOCAL "X
HAZ "X 1
REPITE :EXP [HAZ "X :X * 2]
DEVUELVE :X
FIN
```

Tanto EXP como X son variables locales. Ejecuta el procedimiento y a continuación ordena que se escriban sus valores y verás que no tienen valor.

## 7.2. Las variables globales

Las variables globales, a diferencia de las locales, pueden existir en el área de trabajo durante todo el tiempo en que LOGO está activo, lo cual permite usarlas en cualquier procedimiento.

Ejemplo:

```
Para ejemplovariables
(Local “uno “dos)
haz “uno  “primero,
haz “dos  “segundo
haz “tres  “tercero
escribe (frase [El orden es] :uno  :dos [y] :tres)
fin

ejemplovariables
El orden es primero, segundo y tercero
escribe :uno
uno No tiene valor
escribe :tres
tercero
```

Las variables *uno* y *dos* son locales pero la variable *tres* es global, ya que no se ha definido como local.

Nota: la orden (Local “uno “dos) se podría sustituir por Local [uno dos]

## 8. DEFINICIÓN DE PROCEDIMIENTOS CON PARA Y FIN

**PARA *nombreproc* :*entrada1* :*entrada2* .....**

La primitiva PARA le dice a MSWLogo que acepte una definición de procedimiento. El procedimiento se llamará *nombreproc* y no debe existir otro procedimiento con el mismo nombre (pues será sustituido por el nuevo). Las entradas o argumentos se llamarán *entrada1*, *entrada2*, etc. Un procedimiento puede tener varias entradas o ninguna. Los nombres de procedimientos y de entradas se pueden escribir indistintamente en mayúsculas o minúsculas.

Obsérvese que el nombre del procedimiento no debe estar precedido por **“** y que los nombres de los argumentos van precedidos por **:** .

Al invocar el procedimiento hay que darle tantos valores como entradas tenga.

Ejemplo:

```
para saludo :nombre :apellido
es (frase “Hola :nombre :apellido [, cómo estás])
fin
saludo “Pedro “García
Hola Pedro García , cómo estás
saludo “Pedro
No hay suficientes datos para saludo
```

Nota: En la segunda ejecución del procedimiento *saludo* sólo le hemos dado un valor, cuando tiene dos entradas; por eso da error.

**FIN**

Esta palabra especial pone término a un procedimiento. Es la última línea y no lleva ningún argumento.

## 9. MANEJO DE DATOS

En primer lugar distingamos los distintos tipos de datos que manejamos en LOGO:

- **Carácter**: es una única letra, número o pulsación de teclado.
- **Palabra**: es un conjunto de caracteres que no tiene espacios entre sí.
- **Lista**: es un conjunto de palabras separadas por espacios. Van encerradas entre corchetes.
- **Matriz:** es un conjunto de listas separadas por espacios entre sí. Van encerradas entre llaves.

**LISTA *objeto1 objeto2***

Cuando tiene más de dos argumentos debe ir entre paréntesis:

(LISTA *objeto1 objeto2 objeto3 .....*)

Devuelve una lista cuyos miembros son sus argumentos, que pueden ser cualquier tipo de datos de LOGO (carácter, palabra, lista) o variables que contengan estos datos.

Ejemplo:

haz "rojo 100
haz "verde 100
haz "azul 100
muestra (lista :rojo :verde :azul)
**[100 100 100]**

Ejemplo:

haz “x 100 haz “y 50
ponpos (lista :x :y)
;<la tortuga se moverá a la posición [100,50]>

**FRASE *objeto1 objeto2***

Cuando tiene más de dos argumentos debe ir entre paréntesis:

(FRASE *objeto1 objeto2 objeto3 .....*)

Hace lo mismo que LISTA, la diferencia es que si alguno de sus argumentos es una lista le elimina los corchetes

Ejemplo:

haz “número 4 haz “cubo 64
es (lista [El cubo de] :número [es] :cubo)
**[El cubo de] 4 [es] 64**
es (frase [El cubo de] :número [es] :cubo)
**El cubo de 4 es 64**

**PRIMERO *objeto***

Esta primitiva se puede abreviar por **PRI**.

Si el argumento es una palabra, devuelve el primer carácter de la palabra. Si el argumento es una lista, devuelve el primer miembro de la lista.

Ejemplo:

escribe primero [1 2 3]
**1**
escribe primero "Hola
**H**

**ULTIMO *objeto***

Esta primitiva se puede abreviar por **UL**.

Funciona igual que PRIMERO salvo que devuelve el último carácter (si el argumento es una palabra) o palabra (si el argumento es una lista).

**MATRIZ *tamaño***

Devuelve una matriz con tantos elementos como indique el argumento *tamaño*. Cada elemento es inicialmente una lista vacía. El primer elemento de la matriz es el elemento número 1.

Recordemos que una matriz es una variable múltiple que puede guardar varios contenidos simultáneamente, pudiéndose acceder individualmente a cada uno de ellos a través de un índice, bien para leerlos o para modificarlos.

Los elementos de la matriz pueden seleccionarse con la primitiva ELEMENTO y cambiarse con PONELEMENTO. Las matrices se escriben en pantalla con ESCRIBE.

Ejemplo 1:

haz “mimatriz matriz 3

Esta instrucción crea una matriz llamada *mimatriz* de tres elementos. Los valores iniciales de cada uno de los elementos son listas vacías.

También puede crearse una matriz y al mismo tiempo darle valores, escribiendo éstos dentro de llaves.

Ejemplo 2:

haz “mimatriz {10 15 20}

**LISTAAMATRIZ *lista***

Convierte la lista (o variable que contiene una lista) *lista* en una matriz.

Ejemplo:

Escribe listaamatriz [1 2 3]
**{1 2 3}**

**ELEMENTO *índice objeto***

Si el argumento *objeto* es una palabra, devuelve el carácter número *índice* de la palabra; si es una lista, devuelve el miembro número *índice* de la lista. Si es una matriz devuelve el elemento número *índice* de la matriz. El índice comienza en 1.

Ejemplo:

muestra elemento 2 [a b c]
**b**
muestra elemento 3 "ABC
**C**

**PONELEMENTO *índice nombrematriz objeto***

Cambia el contenido del elemento número *índice* de la matriz *nombrematriz* por el contenido *objeto*. El argumento objeto puede ser un número, una palabra o una lista.

Ejemplo:

haz “mimatriz {mesa silla}
ponelemento 2 :mimatriz “armario
es :mimatriz
**{mesa armario}**

**CUENTA *objeto***

Si el argumento objeto es una palabra, devuelve el número de caracteres que la componen. Si objeto es una lista o una matriz, devuelve el número de miembros que la componen.

Ejemplo 1:

muestra cuenta [1 2 3]
**3**
muestra cuenta "ab
**2**

**CARÁCTER *número***

Esta primitiva se puede abreviar por **CAR**.

Devuelve el carácter cuyo código ASCII coincide con el *número* que se le da como argumento.

**IGUALES? *objeto1 objeto2***

Otra forma que adopta esta primitiva es:

*objeto1* = *objeto2*

Devuelve VERDADERO si los argumentos *objeto1* y *objeto2* son iguales y FALSO si no lo son.

## 10. OPERACIONES ARITMÉTICAS

Son muchas y simples de entender. Todas las operaciones convencionales y además: potencia, exponencial, raíz cuadrada, logaritmos y todas las trigonométricas. Sólo veremos algunas:

**SUMA *valor1 valor2***

Otras formas de esta primitiva son:

*valor1* + *valor2*

(SUMA *valor1 valor2 valor3* .....)

La segunda forma (entre paréntesis) se usa cuando son más de dos sumandos.

Devuelve la suma de los valores numéricos de los argumentos, que pueden ser números o variables cuyo contenido sean números.

**DIFERENCIA *valor1 valor2***

Adopta también la forma:

*valor1 – valor2*

Devuelve la diferencia entre las entradas. En su segunda forma hay que tener cuidado al colocar espacios, ya que si se deja un espacio entre el primer número y el signo – y no se deja espacio entre dicho signo y el segundo número, lo que entiende MSWLogo es que estamos introduciendo dos números, siendo el segundo de ellos negativo, en lugar de una operación.

Observa los siguientes ejemplos:

escribe 7 – 5
**2**
escribe 7–5
**2**
escribe 7 –5 ; hay un espacio entre 7 y el –
**7**
**No me has dicho qué hacer con –5**
escribe 7– 5
**2**

**PRODUCTO *valor1 valor2***

Otras formas de esta primitiva son:

*valor1* * *valor2*

(PRODUCTO *valor1 valor2 valor3* .....)

La segunda forma (entre paréntesis) se usa cuando son más de dos factores.

Ejemplo:

haz “a 5 haz “b 6
escribe producto :a * :b
**30**

**DIVISION *valor1 valor2***

Otras formas de esta primitiva son:

*valor1* / *valor2*

Devuelve el resultado de la división del *valor1* entre el *valor2*.

**RESTO *valor1 valor2***

Devuelve el resto de la división entre *valor1* y *valor2*; para ello, ambos valores deben ser enteros y el resultado es un número entero.

**ENTERO *valor***

Devuelve la parte entera del número que se le da como argumento *valor* (*valor* puede ser un número o el nombre de una variable que contenga un número o una operación aritmética).

Ejemplo:

haz “a 23.67  haz “b 8.42
escribe entero :a + :b
**32**

**REDONDEA *valor***

Devuelve el entero más cercano al valor numérico de su argumento.

**RAIZCUADRADA *valor***

Se puede abreviar por **RC**. Devuelve la raíz cuadrada del valor numérico de su argumento.

**POTENCIA *valor1 valor2***

Devuelve *valor1* elevado a *valor2*. Si *valor1* es negativo, *valor2* debe ser un número entero.

**MENOR? *valor1 valor2***

Otra forma que adopta esta primitiva es:

*valor1* < *valor2*

Devuelve VERDADERO si el valor numérico del primer argumento (*valor1*) es menor que el del segundo argumento (*valor2*). Devuelve FALSO en caso contrario.

**MAYOR? *valor1 valor2***

Otra forma que adopta esta primitiva es:

*valor1* > *valor2*

Devuelve VERDADERO si el valor numérico del primer argumento (*valor1*) es mayor que el del segundo argumento (*valor2*). Devuelve FALSO en caso contrario.

**AZAR *valorlímite***

Devuelve un número entero positivo (se incluye el 0) aleatorio menor que la entrada *valorlímite*, que debe ser un número entero positivo o el nombre de una variable que lo contenga.

Ejemplo:

repite 5 [muestra azar 10]

Ejemplo:

haz “valim 25
escribe azar :valim

**SACAALAZAR *objeto***

Devuelve un elemento elegido al azar del *objeto* de entrada. Si el argumento objeto es una palabra, devuelve al azar uno de sus caracteres; si es un número, devuelve uno de sus dígitos, si es una lista o una matriz, devuelve uno de sus elementos. Si es una variable, devuelve al azar uno de los elementos de su contenido.

Ejemplo:

sacaalazar “dibujo
**b**
sacaalazar [1 2 3 4]
**2**

## 11. COMUNICACIÓN DEL PROGRAMA CON EL USUARIO.

Hasta ahora sólo hemos usado la primitiva ESCRIBE para que nuestros procedimientos comuniquen algo al usuario. Veremos que existen además otras primitivas que permiten captar información del usuario, que éste puede suministrar por el teclado o por el ratón.

**ESCRIBE *objeto***

Esta primitiva puede abreviarse por **ES**.

Adopta otra forma (entre paréntesis) cuando se quieren escribir varios objetos:

(ESCRIBE *objeto1 objeto2* .....)

Esta primitiva escribe su argumento o argumentos en una sola línea, separados por espacios y al final produce un retorno de carro.

Si la entrada es una lista no escribe los corchetes; sí se escriben los corchetes de las sublistas (listas dentro de listas).

Ejemplos:

es “hola
**hola**
(es “hola, “¿cómo “estás?)
**hola, ¿cómo estás?**
(es “muy [bien ¿y tú?])
**muy bien ¿y tú?**
haz “saludo “hola
es :saludo
**hola**

**LEELISTA**

Esta primitiva se puede abreviar por **LL**.

Lee una línea desde el teclado y devuelve la línea como una lista. La entrada se introduce en una ventana de diálogo llamada Modo LISTA.

**LEEPALABRA**

Esta primitiva se puede abreviar por **LP**.

Lee una línea desde el teclado y la devuelve como una palabra (aunque la línea tenga espacios, corchetes, etc). La entrada se introduce en una ventana de diálogo llamada Modo Entrada.

**BORRATEXTO**

Esta primitiva se puede abreviar por **BT**.

Borra el texto (órdenes y respuestas del sistema) de la caja de lista de la ventana Trabajo.

**LEECAR**

Esta primitiva se puede abreviar por **LC**.

Devuelve el valor ASCII correspondiente a la última tecla pulsada.

Ejemplo:

Ponteclado [es leecar]
;haz clic en la ventana de gráficos y pulsa teclas

**PONTECLADO *tecla_pulsada***

Esta primitiva adopta otra forma:

(PONTECLADO *tecla_bajada tecla_liberada*)

Esta orden captura los eventos del teclado (pulsaciones) y cuando ello se produce ejecuta la o las listas de instrucciones de su argumento.

Si se quiere obtener el carácter pulsado se llama a LEECAR.

Para que funcione, la ventana “Pantalla de MSWLogo debe tener el foco (no la ventana Trabajo). Esto puede hacerse haciendo clic en ella con el ratón o, cuando se necesita controlar por software, con la primitiva PONFOCO.

En su primera forma ejecuta la lista de instrucciones *tecla_pulsada* al pulsar una tecla. Si se mantiene pulsada la tecla vuelve a ejecutar las instrucciones una y otra vez.

En su segunda forma ejecuta la lista de instrucciones *tecla_bajada* al pulsar la tecla y ejecuta la lista de instrucciones *tecla_subida* al soltar la tecla. En este caso, si se mantiene la tecla pulsada ejecuta las instrucciones de la *lista tecla_bajada* una sola vez.

Ejemplo:

(ponteclado [es “hola] [es “adiós])
ponfoco [Pantalla de MSWLogo]
;<pulso cualquier tecla y mantengo>
**hola**
;<suelto la tecla>
**adiós**
quitateclado

**QUITATECLADO**

Esta orden desactiva la captura de los eventos del teclado.

**PONRATON *bot_izq_abajo bot_izq_arriba bot_der_abajo bot_der_arriba mover***

Esta orden captura los eventos del ratón (clics) y cuando ello se produce ejecuta la o las listas de instrucciones de su argumento.

Si se quiere obtener la posición del ratón se usa POSRATON.

Para que funcione, la ventana “Pantalla de MSWLogo debe tener el foco (no la ventana Trabajo). Esto puede hacerse haciendo clic en ella con el ratón o, cuando se necesita controlar por software, con la primitiva PONFOCO.

Los argumentos son listas de instrucciones:

***bot_izq_abajo*** es la lista de instrucciones que se ejecuta cuando se presiona el botón izquierdo.

***bot_izq_arriba*** es la lista de instrucciones que se ejecuta cuando se libera el botón izquierdo.

***bot_der_abajo*** es la lista de instrucciones que se ejecuta cuando se presiona el botón derecho.

***bot_der_arriba*** es la lista de instrucciones que se ejecuta cuando se libera el botón derecho.

***mover*** es la lista de instrucciones que se ejecuta cuando se mueve el puntero del ratón.

Algunas de las listas pueden ser vacías.

**QUITARRATON**

Desactiva la captura de los eventos de ratón.

**POSRATON**

Devuelve la posición del puntero del ratón tras el último evento (previamente se usó PONRATON).

La posición del puntero la devuelve como una lista de dos miembros con las posiciones X e Y que definen dicha posición.

Ejemplo:

```
subelapiz
ponraton [ponpos posraton bajalapiz] ~
              [subelapiz] [ ] [ ] [ponpos posraton]
;opera con el ratón por la pantalla de Logo
quitarraton
```

**PONFOCO *títuloventana***

Esta primitiva permite que MSWLogo controle la ventana que tiene el foco (activa). Cada ventana se especifica por su título.

Los dos posibles usos de esta primitiva son:

Ponfoco [Pantalla de MSWLogo]

Ponfoco [Trabajo]

## 11.1. FUNCIONES BÁSICAS DE WINDOWS: VENTANAS PREDEFINIDAS

Con las funciones de Windows el programador puede crear una Interfaz Gráfica de Usuario mediante ventanas. Aquí veremos sólo las ventanas predefinidas (no controlamos su posición, tamaño, etc.). Otras funciones de Windows permiten usar botones, barras de desplazamiento, etc.

**MENSAJE *bandera cuerpo***

Detiene el proceso en curso y despliega una ventana con el mensaje *cuerpo*. En la barra de título aparece la etiqueta *bandera*. El proceso continua cuando el usuario hace clic sobre el botón ACEPTAR o CANCELAR.

***bandera*** : Es una lista que se usa como etiqueta de la ventana.

***cuerpo*** : Es una lista con el texto del mensaje. La ventana se adapta al tamaño del mensaje, el cual está limitado a 235 caracteres.

Ejemplo:

```
mensaje [Título] [Contenido del mensaje]
```

**SELECCIONBOX *bandera listaopciones***

Detiene el proceso y despliega una ventana de mensaje en la que en la barra de título aparece la lista *bandera* y con las opciones de la lista *listaopciones*. El proceso anterior continua cuando el usuario hace clic sobre el botón OK o CANCELAR.

Una vez que se hace clic sobre el botón OK, devuelve el índice correspondiente al elemento seleccionado (1 para la opción 1, 2 para la opción 2, etc). Si se hace clic sobre el botón CANCELAR para la ejecución del programa y no devuelve nada (Nota: no devuelve 0 como dice en la ayuda).

***bandera*** : Es una lista que se usa como etiqueta de la ventana.

***listaopciones*** : Es una lista con opciones.

Ejemplo:

```
es seleccionbox [Escoja Color] [Rojo Verde Azul]
; <seleccione Verde y pulse OK>
2
es seleccionbox [Colores] [[Rojo Verde Azul]
[Uno de los tres] Ninguno]
; <seleccione Rojo Verde Azul y pulse OK>
1
haz “x “Rojo haz “y “Verde haz “z “Azul
es seleccionbox [Elija]  (lista :x  :y :z)
; <seleccione Azul y pulse OK>
3
```

**SINOBOX *bandera* cuerpo**

Detiene el proceso y despliega una ventana con el mensaje *cuerpo*. En la barra de título aparece la etiqueta *bandera*. El proceso anterior continua cuando el usuario hace clic sobre uno de los botones SI o NO. Si se hace clic en CANCELAR se para la ejecución del programa.

Una vez que se hace clic sobre un botón, devuelve un valor booleano, que es Verdadero (si se hace clic sobre SI) o Falso (si se hace clic sobre NO).

***bandera*** : Es una lista que se usa como etiqueta de la ventana.

***cuerpo*** : Es una lista con el texto del mensaje. La ventana se adapta al tamaño del mensaje, el cual está limitado a 235 caracteres.

Ejemplo:

escribe sinobox [Pregunta] [¿Te gusta Logo?]
; <Haga clic sobre el botón SI>
**verdadero**

**PREGUNTABOX *bandera cuerpo***

Detiene el proceso y despliega una ventana con el mensaje *cuerpo*. En la barra de título aparece la etiqueta *bandera*. El proceso anterior continua cuando el usuario hace clic sobre el botón OK o pulsa INTRO. Si se hace clic sobre el botón CANCELAR se para la ejecución del programa.

Una vez que se haga clic sobre el botón OK o se pulse INTRO, devuelve una lista con la respuesta del usuario.

***bandera*** : Es una lista que se usa como etiqueta de la ventana.

***cuerpo*** : Es una lista con el texto del mensaje. Su extensión está limitada por el tamaño estándar de la ventana.

Ejemplo:

escribe preguntabox [Pregunta] [¿Tienes sed?]
; <Escriba *Sí, tengo sed* y haga clic sobre OK>
**Sí, tengo sed**

## 12. ÓRDENES DE CONTROL

**VERDADERO**

Es una palabra especial que indica el resultado positivo de una condición.

Ejemplo:

escribe 1 = 1
**verdadero**
si “verdadero [es [Hola, cómo estás]]
**Hola, cómo estás**

**FALSO**

Es una palabra especial que indica el resultado negativo de una condición.

escribe 1 = 0
**falso**
sisino “falso [es [Hola, cómo estás]][es [Adiós]]
**Adiós**

**DEVUELVE *valor***

Esta primitiva se puede abreviar por **DEV**.

Termina la ejecución del procedimiento en el que aparece y devuelve un valor. Este valor puede ser un número, una palabra, una lista, etc.

Ejemplo:

para ejemplodev :num1 :num2
devuelve (:num1 + :num2)
es "hola
fin
escribe ejemplodev 5 7
**12**

Nota: obsérvese que no se escribe hola, ya que el procedimiento se interrumpe al llegar a DEVUELVE.

**ALTO**

Termina la ejecución del procedimiento en el que aparece. Devuelve el control al contexto en el que se llamó el procedimiento. El procedimiento no devuelve ningún valor.

Ejemplo:

para myprog :arg
escribe [Antes de Alto]
si 1=:arg [alto]
escribe [Después de Alto]

fin
myprog 1
**Antes de Alto**
myprog 2
**Antes de Alto**
**Después de Alto**

**ESPERA *tiempo***

Retrasa la ejecución durante un tiempo (expresado en segundos) igual al número introducido en el argumento *tiempo* dividido por 60. Es decir, para retrasar 1 segundo introduciremos como argumento 60.

Ejemplo:

para pruebatiempo
es [ahora empiezo]
espera 300
es [ya acabé]
fin
pruebatiempo
**ahora empiezo**
;<*al cabo de 5 segundos escribirá:*>
**ya acabé**

## 12.1. LAS SENTENCIAS CONDICIONALES

Las **sentencias condicionales** son la base del "pensamiento" de los programas. Permiten al programa "elegir" ejecutar unas instrucciones u otras o ninguna en función de que se cumplan o no unas condiciones.

**SI *prueba listainstrucción***

Esta primitiva presenta una segunda forma:

(SI *prueba listainstrucción1 listainstrucción2*)

Funcionamiento de la primera forma:

Si la condición, dada por el argumento *prueba*, tiene el valor VERDADERO, entonces se ejecuta la lista de instrucciones contenidas en el argumento *listainstrucción*. Si el valor de *prueba* es FALSO, entonces no se ejecuta nada.

Funcionamiento de la segunda forma:

Funciona igual que una primitiva que veremos a continuación llamada SISINO. Si el valor de *prueba* es VERDADERO se ejecuta *listainstrucción1* y si tiene valor FALSO se ejecuta *listainstrucción2*. Obsérvese que hay que usar los paréntesis para que funcione bien. Si se omiten, MSWLogo la trata como si fuera SISINO, enviando un mensaje de advertencia.

Ejemplo:

haz "a 1
si :a =1 [es "Hola]
**Hola**
(si :a = 1 [es "Hola][es "Adiós])
**Hola**
haz "a 2
(si :a = 1 [es "Hola][es "Adiós])
**Adiós**

Tanto en una como en otra forma, la primitiva SI puede devolver un valor.

Ejemplo:

para max :a :b
devuelve (si :a > :b [:a][:b])
fin
escribe max 1 2
**2**

**SISINO *prueba listainstrucc1 listainstrucc2***

Esta primitiva condicional funciona exactamente igual que la segunda forma de la primitiva SI que acabamos de describir, con la única diferencia de que no lleva paréntesis.

## 12.2. LOS BUCLES

Otra potente herramienta de LOGO son los bucles, es decir, porciones de programa que se ejecutan una y otra vez durante un determinado número de veces o mientras se cumpla o hasta que se cumpla una condición.

Ejemplo:

HAZ.HASTA [HAZ "VAR AZAR 50

ESCRIBE :VAR] [:VAR = 25]

Se ejecutan una y otra vez las instrucciones de sacar un número aleatorio entre 0 y 50 y escribirlo hasta que se cumpla la condición de que salga el 25.

**REPITE *número listainstrucción***

Repite una lista de instrucciones (dada en el argumento *listainstrucción*) tantas veces como indique el argumento *número*. Si como argumento *número* ponemos la palabra **siempre**, la lista de instrucciones se repite indefinidamente.

**DESDE *listacontrol listainstrucción***

El procedimiento DESDE ejecuta repetidamente una lista de instrucciones al tiempo que incrementa una variable local. Cuando esta variable supera un valor límite deja de ejecutar dichas instrucciones y se ejecuta la línea siguiente.

El argumento *listacontrol* es una lista (entre corchetes) con tres o cuatro miembros:

- El 1º es una palabra que es el nombre de una variable local
- El 2º contiene el valor inicial de la variable (puede ser un número o una variable cuyo contenido sea un número).
- El 3º contiene el valor límite de la variable (puede ser un número o una variable cuyo contenido sea un número).
- El 4º contiene el incremento que sufre la variable en cada pasada. Este 4º miembro es opcional. Si se omite, MSWLogo toma por defecto el valor 1 (si el 2º miembro es menor que el 3º) ó –1 (si el 2º miembro es mayor que el 3º).

Si los miembros 2º y 3º son iguales la lista de instrucciones se ejecuta una sola vez.

El argumento *listainstrucción* es una lista de instrucciones (entre corchetes).

El orden en el que DESDE efectúa las diversas acciones es el siguiente:

1. Asigna el valor inicial a la variable de control.
2. Compara el valor de la variable con el valor límite; si la variable es mayor que el valor límite, la ejecución continúa en la línea siguiente y si no lo es se ejecutan las instrucciones de la lista y se incrementa la variable de control en el valor del incremento.
3. Vuelve a comparar la variable de control con el valor límite y así sucesivamente hasta que el valor de la variable supera al valor límite.

Ejemplo:

```
desde [i 2 4 1] [es :i]
2
3
4
```

**HAZ.HASTA *listainstrucción pruebalista***

Este procedimiento ejecuta la lista de instrucciones dada en el argumento *listainstrucción* hasta que la condición *pruebalista* (condición dentro de una lista) tenga el valor VERDADERO. Primero ejecuta y después verifica si se cumple la condición; esto implica que la lista de instrucciones se ejecuta al menos una vez.

Obsérvese que la condición es el 2º argumento.

Ejemplo:

```
haz "i 0
haz.hasta [haz "i :i+1 escribe :i] [:i>3]
1
2
3
4
```

**HASTA *pruebalista listainstrucción***

Este procedimiento ejecuta la lista de instrucciones dada en el argumento *listainstrucción* hasta que la condición *pruebalista* (condición dentro de una lista) tenga el valor VERDADERO. Primero verifica si se cumple la condición y después ejecuta. Puede que la lista de instrucciones no se ejecute ninguna vez.

Obsérvese que la condición es el 1er argumento.

Ejemplo:

```
haz "i 0
hasta [:i>3] [haz "i :i+1 escribe :i]
1
2
3
4
```

**HAZ.MIENTRAS *listainstrucción pruebalista***

Este procedimiento ejecuta la lista de instrucciones dada en el argumento *listainstrucción* mientras se cumpla la condición *pruebalista* (condición dentro de una lista). Primero ejecuta las

instrucciones y después verifica si se cumple la condición. Por ello, el bloque de instrucciones se ejecuta al menos una vez.

Obsérvese que la condición es el 2º argumento.

Ejemplo:

```
haz "i 0
haz.mientras [haz "i :i+1 escribe :i] [:i<3]
1
2
3
```

**MIENTRAS *pruebalista listainstrucción***

Este procedimiento ejecuta la lista de instrucciones dada en el argumento *listainstrucción* mientras se cumpla la condición *pruebalista* (condición dentro de una lista). Primero verifica si se cumple la condición y si se cumple ejecuta las instrucciones. Por ello, puede que el bloque de instrucciones no se ejecute ninguna vez.

Obsérvese que la condición es el 1er argumento.

Ejemplo:

```
haz "i 0
mientras [:i<3] [haz "i :i+1 escribe :i]
1
2
3
```

## 13. OPERACIONES LÓGICAS

**Y *expresión1 expresión2***

Cuando son más de dos los argumentos adopta la forma:

(Y *expresión1 expresión2 expresión3 .....)*

Devuelve VERDADERO si las entradas (que son expresiones que pueden valer VERDADERO o FALSO) tienen **todas** valor VERDADERO. Si esto no se cumple devuelve FALSO.

Ejemplo:

```
haz “var 8
escribe y 3 < 5  :var > 6
verdadero
```

**O *expresión1 expresión2***

Cuando son más de dos los argumentos adopta la forma:

(O *expresión1 expresión2 expresión3 .....)*

Devuelve VERDADERO si **al menos una** de las entradas (que son expresiones que pueden valer VERDADERO o FALSO) tiene valor VERDADERO. Si esto no se cumple devuelve FALSO.

Ejemplo:

```
haz “var 8
escribe o 3 < 5  :var < 2
verdadero
escribe o 3 > 5 :var < 2
falso
```

**NO *expresión***

Devuelve VERDADERO si la entrada tiene el valor FALSO, y viceversa.

Ejemplo:

```
para acierta :var
(si no :var = 5 [escribe [el número es distinto]] ~
   [escribe “acertaste])
fin
acierta 8
el número es distinto
acierta 5
acertaste
```

## 14. LA RECURSIVIDAD

Se llama recursividad a la posibilidad que ofrece un programa diseñado en Logo de llamarse a sí mismo en un bucle sin fin. Un sencillo ejemplo lo constituye el siguiente programa, que dibuja una circunferencia de modo recursivo:

```
para cirec
av 1
gd 1
cirec
fin
```

Si ejecutamos el procedimiento observamos que se ejecuta por tiempo indefinido y en principio no tenemos forma de salir de él sin hacer clic en el botón ALTO.

Una forma de resolver el problema sería hacer que el procedimiento deje de ejecutarse cuando se cumpliera alguna condición, por ejemplo que se pulsara una determinada tecla. Para ello debemos valernos de la orden Ponteclado, que captura los eventos del teclado (para ello el foco debe estar situado en la Pantalla de MSWLogo, es decir, que ésta sea la ventana activa) y de alguna sentencia condicional como, por ejemplo, SI.

Modifiquemos el procedimiento anterior de la siguiente forma

```
para cirec
ponteclado [haz "stop car leecar]
ponfoco [pantalla de MSWlogo]
si :stop = 1 [haz "stop 0  alto]
av 1
gd 1
cirec
fin
```

Nota: las instrucciones ponteclado y ponfoco no tienen por qué estar dentro de procedimiento, bastaría con ejecutarlas antes de ejecutar ***cirec***.

Ahora al ejecutar ***cirec*** vemos cómo podemos salir del procedimiento pulsando la tecla 1. Obsérvese que ***stop*** es una variable global. Si ejecutamos el procedimiento nos dirá que STOP no tiene valor. Hemos de asegurarnos de darle algún valor a ***stop*** antes de ejecutar el procedimiento.

Hemos utilizado las primitivas ya conocidas:

- **Ponteclado** que captura los eventos del teclado y ejecuta la lista de instrucciones que le sigue. Para que funcione la Pantalla de MSWLogo debe tener el foco (activa) no la de Trabajo.
- **Ponfoco**, que deja que LOGO controle la ventana que tiene el foco.
- **Car** (abreviatura de CARACTER) devuelve el carácter que corresponde a un código ascii.
- **Leecar**, que devuelve el código ascii de la última tecla pulsada.
- **Si** que es una primitiva condicional. Si se cumple la prueba que le sigue ejecuta la lista de instrucciones que sigue a la prueba.
- **Alto**, que termina la ejecución del procedimiento en el que aparece.

## ¡¡¡Cuidado con la recursividad!!!

Salvo que se domine muy bien, hay que utilizar la recursividad con cuidado, pues pueden darse resultados inesperados, sobre todo cuando el procedimiento se llama a sí mismo cuando por debajo de esta llamada aún quedan más instrucciones.

Cuando la llamada a sí mismo del procedimiento recursivo es la última instrucción del mismo no se suelen presentar problemas.

## 15. MANEJO DE LA MEMORIA

En el espacio de trabajo de MSWLogo es donde se van guardando todas las variables y procedimientos que vamos creando. O sea, es la memoria del programa. Veamos algunas primitivas relacionadas con la consulta y eliminación de variables y procedimientos en la memoria.

**EDITATODO**

Este procedimiento edita todos los procedimientos y variables de la memoria.

**EDNS**

Edita todas las variables de la memoria.

**BON *nombrevar***

Puede adoptar también la forma:

BON *listanombrevar*

Borra de la memoria la variable cuyo nombre (precedido de comillas) se da en el argumento *nombrevar* o las variables cuyos nombres (encerrados entre corchetes sin comillas) se dan en la lista *listanombrevar*.