EJERCICIOS BÁSICOS DE MSWLOGO

(VERSIÓN 1.6)

☼

Luis González
Departamento de Tecnología
I.E.S. Santa Eugenia

Edición: 050514

INTRODUCCIÓN

Este manual nació de la necesidad de revisar la sintaxis y adaptar, a la versión más reciente de MSWLogo, los ejercicios propuestos en el excelente manual que acompaña a las tarjetas controladoras de Enconor, para usarlos en mis clases.

Debo agradecer por tanto, en primer lugar, a los amigos de Enconor la redacción del manual de usuario que, para muchos de nosotros, ha sido la primera incursión sistemática en el lenguaje Logo.

A medida que iba probando los procedimientos propuestos, los modificaba para corregir alguna pequeña inconsistencia del código, para hacerlos más claros y comprensibles o, simplemente, para ponerlos a mi gusto. Al hacerlo se me ocurrian nuevos disparates que iban engrosando mi colección de ejercicios.

Luego lo ajusté a la excelente versión de MSWLogo distribuida por el Observatorio Tecnológico del CNICE, que incluye ayuda en castellano y apoyo para manejar controladoras y simuladores de robótica para uso escolar. Desde aquí mi reconocimiento y agradecimiento al trabajo del Observatorio.

El manual ha sido utilizado en las clases de Tecnologías de la Información del Bachillerato, en el I.E.S. Santa Eugenia de Madrid. Mis alumnos han puesto a prueba todos los procedimientos propuestos y detectado los errores en la redacción de este manual. A todos ellos, por tanto, mi agradecimiento.

También quiero mostrar mi agradecimiento a algunos colegas, que han elaborado manuales semejantes para introducirse en el lenguaje Logo, y que me han proporcionado alguno de los ejercicios más *flipantes*: a Vera M. Rexach, J. P. Fuller y tantos otros. Gracias también a Rosario Ruiz por su traducción del archivo de ayuda, sin el cual estaría perdido.

A todos, gracias

Santa Eugenia (Madrid)

Luis González

INDICE:

MSWLOGO

Logo es un lenguaje de programación, desarrollado a finales de los años 70 por un equipo de matemáticos, dirigidos por Seymour Papert en la Universidad de California (Berkeley), con el fin de entrenar el pensamiento lógico.

Todas las versiones de Logo desarrolladas para Unix, DOS, Macintosh y Windows, son gratuitas y pueden ser copiadas y distribuidas, sin ningún tipo de restricción, con fines educativos. **MSWLogo** es la versión de Logo para Windows.

Originalmente, el lenguaje Logo estaba destinado a controlar el movimiento de un vehículo robotizado, con aspecto de tortuga, controlado desde un ordenador. En la actualidad, el icono de la tortuga se ha sustituido por un triángulo que se mueve en la pantalla del ordenador.

Cada vez que el programador introduce una orden como, por ejemplo, AVANZA 200, el icono de la tortuga se mueve obedeciendo la orden y dibujando, si el lápiz de la tortuga está bajado, una linea en la pantalla.

PRIMITIVAS

MSWLogo es un lenguaje interpretado, al igual que VisualBasic. Cuando se escribe código de programa en un lenguaje interpretado, cada una de las órdenes escritas es interpretada y ejecutada al momento, lo que permite ver inmediatamente si la instrucción escrita produce el efecto deseado.

Por el contrario, los lenguajes que deben ser compilados (como es el caso de C++, Fortran, Pascal, etc.), deben ser convertidos a lenguaje máquina, es decir, convertidos a unos y ceros, para que se puedan cargar en memoria y ser ejecutados.

> Las órdenes o instrucciones básicas de **MSWLogo** se llaman **primitivas**. Las primitivas deben escribirse en el cuadro inferior de la ventana de trabajo. Al pulsar Enter o hacer clic en el botón Ejecutar, la primitiva escrita se ejecuta. Si la primitiva está mal escrita o si le faltan datos, el intérprete constesta “no sé cómo...” Cada una de las órdenes queda anotada en la ventana de trabajo.

Un programa típico, para dibujar un triángulo equilátero, en lenguaje logo tiene el siguiente aspecto:

```
avanza 100
giraderecha 120
avanza 100
giraderecha 120
avanza 100
giraderecha 120
```

Las palabras **avanza** y **giraderecha** son **primitivas**, que indican a la tortuga que debe desplazarse hacia adelante o girar hacia la derecha. Los números que completan las instrucciones son los **argumentos**, que le indican cuánto debe avanzar o qué ángulo debe girar.

Para salir de **MSWLogo** basta con introducir la primitiva **adios** en la ventana de trabajo o seleccionar <**Fichero** / **Salir**> en el menú principal.

Veamos a continuación una pequeña muestra de las primitivas más sencillas de **MSWLogo** y de sus abreviaturas:

PRIMITIVA	ACCIÓN	ABREVIADO
AVANZA 150	La tortuga avanza el número indicado de puntos.	**AV 150**
RETROCEDE 80	Retrocede el número de puntos especificado.	**RE 80**
GIRADERECHA 15	Gira a la derecha el ángulo especificado en grados.	**GD 15**
GIRAIZQUIERDA 30	Gira a la izquierda el ángulo indicado.	**GI 30**
SUBELAPIZ	Levanta el lápiz y no pinta al moverse.	**SL**
BAJALAPIZ	El lápiz toca el papel y pinta cuando se mueve.	**BL**
GOMA	La tortuga borra a lo largo de su trayectoria.	
OCULTATORTUGA	Oculta el icono de la tortuga.	**OT**
MUESTRATORTUGA	Muestra el triángulo que representa la tortuga.	**MT**
BORRAPANTALLA	Borra la pantalla y sitúa la tortuga en el centro.	**BP**
CENTRO	Lleva la tortuga al centro de la pantalla sin borrarla.	
ROTULA [HOLA]	Escribe texto en la dirección de la tortuga.	**RO**
PONCOLORLAPIZ [3]	Establece el color con el que pinta.	**PONCL**
TONO [1000 200]	Emite un sonido de [frecuencia(Hz) duración(ms)]	
PONGROSOR [10 10]	Fija el grosor del trazo y la altura.	**PONG**
ADIOS	Sale de MSWLogo.	

PROCEDIMIENTOS.

Un procedimiento es un fragmento de programa, formado por una sucesión de primitivas. Todo procedimiento empieza con la palabra **para** que declara el nombre del procedimiento y termina con la palabra **fin**. Por ejemplo:

```
para procedimiento
primitiva 1
primitiva 2
.
etc.
fin
```

Una vez creado un procedimiento, puede ser invocado como una primitiva más del lenguaje.

VARIABLES.

> Una variable es un **dato**, que tiene un **nombre**, y que puede tomar distintos **valores**.

El programador puede crear una variable en cualquier momento, definiendo su **nombre** y asignándole un **valor**. Para ello utiliza la primitiva <**haz**>. Por ejemplo:

```
haz “variable 50
```

Las comillas señalan el comienzo del **nombre de la variable**.

En lo sucesivo, <**variable**> vale 50. Se puede utilizar este valor asignado a la variable, en un procedimiento, escribiendo dos puntos delante del nombre de la variable. Por ejemplo:

```
rotula :variable
```

CREAR PROCEDIMIENTOS

Primitivas <para> <fin> y <escribe>

1. Crea un procedimiento llamado <**tax**> que escriba el mensaje "Hola, mundo" **en la ventana de textos**.

```
para tax
escribe [Hola, mundo]
fin
```

Invoca ahora el procedimiento que has creado, escribiendo en la ventana de órdenes la palabra <**tax**>. ¿Funciona?

2. Modifica el procedimiento <**tax**> de forma que escriba el mensaje "Hola, mundo" y en la siguiente línea "¿Qué tal va todo?" en la pantalla de trabajo.

```
Fichero → editar → tax
para tax
escribe [Hola, mundo]
escribe [¿Qué tal va todo?]
fin
```

Invoca de nueco el procedimiento <**tax**> y comprueba que los cambios que has introducido se ejecutan correctamente.

Primitivas <rotula> <borrapantalla> <giraderecha> <giraizquierda>

3. Crea un procedimiento llamado <**mox**> que escriba el mensaje "Esto es un mensaje" en la pantalla de dibujo.

```
para mox
rotula [Esto es un mensaje]
fin
```

Invócalo, escribiendo **mox** en la línea de comandos de la ventana de trabajo. Notarás que el rótulo aparece en la dirección apuntada por la tortuga.

4. Modifica el procedimiento <**mox**> para que el rótulo se pueda leer horizontalmente, abriendo el menú:

```
Fichero → editar → mox
para mox
gd 90
rotula [Esto es un mensaje horizontal]
fin
```

Invócalo varias veces. Notarás que el mensaje aparece girado 90 grados respecto al anterior. Modifica el procedimiento <**mox**> para que el rótulo **siempre** aparezca **horizontal**.

```
Fichero → editar → mox
para mox
bp
gd 90
rotula [Este mensaje siempre es horizontal]
fin
```

Si te fijas, al incluir en el código del programa la orden bp (borrapantalla), cada vez que llamas al procedimiento <mox> la tortuga se sitúa en el centro de la pantalla y orientada hacia arriba. Por eso siempre rotula en dirección horizontal.

Primitiva <haz>

5. Crea un procedimiento llamado <**por**> que calcule el producto de 15 por 68 y lo presente en la pantalla gráfica:

```
para por
haz “numero1 15
haz “numero2 68
haz “resultado :numero1 * :numero2
bp gd 90
rotula [15 x 68 = ]
av 90
```

```
rotula :resultado
fin
```

Es importante que comprendas la diferencia entre el **nombre** de una variable y el **valor** de esa misma variable. Por ejemplo:

```
rotula  “hola        producirá    hola
rotula  [hola]       producirá    hola

haz  “hola 79
rotula  :hola        producirá    79
rotula  hola * 2     producirá    error
rotula  :hola * 2    producirá    158
```

Primitiva <mensaje>

6. Crea un procedimiento llamado <**sox**> que muestre en la pantalla un aviso pidiendo que lo aceptes o lo rechaces:

```
para sox
mensaje [Plan para esta tarde] [¿Quieres ir al cine?]
fin
```

7. Modifica el procedimiento <**sox**> para que pregunte otra cosa distinta.

```
fichero → editar → sox
para sox
mensaje [Plan para esta tarde] [¿Qué tal una siesta?]
fin
```

MANEJAR PROCEDIMIENTOS ALMACENADOS EN EL DISCO DURO

Primitivas <contenido> <borra> <guardar> <cargar>.

8. Guarda en el disco duro todos los procedimientos creados hasta ahora bajo el nombre <**Mis ejercicios.lgo**>

```
fichero → guardar como... → Mis ejercicios.lgo
```

Si obtienes un mensaje de error se debe a que estás utilizando una versión de MSWLogo anterior a la 6.5b y no puedes utilizar más de ocho caracteres para dar nombre a un archivo. En ese caso, guárdalo bajo un nombre más corto

```
fichero → guardar como → ejeruno.lgo
```

9. Cierra MSWLogo y busca en tu disco duro el archivo **Mis ejercicios.lgo** que acabas de guardar. Haz doble clic sobre el y observarás que se trata de un simple archivo de texto, que se puede editar con un editor de texto simple, como **Wordpad** o el **Cuaderno de notas** de Windows.

10. Inicia de nuevo MSWLogo, carga tu archivo y saca un listado de los procedimientos creados hasta este momento. y borra el procedimiento <**sox**>

    fichero → cargar → Mis ejercicios.lgo
    escribe contenido

11. Borra uno de los procedimientos contenidos en el archivo <**Mis ejercicios.lgo**>

    borra "sox
    escribe contenido

    Comprobarás que el procedimiento <**sox**> ha desaparecido de la lista

12. Borra todos los procedimientos contenidos en el archivo <**Mis ejercicios.lgo**>

    borra contenido
    escribe contenido

En este momento, el archivo <**Mis ejercicios.lgo**> ha sido vaciado de procedimientos. Como no queremos perderlos, NO VAMOS A GUARDAR LOS CAMBIOS EFECTUADOS.

fichero → salir → rechazar

13. Carga del disco nuevamente los procedimientos salvados anteriormente.

    fichero → cargar → Mis ejercicios.lgo
    escribe contenido

Comprueba que has recuperado la versión guardada de <Mis ejercicios.lgo> y que están disponibles todos los procedimientos que incluiste en ese archivo.

MANEJO DEL ENTORNO GRÁFICO

Primitivas <sl> <bl> <av> <re> <ot> <mt> <limpia> <ponpos>

14. Crea un procedimiento llamado <**trazos**> que dibuje una linea de trazos, de 100 pixel de largo. Al terminar el procedimiento, muestra la tortuga en el centro de la pantalla.

```
para trazos
limpia
sl ot
ponpos [-500 0]
gd 90
bajalapiz avanza 100 subelapiz avanza 100
bajalapiz avanza 100 subelapiz avanza 100
bajalapiz avanza 100 subelapiz avanza 100
bajalapiz avanza 100 subelapiz avanza 100
bajalapiz avanza 100 subelapiz avanza 100
bajalapiz avanza 100 subelapiz avanza 100
ponpos [0 0]
gi 90
mt
fin
```

Observarás que, por comodidad, está permitido usar abreviaturas como <**ot**> o <**mt**> en lugar del nombre completo de la primitiva <**ocultatortuga**> o <**muestratortuga**>, <**sl**> o <**bl**> en lugar de <**subelapiz**> o <**bajalapiz**>

15. Haz un procedimiento llamado <**cuadro**> para dibujar, en la pantalla gráfica, un cuadrado de 50 pixels de lado. La posición de partida será el centro de la pantalla.

```
para cuadro
bp bl
av 50 gd 90
av 50 gd 90
av 50 gd 90
av 50 gd 90
fin
```

16. Averigua las coordenadas de las cuatro esquinas de tu pantalla, haciendo avances exploratorios. Cuando las conozcas, escribe un procedimiento, llamado <**marco**> que dibuje un marco rectangular alrededor de la pantalla.

REITERACIÓN

Para evitar la escritura repetida de líneas de código idénticas, recurrimos a órdenes primitivas que indican al ordenador cuántas veces debe repetir una instrucción.

Primitivas <repite> <poncolorpapel> <poncolorlapiz> <pongrosor>

17. Modifica el procedimiento <**cuadro**> para dibujar un cuadrado de 50 pixels de lado, utilizando **muy pocas lineas de código**.

```
para cuadro
bp bl
repite 4 [av 50 gd 90]
fin
```

Los colores, tanto del fondo de la pantalla, como del trazo del lápiz, se fijan con las órdenes **poncolorpapel** (**poncp**) y **poncolorlapiz** (**poncl**) seguidos de un código de color.

Ese código puede ser un número entre 0 y 15, de modo que cada número se corresponde con un color, por ejemplo: poncl 1 para dibujar líneas azules.

También se puede emplear un conjunto de tres cifras entre corchetes, entre 0 y 255, que se corresponden con la cantidad de luz roja, verde y azul que compone el color deseado. Por ejemplo: poncp [207 176 89]

COLORES:

0		1		2		3		4		5		6		7	
8		9		10		11		12		13		14		15	

18. Haz un procedimiento, llamado <**hexa**>, que dibuje un hexágono de lado 120, con grosor 3 y de color marrón (8) sobre fondo carmin (12).

```
para hexa
bp bl
poncp 12
poncl 8
pong [3 3]
repite 6 [av 120 gd 60]
ot
fin
```

19. Haz un procedimiento llamado <**dode**> que dibuje un dodecágono de lado 40, con grosor 2 y de color azul (1) sobre fondo amarillo (6)

```
para dode
bp bl
poncp 14 poncl 1 pong [2 2]
repite 12 [av 40 gd 30]
ot
fin
```

Fíjate que, para dibujar cualquier polígono regular de 80 puntos de lado, el conjunto de órdenes es el mismo. La única variación entre un polígono y otro es el **número de lados**, que coincide con el número de repeticiones, y el **ángulo** que debe girar la tortuga.

Toma una hoja de papel, copia la tabla siguiente y rellena las celdas vacías con los valores adecuados:

Polígono	Lados	Orden que se repite
Triángulo	3	repite 3 [av 80 gd 120]
Cuadrado	4	repite 4 [av 80 gd 90]
Pentágono	5	
Hexágono	6	
Heptágono	7	
Octógono		
Eneágono		
Decágono		
Icosígono		

UTILIZAR PARÁMETROS PARA LA EJECUCIÓN DE PROCEDIMIENTOS

Se puede escribir un procedimiento de tal manera que se ejecute con el valor del parámetro **que acompaña** al nombre del procedimiento.

Una **variable** permite guardar un valor, tanto numérico como de texto. En cualquier momento se puede tomar el contenido de esa variable o cambiarlo por otro nuevo. Cuando damos un valor a una variable que ya existía, el valor antiguo se pierde.

20. Escribe un procedimiento capaz de dibujar un cuadrado cualquiera. El tamaño del cuadrado se especificará mediante la variable <**lado**> que acompañará al nombre del procedimiento cuando lo invoques.

```
para cuadro :lado
repite 4 [av :lado gd 90]
fin
```

Ahora invócalo para que dibuje una cuadrado de lado 100, tecleando <**cuadro 100**>. Prueba también a teclear <**cuadro 20**>, <**cuadro 78**> o <**cuadro 35**>. ¿Lo ves?. Te obedece.

Pensemos: ¿Sería posible construir un único procedimiento, que fuese capaz de dibujar cualquier polígono regular? Si, porque el ángulo que debe girar la tortuga puede deducirse fácilmente a partir del número de lados del polígono.

21. Escribe un procedimiento, de nombre <**poli**> seguido de la variable <**lados**>, que dibuje un polígono regular de 80 puntos de lado, con el número de lados indicado en la variable <**lados**>

```
para poli :lados
bp bl
poncp 12 poncl 8 pong [3 3]
haz “angulo 360/:lados
repite :lados [av 80 gd :angulo]
ot
fin
```

22. Modifica el procedimiento <**poli**> para que admita dos parámetros: el número de <**lados**> y la <**longitud**> de esos lados

```
para poli :lados :longitud
bp bl
poncp 12 poncl 0 pong [3 3]
haz “angulo 360/:lados
repite :lados [av :longitud gd :angulo]
```

```
ot
fin
```

23. Y, si somos capaces de dibujar un polígono cualquiera, ¿podremos dibujar un círculo?. Escribe un procedimiento, llamado <**aro**>, capaz de dibujar un círculo del radio que se le indique.

```
para aro :radio
haz "avance :radio*tan 1
repite 360 [av :avance gd 1]
fin
```

24. Ahora, para practicar la programación de reiteraciones con parámetros, modifica el procedimiento <**trazos**> para crear otro procedimiento llamado <**trazo** :longitud>, que dibuje en la pantalla una línea de trazos de **longitud** ajustable. Redacta un programa **con el menor número posible de líneas de código**, utilizando la repetición de una única orden. El procedimiento debería tener un aspecto como este:

```
para trazo :longitud
...
...
fin
```

Primitivas <rellena> <poncolorrelleno> <ponrumbo>

Las figuras creadas en el entorno gráfico pueden rellenarse de color, utilizando la primitiva <**rellena**>. El color de relleno se establece con la primitiva <**poncolorrelleno**>.

Para que el relleno funcione se deben cumplir algunos requisitos: la figura debe estar **cerrada**, la tortuga debe situarse **en el interior** de dicha figura y el lápiz debe estar levantado con la primitiva subelápiz <**sl**>.

25. Vamos a modificar el procedimiento <**cuadro**> para que el cuadrado que resulta quede relleno de color.

```
para cuadro :lado
bp bl
poncp 14 poncl 0 poncolorrelleno 10
repite 4 [ av :lado gd 90]
sl gd 45 av 3
rellena
ot
fin
```

USO DE PROCEDIMIENTOS YA CREADOS EN NUEVOS PROCEDIMIENTOS.

Un procedimiento creado con primitivas u otros procedimientos, puede ser utilizado **como una primitiva más** del lenguaje.

26. Haz un procedimiento, llamado <**penta**>, que dibuje en la pantalla gráfica un pentágono de cualquier longitud de lado. Posteriormente utiliza ese procedimiento ya creado, en uno nuevo que dibuje seis pentágonos, unidos entre sí por un vértice y formando una estrella.

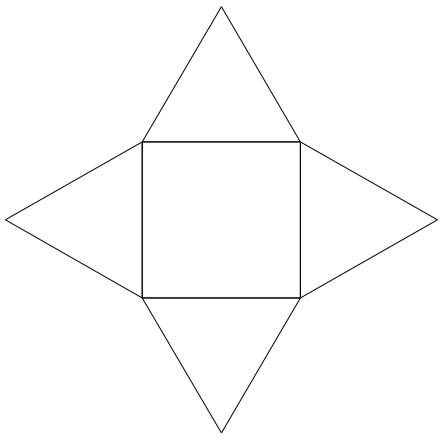
```
para penta :lado
repite 5 [av :lado gd 72]
fin

para estrella :lado
bp bl
poncp 10 poncl 0 pong [3 3]
repite 6 [penta :lado gd 60]
ot
fin
```

Fíjate que, en el código del procedimiento <**estrella**> aparece una llamada al procedimiento <**penta** :lado>, ¡como si fuese una primitiva más del lenguaje!

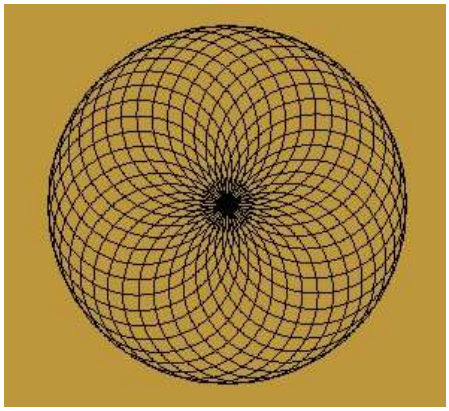
27. Guarda este gráfico, creado con MSWLogo, en una carpeta de tu disco duro con formato de mapa de bits **.bmp**. Para ello abre el menú
    bitmap → Guardar como... → Estrellapenta

Ábrelo con un editor de imágenes, como **Gimp** o **Pixia**, y comprobarás que puedes trabajar con él y exportarlo a otro formato. Insértalo también en un documento de texto como OpenOffice writer.

28. Escribe un procedimiento, llamado <**figura**>, que dibuje una figura compuesta por un cuadrado, de lado cualquiera, que tiene un triángulo equilátero en cada uno de sus lados, como el de la siguiente ilustración:



29. Aprovechando el procedimiento <**aro :radio**> que ya tienes creado, escribe un procedimiento llamado <**donut :radio**> que dibuje círculos consecutivos, pero de forma que cada vez que empieza a dibujar un círculo nuevo lo haga desviándose un cierto ángulo respecto al anterior, para llegar a formar una corona con aspecto de rosquilla como la de la ilustración siguiente:



Cuando veas aparecer el donut ante tus ojos podrás comprender, quizá, qué es la **programación elegante**: conseguir un resultado espectacular con muy pocas líneas de código.

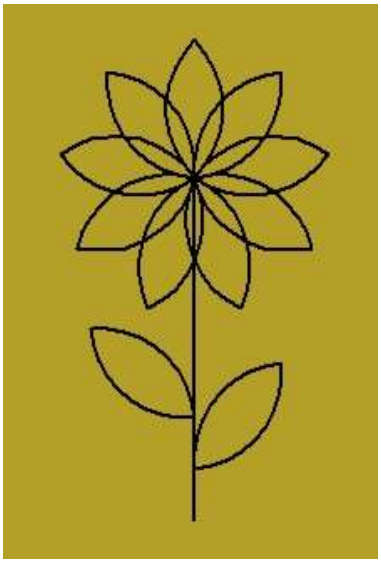
30. ¿Podrias dibujar una flor? Un método sencillo podría consistir en dibujar cada pétalo con un cuarto de círculo hecho con varios segmentos. Por ejemplo:

```
para cuarto :paso
repite 9 [av :paso gd 10]
fin
```

Dibujar un pétalo es fácil, con dos cuartos de círculo:

```
para petalo :paso
repite 2 [cuarto :paso gd 90]
fin
```

Ahora te toca a ti lo más fácil: escribe un procedimiento, llamado <**flor**> que sea capaz de dibujar una flor compuesta de varios pétalos, como esta:



COMENTARIOS EN EL CÓDIGO:

Es una buena norma completar con comentarios las lineas de código del programa que puedan resultar difíciles de entender o puedan suscitar alguna duda a un lector del código.

Para incluir comentarios en el código se utiliza el símbolo ; (punto y coma). Todo lo que se escriba a continuación del punto y coma no se ejecuta en el programa.

Primitivas <haz> <leepalabra> <lista> <leecar> <borratexto>

31. Haz un procedimiento, llamado <**nombre**>, que te pregunte tu nombre, lo guarde en una variable, y lo muestre en la ventana de trabajo con el formato TU NOMBRE ES: nombre.

```
para nombre                ; Comienza el procedimiento llamado <nombre>
bp sl ot gd 90             ; borra la pantalla y situa la tortuga en posición
rotula [¿Cómo te llamas?]
haz "name leepalabra       ; asigna la entrada tecleada a <name>
bp gd 90                   ; limpia la pantalla
rotula [¡Hola!, ]          ; escribe un texto de saludo y un espacio
ponpos [60 0]              ; se desplaza a la derecha
rotula :name               ; presenta el valor de la variable
ponpos [0 -25]             ; se desplaza a la línea inferior
rotula [¿Qué tal estás?]   ; completa el saludo
fin                        ; fin del procedimiento <nombre>
```

32. Haz un procedimiento, llamado <**sumar**>, que nos pida dos números y nos devuelva el resultado de sumarlos en la ventana de gráficos.

```
para sumar
bp gd 90 ot sl bt          ; inicializa las pantallas
ponpos [-200 0]            ; sitúa el puntero
rotula [¿Qué números quieres sumar?]
haz "num1 leepalabra       ; captura la primera variable
haz "num2 leepalabra       ; captura la segunda variable
bp gd 90                   ; limpia la pantalla
ponpos [-200 0]            ; sitúa el puntero al comienzo
rotula [LA SUMA DE ]
ponpos [-85 0]             ; de cada uno de los
rotula :num1
ponpos [-45 0]             ; fragmentos del mensaje
rotula [+]
ponpos [-30 0]             ; de salida
rotula :num2
ponpos [10 0]
rotula [= ]
ponpos [30 0]
```

```
rotula :num1 + :num2        ; calcula la suma y la presenta
fin
```

Primitivas <lista> <leelista> <primero>

Observa que, si intentas sumar números largos, se producen problemas al rotular cada una de las partes del mensaje de salida. Si quieres que la presentación del resultado sea más elegante, en una sola línea de código y sin tener que preocuparte de cuánto espacio necesitas para cada cifra, puedes conseguirlo utilizando la primitiva <**lista**>. El procedimiento <**sumar**> quedaría asi:

```
para sumar
bp gd 90 ot sl bt
ponpos [-200 0]
rotula [¿Qué números quieres sumar?]
haz "num1 leepalabra
haz "num2 leepalabra
bp gd 90
ponpos [-200 0]
rotula (lista "La "suma "de :num1 "+ :num2 "= :num1+:num2)
fin
```

Mucho más elegante, ¿no es cierto?

33. Haz un procedimiento llamado <**identidad**>, capaz de preguntarte tu nombre y luego saludarte escribiéndolo en la pantalla.

```
para identidad
bp gd 90 ot sl bt
rotula [¿Cuál es tu nombre y apellidos?]
haz "napel leelista
bp gd 90
rotula [¡Hola! ]
ponpos [70 0] rotula :napel
fin
```

34. Haz un procedimiento con el nombre <**inicial**>, que pregunte tu nombre y responda cuál es la letra inicial de tu nombre.

```
para inicial
bp ot sl gd 90
rotula [¿Cuál es tu nombre?]
haz "letra primero leepalabra
bp gd 90
ponpos [-100 0]
rotula [Tu nombre empieza por ]
ponpos [125 0]
```

```
rotula :letra
fin
```

35. Escribe un procedimiento capaz de convertir euros a pesetas

```
para pesetas
bp gd 90 sl ot
ponpos [-200 0]
rotula [¿De cuántos euros se trata?]
haz “euros primero leelista
haz “pesetas :euros *166.386
bp gd 90
ponpos [-200 0] rotula :euros
ponpos [-150 0 ] rotula [Euros son: ]
ponpos [-40 0] rotula :pesetas
ponpos [60 0] rotula [Pesetas.]
fin
```

Del mismo modo que hicimos en el procedimiento <**sumar**>, podemos resolver la presentación del resultado en una única línea de código con la primitiva <**lista**>:

```
para pesetas
bp gd 90 sl ot
ponpos [-200 0]
rotula [¿De cuántos euros se trata?]
haz "euros primero leelista
haz "pesetas :euros * 166.386
bp gd 90
ponpos [-200 0] rotula (lista :euros "Euros "son: :pesetas "Pesetas)
fin
```

CÁLCULOS MATEMÁTICOS

Primitivas <sen> <cos> <tan>

36. Haz un procedimiento, con el nombre de <**trigo**>, que nos pida el valor de un ángulo y nos devuelva el seno, el coseno y la tangente en la ventana de trabajo.

```
para trigo
bt                                  ; borra la pantalla de trabajo
haz "valor leepalabra
escribe (lista “Ángulo: :valor “grados)
escribe (lista “Su “seno “es: sen :valor)
escribe (lista “Su “coseno: cos :valor)
escribe (lista “Y “su “tangente: tan :valor)
fin
```

37. Ahora tú solo: escribe un procedimiento, llamado <**senoide :pico**>, capaz de dibujar en la pantalla la función $y=V_{max}\cdot sen\ (x)$ dándole el valor máximo de la función <**:pico**>

Primitivas <entero> <redondea> <raizcuadrada>

38. Haz un procedimiento que nos pida un número con decimales y nos devuelva la parte entera del mismo, el número redondeado y su raíz cuadrada, en la ventana de trabajo.

```
para numeros
bt
escribe [ESCRIBE UN NUMERO CON DECIMALES]
haz "valor leepalabra
escribe (lista "NUMERO "SOLICITADO: :valor)
escribe (lista "LA "PARTE "ENTERA "ES: entero :valor)
escribe (lista "SU "VALOR "REDONDEADO "ES: redondea :valor)
escribe (lista "SU "RAIZ "CUADRADA "ES: rc :valor)
fin
```

Primitivas <azar> <resto> <muestra> <espera>

39. Haz un procedimiento, llamado <**loteria**>, que genere dos números aleatorios entre el 0 y el 255 y los muestre en la ventana de trabajo. Además devolverá el resultado de dividir el primero entre el segundo, y su resto.

```
para loteria
bt                          ;borra la pantalla de trabajo
haz "num1 azar 255          ;genera al azar dos números menores que 255
haz "num2 azar 255
escribe (lista "Primer "número "aleatorio: :num1)
escribe (lista "Segundo "número "aleatorio: :num2)
escribe (lista "Su "cociente "es: :num1/:num2)
escribe (lista "Y "queda "un "resto "de: resto :num1 :num2)
fin
```

40. Escribe un procedimiento, llamado <**num10**>, que sea capaz de escribir 6 números aleatorios entre 0 y 10.

```
para num10
bt
repite 6 [muestra azar 11]
fin
```

41. Ahora escribe un procedimiento, al que llamaremos <**num25**>, que escriba ocho cifras elegidas al azar entre el 2 y el 5. Las cifras no deben ser menores que 2 ni mayores que 6

```
para num25
bt
```

```
repite 8 [haz "num1 azar 4
escribe :num1+2]
fin
```

PROCEDIMIENTOS RECURSIVOS

A veces nos puede interesar que un procedimiento se repita una y otra vez, indefinidamente. Para conseguirlo basta con incluir una línea de código, antes del comando <fin> que vuelve a llamar al propio procedimiento para que vuelve a empezar. Este tipo de procedimientos necesitan una orden <alto> para detenerse.

Primitiva <espera>

42. Por ejemplo, escribe un procedimiento llamado <**num :bajo :alto**>, que muestre **una vez cada segundo** un número elegido al azar, entre dos cifras límite.

```
para num :bajo :alto
bt
haz "lapso :alto-:bajo
haz "num1 azar :lapso+1
escribe :bajo+:num1
espera 60
num :bajo :alto
fin
```

Puedes detener la ejecución del procedimiento tecleando <**Alto**> en la botonera del área de comandos.

43. Otro ejemplo: hagamos un procedimiento que dibuje una línea en **zigzag**, indefinidamente:

```
para zigzag
ponrumbo 45
av 50 gd 90
av 50 gi 90
espera 10
zigzag
fin
```

CÓDIGO ASCII

El código ASCII establece la equivalencia entre cada uno de los símbolos de texto que pueden introducirse en el teclado, y un valor numérico binario de 8 bits (1 byte).

A la letra "A", por ejemplo, le corresponde el número 65 en binario determinado, a la "z" le corresponde el número 122, y así sucesivamente. También están incluidos los símbolos gráficos y los comandos no imprimibles, como el espacio o el retorno de carro.

Primitivas <ascii> y <caracter>.

44. Haz un procedimiento llamado <**tabasci**>, que genere una tabla entre el 0 y el 255 en la que aparezca el valor decimal y su correspondiente código ASCII. Se mostrará en la ventana de trabajo.

```
para tabasci
bt                                      ; Borra la pantalla de trabajo
haz "x 0                                ; Crea un contador y lo pone a 0
repite 255 [escribe (lista :x caracter :x)   ; Escribe un número y el carácter
haz "x :x+1]                            ; equivalente en ascii
fin
```

45. ¿Te sientes capaz de escribir un procedimiento, llamado <**tabasci2**> que, en lugar de escribir los caracteres ASCII en una sola columna en el cuadro de texto, los presente rotulados en la pantalla gráfica en forma de tabla? Inténtalo

46. Escribe ahora un procedimiento que nos pida una letra y devuelva su código ASCII en la pantalla.

```
para traduce
bt                                  ; borra la pantalla de trabajo
escribe [INTRODUCE UNA LETRA EN LA VENTANA]
haz "valor leepalabra
bt
escribe (lista "LETRA: :valor)
escribe (lista "Su "valor "ASCII "es: ascii :valor)
fin
```

EJECUCIÓN SECUENCIAL Y EJECUCIÓN CONDICIONADA.

En algunos casos, para escribir el código de un programa, basta con escribir una serie ordenada de primitivas, que se ejecutarán una detrás de otra (ejecución secuencial).

Pero es frecuente también que en un programa haya que ejecutar unas primitivas u otras en función de alguna condición. De esta forma la ejecución del programa se ramificará por unas primitivas u otras, según se vayan dando determinadas condiciones (ejecución condicionada).

Primitivas <si> y <sisino>

La primitiva **si** permite ordenar al programa que ejecute una acción, en función de que una condición se cumpla o no. La sintaxis de la primitiva **si** es la siguiente:

si condición accion

Si la condición es verdadera ejecuta la acción. Si la condición es falsa no hace nada. Por ejemplo:

si 1>0 escribe “cierto

47. Haz un procedimiento llamado <**numpos**> que nos pida un número. Si el número que introducimos es positivo se mostrará su raíz cuadrada. Si es negativo no debe calcularlo, ya que daría un error.

```
para numpos
bt
escribe [INTRODUCE UN NÚMERO EN LA VENTANA]
haz "valor leepalabra
bt
escribe (lista "El "número "es: :valor)
si :valor>0 [escribe (lista "Su "raiz "cuadrada "es: rc :valor)]
fin
```

La primitiva **si** también permite ordenar al programa que ejecute una acción u otra, en función de que una condición se cumpla o no. En este caso, la sintaxis de la primitiva **si** es la siguiente:

si condición [accion1] [accion2]

Si la condición es **verdadera** ejecutará la acción1. Si la condición es **falsa** ejecutará la acción 2. Por ejemplo:

48. Escribe un procedimiento, llamado <**posneg**>, que nos pida un número. Si el número que introducimos es positivo se mostrará su raíz cuadrada. Si es negativo se escribirá un mensaje que diga "NO SE PUEDE CALCULAR LA RAIZ DE UN NEGATIVO".

```
para posneg
bt
escribe [INTRODUCE UN NÚMERO EN LA VENTANA]
haz "valor leepalabra
bt
escribe (lista "El "número "es: :valor)
si :valor>0 [escribe (lista "Su "raiz "cuadrada "es: rc :valor)]  [escribe (lista "No
"tiene "raiz, "es "un "número "negativo)]
fin
```

La sintaxis de la primitiva **si** exige que las dos acciones estén **en la misma línea de código**. Hay una primitiva que se comporta de un modo semejante: se llama **sisino**. Su sintaxis es similar:

sisino condición [accion1] [accion2]

49. Escribe un procedimiento, llamado <**santiago**>, que formule una pregunta y que informe si la respuesta es correcta o no lo es.

```
para santiago
bt bp sl ot gd 90 poncp 14
ponpos [-50 100]
rotula [¿De qué color es el caballo blanco de Santiago?]
haz "respuesta leepalabra
limpia ponpos [-50 100]
sisino :respuesta="blanco [rotula (lista "¡!Muy "bien!! "¡Acertaste! "  "Es "blanco)]
[rotula (lista "¡¡Cuánto "lo "siento!! "Fallaste. "  "No "es :respuesta)]
fin
```

50. Ahora, y para terminar con las prácticas de ejecución condicionada, escribe un procedimiento llamado <**elegir**> que te ofrezca la posibilidad de elegir entre cuatro tareas distintas. Por ejemplo, que te pregunte qué tipo de figura quieres que dibuje:
    a) Un cuadrado rojo
    b) Un círculo azul
    c) Un triángulo amarillo
    d) Un pentágono blanco

Si respondes <a> dibujará un cuadrado rojo; si tecleas <b> deberá dibujar un círculo azul, y así sucesivamente. Si tecleas cualquier otra tecla por error te lo debe advertir y negarse a hacer ninguna de las cuatro tareas previstas. Procura que las figuras sean de tamaño semejante, rellenas de color y sobre un fondo del mismo color.

Primitivas lógicas <o> <no> <y> <bity> <bito> y <bitinverso>

Naturalmente, un lenguaje de programación debe servir para manejar funciones lógicas básicas: Y, O, OEXCLUSIVA, etc.

La función lógica **y** evalúa varias expresiones. Devuelve el valor **cierto** si **todas** las expresiones evaluadas tienen el valor **cierto** y devuelve el valor **falso** si **alguna** de las expresiones es falsa. Su sintaxis es como sigue:

y expresion1 expresion2	si debe evaluar 2 expresiones
(**y** expresion1 expresion2 expresion3 ...)	si debe evaluar más de 2 expresiones

51. Por ejemplo: escribe un procedimiento llamado <**logica**>, que nos pida dos números. Si los dos números son mayores de 100, nos devolverá la suma de ambos. Si es alguno de ellos es igual o menor que 100 nos devolverá un mensaje de error.

```
para logica
bp gd 90 sl ot ponpos [-200 0]
rotula (lista "Introduce "dos "números "mayores "que "100)
haz "num1 leepalabra
haz "num2 leepalabra
bp gd 90 ponpos [-200 0]
sisino y :num1>100 :num2>100 [rotula (lista "La "suma "de :num1 "y :num2 "es:
:num1+:num2)] [rotula (lista "¡¡¡Error!!! "Uno "de "los "dos "números "es "menor
"que "100)]
fin
```

La función lógica **o** evalúa varias expresiones. Devuelve el valor **cierto** si **alguna** de las expresiones evaluadas tienen el valor **cierto** y devuelve el valor **falso** si **todas** las expresiones son falsas. Su sintaxis es como sigue:

o expresion1 expresion2	si debe evaluar 2 expresiones
(**o** expresion1 expresion2 expresion3 ...)	si debe evaluar más de 2 expresiones

52. Para comprenderlo, vamos a redactar un procedimiento llamado <**logica2**> que nos pida dos números, uno de los cuales **debe** ser mayor que 500. Si uno o los dos números son mayores que 500, nos devolverá la suma de ambos. Si los dos números son iguales o menores que 500, nos devolverá un mensaje de error.

```
para logica2
bp gd 90 sl ot ponpos [-200 0]
rotula (lista "Introduce "dos "números. "Uno "de "ellos "debe "ser "mayor "que
"500)
haz "num1 leepalabra
haz "num2 leepalabra
bp gd 90 ponpos [-200 0]
sisino o :num1>500 :num2>500 [rotula (lista "La "suma "de :num1 "y :num2 "es:
:num1+:num2)] [rotula (lista "¡¡¡Error!!! "Ambos "números "son "menores "que
"500)]
fin
```

La primitiva **bity** devuelve el resultado de aplicar la función lógica **Y** sobre dos o más números, que deben ser números enteros. Su sintaxis es como sigue:

bity numero1 numero2	Si debe hacer el cálculo con 2 números
(**bity** numero1 numero2 numero3 ...)	Si debe calcularlo con más de 2 números

La primitiva **bito** devuelve el resultado de aplicar la función lógica **O** sobre dos o más números enteros. Su sintaxis es similar a la de la primitiva bity

53. Para practicarlo, escribe un procedimiento llamado <**funcionyo**>, que nos pida dos números enteros y nos devuelva el resultado de hacer la función lógica <**Y**> entre ellos, y la función lógica <**O**>.

```
para funcionyo
bp gd 90 sl ot
ponpos [-50 0]
rotula (lista "Introduce "dos "números "enteros.)
haz "num1 leepalabra
haz "num2 leepalabra
bp gd 90
ponpos [-50 0]
rotula (lista "Los "números "introducidos "son: :num1 "y :num2)
ponpos [-50 -20]
rotula (lista "El "resultado "de "aplicar "la "función "Y "es: bity :num1 :num2)
ponpos [-50 -40]
rotula (lista "El "resultado "de "aplicar "la "función "O "es: bito :num1 :num2)
fin
```

Primitivas <prueba> <sicierto> <sifalso>

Estas tres primitivas están vinculadas entre si. La primitiva **prueba** evalúa una expresión y recuerda si el resultado ha sido **cierto** o **falso**, para que luego puedan utilizarlo las otras dos primitivas: **sicierto**, que puede abreviarse como **sic** y **sifalso**, que puede abreviarse como **sif**.

La sintaxis del conjunto es como sigue:

```
prueba expresion
sic [acciones a realizar]
sif [acciones a realizar]
```

54. Veamos un ejemplo, con un procedimiento, llamado <**sorteo50**> que genera un número aleatorio entre el 0 y el 100. Si el número generado es menor que 50 aparecerá un mensaje indicándolo. Si el número es igual o mayor que 50 aparecerá otro mensaje que nos lo indica.

```
para sorteo50
bp gd 90 sl ot
haz "num azar 100
ponpos [-50 0]
rotula (lista "El "número "agraciado "es: :num)
ponpos [-50 -20]
prueba :num<50
sic [rotula (lista "que "es "menor "que "50)]
sif [rotula (lista "que "es "mayor "que "50)]
fin
```

CONCEPTO DE BUCLE EN UN PROGRAMA

Un bucle en un programa es una parte del mismo que se ejecuta repetidamente un número determinado de veces. Este número puede ser fijo o depender de determinadas circustancias.

Primitivas <mientras> <hasta> <haz.mientras> <haz.hasta>

Las primitivas **mientras** y **hasta** dirigen la ejecución de una o varias instrucciones mientras una expresión sea cierta. Su sintaxis es como sigue:

mientras [expresion] [instrucciones]

hasta [expresion] [instrucciones]

Lo primero que hacen es evaluar si la expresión da como resultado **cierto**, por lo que las instrucciones podrían no ejecutarse nunca si la evaluación de las expresiones arrojase como resultado **falso**.

55. Por ejemplo, haz un procedimiento, llamado <**tablamul**>, que genere un número aleatorio entre el 0 y el 30. Una vez elegido ese número, el programa debe presentarlo en la pantalla y, además, presentará en la ventana de texto la tabla de multiplicar por 7, desde el 0 hasta el número generado al azar.

```
para tablamul
bp bt gd 90 sl ot
haz "num azar 30                ;elige un número al azar menor que 30
rotula (lista "Número "elegido: :num)
escribe (lista "TABLA "DEL "SIETE "DE "0 "A :num)
escribe [ ]                     ;línea en blanco
haz "contador 0                 ;inicializa un contador
mientras [:contador<:num+1] [escribe (lista :contador "x "7 "= :contador * 7)
haz "contador :contador + 1]
fin
```

56. Para redundar en la ejecución de tareas **hasta** que se cumpla una condición, vamos a hacer un procedimiento, llamado <**tablatrigo**>, que calcule los valores del seno y el coseno de un ángulo, desde 0 a 120 grados, y los presente en la pantalla en forma de tabla ordenada.

```
para tablatrigo
bp gd 90 sl ot
poncp [245 196 127]
haz "angulo 0
haz "x -400
haz "y 300
hasta [:angulo>120] [ponpos (lista :x :y) rotula (lista "Ángulo: :angulo) ponpos
(lista :x+150 :y) rotula (lista "Seno: sen :angulo) ponpos (lista :x+450 :y) rotula
(lista "Coseno: cos :angulo) haz "angulo :angulo+5 haz "y :y-20]
fin
```

Es posible que tengas que retocar las coordenadas iniciales “x e “y para que la tabla quede correctamente centrada en tu pantalla.

Las primitivas **haz.mientras** y **haz.hasta** son semejantes: ejecutan un grupo de instrucciones **mientras** una condición sea cierta o **hasta** que una condicion sea cierta. Su sintaxis es como sigue:

haz.mientras [instrucciones] [condición]

haz.hasta [instrucciones] [condición]

Por ejemplo:
```
haz "A 0
haz.mientras [haz "A :A+1 escribe :A] [:A<3]
```
Produce:
```
1
2
3
```

Primitiva <formatonumero>

57. Haz un procedimiento llamado <**formato203**> que genere una lista de números entre el 5,00 y el 15,00 con **dos** decimales y con un incremento entre ellos de **0,03**.

```
para formato203
bt
escribe [NÚMEROS DE 5 A 15]
escribe [INCREMENTO: 0.03]
escribe [ ]
haz "num1 5.00
haz.hasta [escribe formatonumero :num1 5 2 haz "num1 :num1+0.03][:num1 > 15]
fin
```

58. Haz un procedimiento, llamado <**multipi**>, que genere la lista de múltiplos de π (3,141592), comprendidos entre el 0 y el 150, con una precisión de tres decimales. La serie de números debe aparecer en la ventana de texto con una cadencia de 0,5 segundos entre cada número presentado.

```
para multipi
bt
escribe [MÚLTIPLOS DE PI 0<n<150]
escribe [ ]
haz "num 0
haz.mientras [escribe formatonumero :num 7 3 haz "num :num + 3.141592654
espera 30] [:num<150]
fin
```

PASO DE ARGUMENTOS ENTRE PROCEDIMIENTOS

Se puede realizar un procedimiento que necesite algún dato previo para poder realizar alguna operación con él y luego nos devuelva el dato o los datos del resultado.

Estos datos que se pasan al procedimiento, en el momento de llamarlo o ejecutarlo o que devuelve el procedimiento al finalizar su ejecución, se denominan argumentos.

Primitivas <devuelve> <alto> <local> <lvars>

59. Haz un procedimiento llamado <**multiplica**>, que reciba dos números y devuelva el producto de ambos a otro procedimiento que lo llame.

```
para prod :x :y   ;Se inicia un procedimiento con dos datos de entrada
haz "z  :x * :y
devuelve :z       ; Al finalizar devuelve z al procedimiento que lo llamó
fin

para multiplica
bt
escribe [Introduce los factores]
haz "x leepalabra
haz "y leepalabra
bt
escribe [El resultado es:]
escribe (lista :x "x :y "= prod :x :y) ; Llama a <prod> y le da los datos.
                                       ; De <prod> recibe un dato y lo presenta
fin
```

60. Haz un procedimiento llamado <**rectanx3**>, que reciba un número y dibuje un rectángulo de lado horizontal 3 veces el vertical. La altura del lado vertical será el número recibido. Las variables que utilice dicho procedimiento serán de carácter local.

```
para rectanx3 :lado
bp bl pong [2 2]
local "ancho
haz "ancho :lado * 3
repite 2 [av :lado gd 90 av :ancho gd 90]
ot
fin
```

PROGRAMACIÓN ORIENTADA A OBJETOS

La técnica de programación orientada a objetos consiste, a diferencia de la programación estructurada mediante procedimientos, en organizar la interacción con el usuario a través de objetos que aparecen en pantalla. Son objetos característicos las ventanas, botones, casillas de selección, etc.

Los objetos son los elementos básicos de la interfaz gráfica de usuario **GUI**[1]

Primitivas <ventana> y <estatico>

Una ventana es un área de trabajo, típica del entorno Windows, dotada de una barra de título, sobre la que aparecerán otros objetos para facilitar la interacción del usuario. Una ventana puede arrastrase por la pantalla, maximizar, minimizar, borrar, etc.

61. Crea tu primera **ventana** tecleando la siguiente orden:

    creaventana " "primera [Mi primera ventana] 90 50 120 80 [ ]

    Observa que la ventana tiene un nombre (primera), su **barra de título** muestra el texto <Mi primera ventana>, su posición respecto a la esquina superior izquierda (90 50) y su tamaño (120 x 80). La ventana ha sido creada en la ventana principal de la pantalla (“)

    Puedes desactivar la ventana o volverla a activar con las órdenes:

    habilitarventana “primera “falso
    habilitarventana “primera “verdadero

62. Ahora borra la ventana, utilizando la orden siguiente:

    borraventana “primera

    Para saber más sobre la sintaxis de las ventanas consulta la ayuda.

63. Puedes crear **textos estáticos**, en el interior de una ventana, que se mueven con ella. Por ejemplo, si está creada y activa la ventana “**primera** y tecleas

    creaestatico “primera “texto1 “PAN 5 20 15 9

    verás aparecer un texto en la ventana “primera, llamado “texto1, que contiene la palabra (PAN), está situado en las coordenadas (5 20) desde la esquina superior izquierda de la ventana y tiene un tamaño de (15 9).

    Vamos a introducir un nuevo texto estático con la orden:

    creaestatico “primera “texto2 “SARDINAS ASADAS 5 30 15 9

    observarás que las nuevas coordenadas (5 30) son adecuadas, pero las dimensiones del texto son insuficientes y deberás aumentarlas, por lo menos, hasta (65 9)

64. Para borrar los textos estáticos debes emplear las órdenes:

    borraestatico “texto1
    borraestatico “texto2

1Graphic User interface

65. Para practicar la creación de ventanas y la introducción de textos estáticos, escribe un procedimiento llamado <**resultado**>, que reciba dos números y devuelva el producto de ambos en una nueva ventana.

```
para resultado
bp gd 90 ot sl
ponpos [-100 100]
rotula [Introduce dos números]
haz "num1 leepalabra
haz "num2 leepalabra
haz "resul :num1 * :num2
bp
creaventana "root "calculo [    Resultado del producto: ] 200 150 100 40 []
creaestatico "calculo  "resul1 :num1 5 5 20 10
creaestatico "calculo  "resul2 [x] 25 5 10 10
creaestatico "calculo  "resul3 :num2 35 5 20 10
creaestatico "calculo  "resul4 [=] 55 5 10 10
creaestatico "calculo  "resul5 :resul 65 5 20 10
fin
```

Si te fijas, la posición de cada uno de los textos estáticos está calculada para números de tamaño medio. Pero, tanto si se introducen números muy pequeños, cmo si los números son muy grandes, las coordenadas fijas de los textos producen una presentación pobre y desajustada.

La presentación mejora al reemplazar las cinco primitivas **creaestático** por una sola línea de código que llama a una lista:

```
para resultado
bp gd 90 ot sl
ponpos [-100 100]
rotula [Introduce dos números]
haz "num1 leepalabra
haz "num2 leepalabra
haz "resul :num1 * :num2
bp
creaventana "root "calculo [Resultado del producto:] 200 150 100 40 [ ]
creaestatico "calculo  "resul1 (lista :num1 "x :num2 "= :resul) 5 10 90 10
fin
```

Haz una prueba: deshabilita la ventana “**calculo** y verás que no puedes manejarla con el ratón. Si la habilitas de nuevo podrás activarla y arrastrarla por la pantalla.

Para despejar la pantalla, debes teclear la orden:

borraventana “calculo

Primitivas <boton>

Un botón es un elemento que permite manipular un programa mediante el ratón, que puede hacer clic en él o, simplemente, pasar sobre él.

66. Escribe un procedimiento que cree una ventana, llamada <segunda>, en la que haya un botón para cerrarla.

```
Para segunda
creaventana " "segunda [Pulsa Aceptar para cerrar] 200 50 90 40 [ ]
creaboton "segunda "boton1 [Aceptar] 30 5 30 20 [sesamo]
fin
```

Observa que la longitud de la ventana (90) ha de ser suficiente para que quepa el texto (pulsa aceptar para cerrar). La altura de la ventana ha de bastar para que quepa el botón, que tiene 30 x 20. Las coordenadas de la posición del botón pretenden que quede centrado en la ventana. Y ahora, crea el procedimiento <**sesamo**>, que se encargará de cerrar la ventana:

```
para sesamo
borraventana "segunda
fin
```

67. Haz un procedimiento, con el nombre <**ven3boton**>, que dibuje en pantalla una ventana con tres botones. Al pulsar el primero aparecerá un mensaje en la misma ventana que diga <**pulsado**>. Al pulsar el segundo se dibuja un donut en la ventana gráfica y el texto del botón cambia a <**DONUT**>. Al pulsar el tercero se cierra la ventana con todo su contenido.

```
para ven3boton
creaventana " "nueva [Ventana con tres botones] 170 50 190 70 [ ]
creaboton "nueva "primero [Pulsar] 15 30 40 20 [boton1]
creaboton "nueva "segundo [Arrancar] 67 30 50 20 [boton2]
creaboton "nueva "tercero [Cerrar] 130 30 40 20 [borraventana "nueva poncp 7]
fin

para boton1
creaestatico "nueva "boton1 [PULSADO] 17 10 35 15
fin

para boton2
actualizaboton "segundo [DONUT]
donut 50
ocultatortuga
fin
```

¡Fíjate qué ahorro!. En el tercer botón se incluye ya el código de lo que debe hacer al ser pulsado: borrar la ventana. No es necesario un procedimiento adicional.

Primitivas <barradesplazamiento>

Una barra de desplazamiento es un objeto que tiene un cursor que puede desplazarse mediante el ratón, para dar diversos valores a una variable.

68. Escribe un procedimiento que cree una ventana, llamada <tercera>, en la que haya un botón para cerrarla y una barra de desplazamiento.

```
Para tercera
creaventana " "tercera [Mi primera barra deslizante] 200 50 140 60 [ ]
creabarradesplazamiento "tercera "cien 30 10 80 10 [ ]
ponbarradesplazamiento "cien 0 100 50
creaboton "tercera "close [Cerrar] 55 25 30 20 [borraventana "tercera]
fin
```

Observa que la longitud de la ventana (140) ha de ser suficiente para que quepa el título (Mi primera barra deslizante) y la barra, que mide 80 de largo. La altura de la ventana ha de bastar para que quepan tanto la barra (80 x 10) como el botón (30 x 20). Las coordenadas de la posición del botón y de la barra pretenden que queden centrados en la ventana. Comprueba que puedes mover el cursor con el ratón y que puedes cerrar la ventana.

69. Aplica lo que has aprendido, escribiendo un procedimiento llamado <**venbar**>, que dibuje en pantalla una barra de desplazamiento dentro de una ventana. La barra tendrá un rango de valores entre 50 y 250. Cada vez que se mueva se debe visualizar en la ventana el valor de su posición y su raíz cuadrada. Tendremos, además, un botón que sirva para cerrar la ventana.

```
para venbar
creaventana " "ultima [Calculadora automática de raices] 150 70 180 70 [ ]
creaestatico "ultima "estat [150] 140 10 20 15
creaestatico "ultima "estat1 [Su raiz cuadrada es:] 15 30 70 10
creaestatico "ultima "estat2 raizcuadrada 150 35 40 30 10
creabarradesplazamiento "ultima "barra 15 10 110 10 [leebarra]
ponbarradesplazamiento "barra 50 250 150
creaboton "ultima "primero [Cerrar] 125 30 40 20 [borraventana "ultima]
fin
```

La barra de desplazamiento adoptará el valor inicial de 150 (en el centro de su recorrido). Por lo tanto, el estático <**estat2**> deberá ener como valor inicial la raiz cuadrada de 150 (señalado en amarillo). Cuando el cursor de la barra se mueva se ejecutará el procedimiento llamado <**leebarra**> que habrá que escribir:

```
para leebarra
haz "x leebarradesplazamiento "barra
haz "y raizcuadrada :x
actualizaestatico "estat :x
```

```
actualizaestatico "estat2 :y
fin
```

Primitivas <dialogo>

Una ventana de dialogo se utiliza como marco de trabajo en el que se pueden añadir otros objetos o controles como botones, barras de desplazamiento, listboxes, etc. Este objeto es similar a una ventana, excepto que **no devolverá el control al procedimiento llamante hasta que la ventana de diálogo se cierre**.

70. Vamos a crear una ventana de diálogo que pida al usuario que acepte pulsando un botón. Pero, en primer lugar, vamos a crear el procedimiento que cerrará la ventana al pulsar el botón:

```
para vendialo
creaboton "dialogo1 "botvale "ACEPTAR 25 10 40 20 [borradialogo "dialogo1]
fin
```

Ahora que está preparado teclea, en la ventana de trabajo, la orden siguiente:

```
creadialogo " "dialogo1 [El sistema debe reiniciarse] 120 60 90 50 [vendialo]
```

Naturalmente, una vez pulsado el botón pueden ponerse en marcha procedimientos más ambiciosos que el simple cierre de la ventana de diálogo.

71. Escribe un procedimiento que pregunte al usuario, mediante una ventana de diálogo, cuántos dias faltan para su cumpleaños:

```
para faltan
creaestatico "cumple "fecha [180] 155 10 20 15
creabarradesplazamiento "cumple "deslizadias 15 10 130 10 [haz "dias leebarra-
desplazamiento "deslizadias actualizaestatico "fecha :dias]
ponbarradesplazamiento "deslizadias 0 365 180
creaboton "cumple "ok [Aceptar] 70 30 40 20 [borradialogo "cumple]
fin
```

y teclea a continuación:

```
creadialogo " "cumple [¿Cuántos dias faltan para tu cumpleaños?] 120 60 180
70 [faltan]
```

Para practicar introduce alguna novedad de tu propia cosecha. Por ejemplo, que rotule al cerrar el diálogo, sobre una pantalla roja y con letras grandes, cuántos dias faltan.

Primitivas <groupbox> y <botonradio>

Un botón radio es un objeto que permite seleccionar una opción con dos estados: verdadero (botón seleccionado) o falso (botón no seleccionado).

Todos los botones radio, que se corresponden con cada una de las opciones que se presentan al usuario, deben estar asociados en un grupo o groupbox.

72. Para practicar, escribiremos el código de una ventana llamada <**cuarta**>, en la que el usuario deberá elegir su rango de edad seleccionando uno de los botones radio que se presentan reunidos en un groupbox.

```
para edad
creaventana " "cuarta [Seleccione su grupo de edad] 200 50 115 65 [ ]
creagroupbox "cuarta "marco 5 5 65 40
creabotonradio "cuarta "marco "bot1 [<15 años] 10 13 40 9
creabotonradio "cuarta "marco "bot2 [15 a 18 años] 10 23 55 9
creabotonradio "cuarta "marco "bot3 [>18 años] 10 33 40 9
escribebotonradio "bot1 "verdadero
creaboton "cuarta "cierre [Aceptar] 75 20 30 20 [borraventana "cuarta]
fin
```

Fíjate bien, todas las coordenadas están referidas a un origen común situado en la esquina superior izquierda de la ventana. La creación de cada objeto incluye una ruta compuesta por la ventana padre, el objeto padre y su propio nombre.

Sólo puede estar seleccionado uno de los botones radio de un groupbox. Cuando el usuario selecciona una de las opciones lo pone en estado “verdadero y, automáticamente, el resto de las opciones se ponen en estado “falso. Además, al iniciar el groupbox, una de las opciones declararse verdadera.

73. Haz un procedimiento llamado <**calcacme**>, que pida dos números y dibuje en una ventana un grupo con cuatro botones radio. Cada botón tendrá asignada una operación matemática. Cuando pulsemos un botón aparecerá el resultado de aplicar la operación matemática seleccionada a los dos números introducidos. Tendrá, además, otro botón que permita cerrar la ventana:

```
para calcacme
bp sl gd 90 ot
ponpos [-450 150]
rotula (lista "Introduce "dos "números "en "la "ventana)
haz "num1 leepalabra
haz "num2 leepalabra
bp
creaventana " "ultima [Calculadora aritmética marca "ACME"] 150 70 180 100 []
creagroupbox "ultima "grupo 10 10 160 50
creabotonradio "ultima "grupo "boton1 [ +] 15 15 25 10
creabotonradio "ultima "grupo "boton2 [ -] 15 25 25 10
creabotonradio "ultima "grupo "boton3 [x] 15 35 25 10
creabotonradio "ultima "grupo "boton4 [/] 15 45 25 10
```

```
escribebotonradio "boton1 "verdadero
creaboton "ultima "calcular [CALCULAR] 20 67 45 15 [calculo]
creaboton "ultima "cerrar [ACABAR] 115 67 45 15 [borraventana "ultima]
creaestatico "ultima "estat [NUMEROS: ] 50 25 45 15
creaestatico "ultima "estat1 :num1 110 25 20 15
creaestatico "ultima "estat2 :num2 140 25 20 15
creaestatico "ultima "estat3 [RESULTADO:] 50 40 50 15
creaestatico "ultima "estat4 [ ] 110 40 30 15 ; Se crea un estático sin mensaje
fin
```

```
para calculo
haz "x leebotonradio "boton1
si :x= "verdadero [haz "num3 :num1 +:num2]
haz "x leebotonradio "boton2
si :x= "verdadero [haz "num3 :num1-:num2]
haz "x leebotonradio "boton3
si :x= "verdadero [haz "num3 :num1 *:num2]
haz "x leebotonradio "boton4
si :x= "verdadero [haz "num3 :num1/:num2]
actualizaestatico "estat4 :num3
fin
```

Primitivas <checkbox>

Un objeto **checkbox** es una casilla que el usuario puede seleccionar. Todas las casillas **checkbox** deben estar reunidas en un groupbox. El usuario puede seleccionar varias casillas simultáneamente.

74. Haz un procedimiento, llamado <**vencom**>, que dibuje en una ventana un grupo con tres botones de tipo **checkbox**. Cada uno tendrá como etiqueta un número. Cuando pulsemos un botón aparecerá un mensaje indicando el estado de cada uno de ellos. Tendrá, además, otro botón que permita cerrar la ventana.

```
para vencom
creaventana " "ultima [Ventana con casillas de comprobación] 150 70 180 90 [ ]
creagroupbox "ultima "grupo 10 5 160 50
creacheckbox "ultima "grupo "boton1 [Primero] 15 15 45 10
creacheckbox "ultima "grupo "boton2 [Segundo] 15 25 45 10
creacheckbox "ultima "grupo "boton3 [Tercero] 15 35 45 10
creaboton "ultima "comprobar [COMPROBAR] 15 60 60 15 [comprueba]
creaboton "ultima "cerrar [CERRAR] 120 60 40 15 [borraventana "ultima]
creaestatico "ultima "estat1 [Desactivado] 100 15 60 10
creaestatico "ultima "estat2 [Desactivado] 100 25 60 10
creaestatico "ultima "estat3 [Desactivado] 100 35 60 10
fin
```

```
para comprueba
haz "x estadocheckbox "boton1
sisino :x= "verdadero [actualizaestatico "estat1 [Activado]] [actualizaestatico "es-
tat1 [Desactivado]]
haz "x estadocheckbox "boton2
sisino :x= "verdadero [actualizaestatico "estat2 [Activado]] [actualizaestatico "es-
tat2 [Desactivado]]
haz "x estadocheckbox "boton3
sisino :x= "verdadero [actualizaestatico "estat3 [Activado]] [actualizaestatico "es-
tat3 [Desactivado]]
fin
```

Primitivas <listbox>

Un objeto listbox es una caja de texto, que contiene una lista de opciones entre las que el usuario puede elegir una.

75. Haz un procedimiento llamado <**venmenu**> que dibuje en una ventana un menú de opciones mediante listbox. Dicho menú tendrá 5 opciones: CIRCULO, CUADRADO, TRIÁNGULO, PENTÁGONO y HEXÁGONO. Cuando pulsemos un botón se dibujará la figura seleccionada en el menú. Con otro botón podremos borrar la pantalla gráfica. Tendrá, además, otro botón que permita cerrar la ventana.

```
para venmenu
creaventana " "ultima [Trazado de polígonos] 300 250 180 90 []
crealistbox "ultima "menu "10 10 100 30
añadecadenalistbox "menu [CIRCULO ROJO]
añadecadenalistbox "menu [CUADRADO AZUL]
añadecadenalistbox "menu [TRIANGULO VERDE]
añadecadenalistbox "menu [PENTAGONO AMARILLO]
añadecadenalistbox "menu [HEXAGONO VIOLETA]
creaboton "ultima "calcular [DIBUJAR] 120 20 40 15 [dibujar]
creaboton "ultima "cerrar [ACABAR] 120 50 40 15 [borraventana "ultima]
creaboton "ultima "borrar [BORRAR] 60 50 40 15 [bp]
fin

para dibujar
haz "x leeseleccionlistbox "menu
si :x=[CIRCULO ROJO] [arorojo]
si :x=[CUADRADO AZUL] [cuadroblue]
si :x=[TRIANGULO VERDE] [triverde]
si :x=[PENTAGONO AMARILLO] [pentaillo]
si :x=[HEXAGONO VIOLETA] [hexaviol]
fin

para arorojo
arcodeelipse 360 120 120 0 sl poccr 4 gd 10 av 10 rellena bl ot
fin
```

```
para cuadroblue
repite 4 [av 200 gd 90] sl poccr 1 gd 10 av 10 rellena bl ot
fin

para triverde
repite 3 [av 220 gd 120] sl poccr 10 gd 10 av 10 rellena bl ot
fin

para pentaillo
repite 5 [av 150 gd 72] sl poccr 6 gd 10 av 10 rellena bl ot
fin

para hexaviol
repite 6 [av 120 gd 60] sl poccr 13 gd 10 av 10 rellena bl ot
fin
```

Primitivas <combobox>

Un objeto **combobox** es un objeto compuesto por una **lista**, en la que el usuario puede elegir una de las opciones que ofrece, y una **ventana de texto** en la que el usuario puede introducir una expresión que no está en la lista.

Para crear un combobox hay que utilizar la sintaxis siguiente:

creacombobox "ventana "nombre "Titulo posx posy largo alto

Una vez creado, podemos añadir el texto de las opciones que se ofrecen en la lista:

añadelineacombobox "nombre [expresion 1]

añadelineacombobox "nombre [expresion 2]

...

A continuación, designamos cuál de las opciones aparecerá seleccionada por defecto al crearse el combobox

pontextocombobox "nombre [expresion x]

Para averiguar cuál es la opción elegida por el usuario, utilizaremos la expresión:

leetextocombobox "nombre

76. Vamos a escribir el código de un script para una agencia de viajes, en el que el cliente puede elegir el lugar elegido para sus vacaciones. La agencia de viajes ofrece una lista de destinos comunes, los más solicitados, pero también permite al cliente solicitar un destino que no está en esa lista. Un objeto combobox es la opción ideal:

```
para viaje
creaventana " "destino [Elija el destino de su viaje] 150 80 120 110 []
creacombobox "destino "listado 20 5 80 50
añadelineacombobox "listado [Turquia]
```

```
añadelineacombobox "listado [Viena clásica]
añadelineacombobox "listado [Praga romántica]
añadelineacombobox "listado [Italia]
añadelineacombobox "listado [Fiordos noruegos]
añadelineacombobox "listado [Viñedos de Francia]
añadelineacombobox "listado [Costas de España]
añadelineacombobox "listado [Islas Canarias]
pontextocombobox "listado [Fiordos noruegos]
creaboton "destino "botacepto "Elegir 20 70 30 20 [confirma]
creaboton "destino "botcierre "Cerrar 70 70 30 20 [borraventana "destino]
creaestatico "destino "texto [Fiordos noruegos] 30 55 80 9
fin

para confirma
haz "lugar leetextocombobox "listado
actualizaestatico "texto :lugar
fin
```

Haz la prueba, eligiendo lugares de la lista ofrecida por la agencia y, también, eligiendo un lugar de vacaciones que no está en la lista.

Afterwords

Bien, espero que hayas disfrutado de tu aprendizaje como programador.

Ahora no me queda más que animarte a que apliques lo que has aprendido a crear los programas que puedas imaginar. Buena suerte.

Luis González

Profesor de Tecnologías de la Información

Madrid 050514