1. BLOQUE 1. INTRODUCCIÓN.

1.1. Órdenes básicas de dibujo.

Todas las órdenes que le demos a nuestra "tortuga" deben tener la misma estructura:

Comando ¡qué debe hacer!
Argumento ¡cuántas veces debe hacerlo!

Ejemplo AV 100. En esta orden el comando es AV (avanza) mientras que el argumento es 100. Por tanto la tortuga avanzará 100 unidades.

Veremos a continuación algunas de las órdenes básicas que podemos darle a la tortuga a la hora de dibujar:

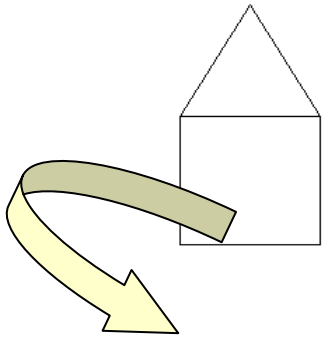
ORDEN	SIGNIFICADO
MT	Muestra la tortuga
OT	Oculta la tortuga
SL	Sube el lápiz
BL	Baja el lápiz
AV	Avanza
RE	Retrocede
GD	Gira a la derecha. Ej. GD 90
GI	Gira a la izquierda Ej. GI 120
BP	Borra la pantalla
PONPOS	Lleva al tortuga a una coordenadas Ej. ponpos [90 90]
PONRUMBO	Orienta la tortuga según un rumbo indicado en grados Ej. PONRUMBO 270
POS	Muestra las coordenadas X,Y de la tortuga
CIRCULO	Traza un círculo. Ej. CIRCULO 90
GOMA	La tortuga borra al avanzar
CENTRO	Centra la tortuga en la pantalla
PONCOLORLAPIZ	Fija un determinado color para el lápiz

- **Actividad 1**: Intenta dibujar las siguientes figuras. Recuerda levantar el lápiz entre dibujo y dibujo.

  - Un triángulo equilátero.
  - Un cuadrado.

  ¡Borra la pantalla!

  - Un pentágono.
  - Un círculo.
  - Un triángulo rectángulo



Ejemplo: Dibujando una casa

1º Dibujamos un cuadrado:	2º Le ponemos un tejado:
AV 100 GD 90 AV 100 GD 90 AV 100 GD 90 AV 100 GD 90	SL AV 100 BL PONRUMBO 30 AV 100 PONRUMBO 150 AV 100

¿No observas que repetimos varias veces las mismas instrucciones?
¡Debe existir una forma más fácil!

Trabajando en color

Tenemos dos formas de cambiar los colores del lápiz, el relleno de las figuras y el fondo:

1. Desde el menú "Configurar" "Color de..."
2. Empleando algunos comandos como:
   a. PONCL [x y z] (PON Color Lápiz)
   b. PONCP [x y z] (PON Color Papel)
   c. PON CR [x y z] (PON Color Relleno)

   x y z reprendan números que indican el código equivalente a cada color.

1.2. Una nueva orden: REPITE

Observa que con esta orden dibujar el cuadrado sería más sencillo:

```
REPITE 4 [
        AV 100
        GD 90]
```

¿Más fácil?
Inténtalo tú ahora...

- **Actividad 2**: Intenta dibujar un círculo empleando la orden repite.

1.3. Procedimientos.

¿Cada vez que deseemos dibujar una casa debemos escribir todas esas órdenes?
¡Debe existir una forma más fácil!

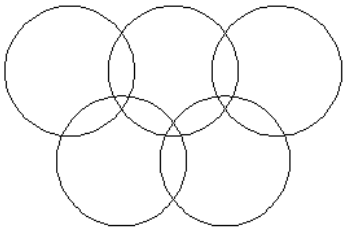
Hasta ahora hemos estado escribiendo órdenes en la pantalla, pero una vez ejecutadas se "pierden" y debemos teclearlas de nuevo. Existe una forma de almacenar un conjunto de órdenes y guardarlas con un nombre de forma que cada vez que las deseemos ejecutar sólo tengamos que "invocarlas". Estos conjuntos de ordenes los denominaremos "procedimientos" o "funciones".

Procedimiento "cimiento"	Procedimiento "tejado"
para cimiento REPITE 4 [AV 100 GD 90] fin	Para tejado SL AV 100 BL PONRUMBO 30 AV 100 PONRUMBO 150 AV 100 fin

Dentro de un procedimiento podemos hacer llamadas a otros procedimientos. Así podemos crear un procedimiento "casa" que llame a "cimiento" y "tejado". De esta forma sólo tendremos que teclear "casa" para obtener nuestra nueva vivienda...

```
para casa
        cimiento
        tejado
fin
```

- **Actividad 3:** Intenta crear un procedimiento pueblo que "construya" 5 casas equidistantes.
- **Actividad 4:** Crea un procedimiento que dibuje una serie de círculos estilo "anillos olímpicos".



- **Actividad 5:** Crea un procedimiento que dibuje un castillo.

Crear un procedimiento.

Para crear un procedimiento tenemos tres opciones:

1. Teclear directamente desde la línea de comandos la orden "para" seguido del nombre que deseemos darle al procedimiento. Automáticamente se abre una nueva ventana en la que vamos introduciendo línea a línea las instrucciones del procedimiento. Para terminar la introducción tecleamos "fin"
2. Teclear el comando ED " seguido del nombre del procedimiento. Se abre la ventana del editor de procedimientos y podemos ir escribiendo todos los comandos.

Modificar un procedimiento.

Para modificar un procedimiento tenemos dos opciones:

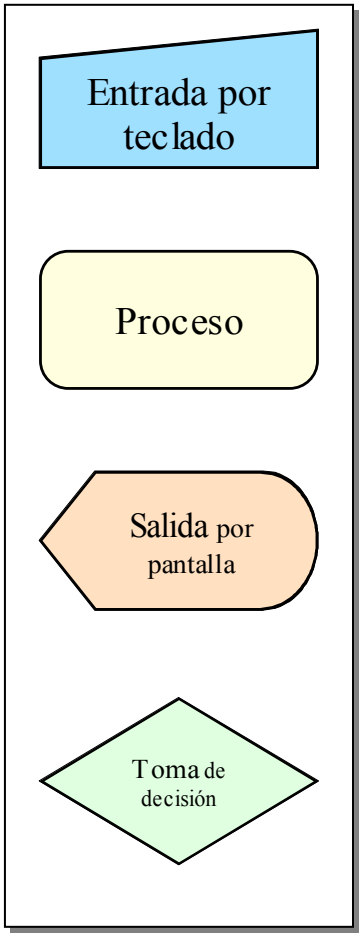
1. Teclear el comando ED " seguido del nombre del procedimiento. Se abre la ventana del editor de procedimientos y modificar y guardar los cambios.
2. Ir al menú "Fichero", "Editar" y luego seleccionar el nombre del procedimiento. Modificar y guardar los cambios.

2. BLOQUE 2. INTRODUCCIÓN A LA PROGRAMACIÓN.

Los pasos a seguir a la hora de elaborar un programa deberían ser los siguientes:

- Hacer un estudio del problema: datos necesarios, posibles soluciones, errores...
- Realizar un análisis del proceso que seguiremos para solucionar el problema.
- Elaborar un pseudo-código, es decir, un programa en el que las órdenes estarían expresadas en nuestro propio idioma y no en un lenguaje de programación.
- Crear el código en un determinado lenguaje.
- Probar el programa y depurar los errores.

2.1. Análisis del proceso.

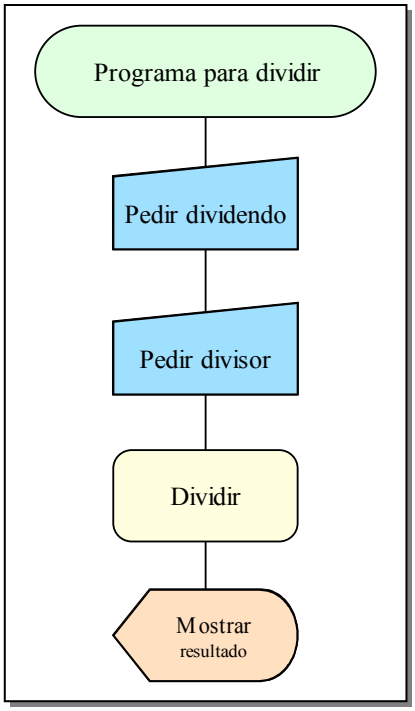


Antes de comenzar a escribir el código de nuestro programa (sea en el lenguaje que sea) lo primero que debemos realizar es un análisis del proceso que queremos realizar. Habitualmente este análisis se expresa en forma de organigrama empleando los símbolos que podemos ver en la figura de la derecha.

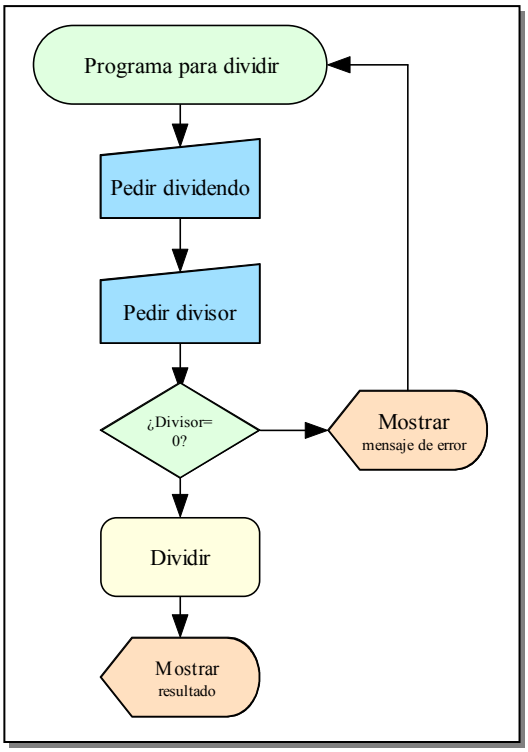
Supongamos que deseamos crear un programa que realice una división de dos números. Lógicamente los pasos que deben seguirse serían los siguientes:

- Pedir el dividendo.
- Pedir el divisor.
- Efectuar la operación.
- Mostrar el resultado

Estos pasos mostrador en forma de organigrama podrían tener el siguiente aspecto:



Este diagrama de flujo tiene un pequeño inconveniente y es que sería posible que el usuario intentará dividir entre 0, con lo que se generaría un error. Podríamos evitarlo si planteamos el programa de la siguiente forma:



- **Actividad 6:** Dibuja el organigrama de un programa que resuelva ecuaciones de segundo grado con soluciones dentro de los números reales.

2.2. Pseudo-código.

El pseudo-código sirve para hacer un programa, no válido en ningún lenguaje particular pero que servirá para una posterior codificación en cualquiera de ellos. Conociendo el significado de cada uno de los símbolos empleados en los organigramas, crear el pseudo-código es algo sencillo. Partiendo del anterior organigrama nuestro primer programa podría ser algo así:

```
Pedir dividendo
Pedir divisor

    Si divisor = 0 entonces
        Mostrar ERROR
        Ir a Pedir divisor

    En otro caso

        Calcular cociente
        Calcular resto
        Mostrar cociente
        Mostrar resto
```

- **Actividad 7:** Escribe el pseudo-código de un programa que resuelva ecuaciones de segundo grado con soluciones dentro de los números reales.

2.3. Manipulación de la Información.

a) Tipos de datos: numéricos, caracteres y booleanos.

Cuando creamos un programa éste deberá trabajar con datos. La forma de manipular y tratar estos datos puede depender del tipo de lenguaje de programación que estemos empleando, pero en general todos suelen dividir los datos en tres grandes grupos: numéricos, caracteres y boolenaos.

- *Numéricos:* Como su nombre indica hacen referencia a datos que representan un determinado valor numérico. Los tipos numéricos suelen dividirse en grupos según distintos criterios. Por una parte tenemos los tipos enteros y los tipos decimales. Los primeros se usan para trabajar con números que no tienen parte fraccionaria, mientras que los segundos son útiles en caso de que esa parte sí exista. Al tratar los números enteros hay que hacer otra división: con signo y sin signo. Los primeros siempre son positivos, mientras que los segundos son positivos y negativos.

- *Caracteres:* En los programas es habitual tratar con datos como el nombre de una persona, su dirección o su empresa. Éstos no son números sino sucesiones de caracteres.

  En realidad los caracteres son tratados por el ordenador como combinaciones de bits, es decir como combinaciones de unos y ceros. La diferencia con un dato numérico está en que cuando el ordenador "sabe" que esa combinación de bits representa un caracter no la interpreta como un número sino que lo "consulta" una tabla para ver que caracter representa esa combinación de unos y ceros.

  La tabla de caracteres más conocida es la llamada ASCII. En ella las combinaciones de bits equivalentes a los números 32 a 126 tienen asociado un caracter visible.

- *Booleanos:* Aparte de números y caracteres, prácticamente todos los lenguajes cuentan con otros tipos de datos capaces de representar de distintas maneras las combinaciones de bits que se almacenan en las celdillas de memoria. Uno de estos tipos es el booleano, que sólo puede contener dos valores posibles: "verdadero" y "falso".

  Este tipo de dato se utiliza para representar datos que suelen tener dos estados. ¿Tienen permiso de conducir? ¿Está el interruptor abierto?.

- **Actividad 8:** Indica a qué tipo de datos de los vistos anteriormente pertenecerían los siguientes.

    - DNI
    - NIF
    - Estado de un interruptor
    - Velocidad del viento
    - Provincia de residencia
    - Valor de una resistencia
    - Nota de un examen
    - Nota de la evaluación
    - Repetir curso

b) Constantes y variables.

Cada celdilla de memoria de un ordenador tiene asignada una dirección, un número que sirve para hacer referencia al dato que contiene dicha celdilla. Aunque la mayoría de los lenguajes de programación disponen de los elementos necesarios para poder trabajar con direcciones de memoria, lo más usual es utilizar una representación simbólica, un nombre o identificador, ya que las personas trabajamos mucho mejor con palabras que con números.

En cualquier lenguaje de programación actual podemos asociar un identificador, a una celdilla o conjunto de celdillas de memoria, de tal forma que a partir de ese momento podemos usar dicho identificador en lugar de las direcciones correspondientes.

Realmente el enlace entre los identificadores simbólicos y las direcciones de memoria lo efectuaría el compilador o intérprete que utilizásemos, de tal manera que nosotros, como programadores, podríamos olvidarnos de las direcciones físicas de las celdillas en que están contenidos nuestros datos.

Al crear un identificador, para así poder acceder a una o más celdillas de memoria y trabajar con ellas, la mayoría de lenguajes nos permitirán declarar ese identificador como una constante o como una variable.

Una constante toma un valor inicial que ya no podrá modificarse durante la ejecución del programa. Un valor como el del número ∏ podrá alojarse en la memoria utilizando una constante ya que no cambiará a lo largo del programa.

En contraposición, **una variable** es un identificador mediante el cual podrán efectuarse operaciones de lectura y escritura de las celdillas de memoria que representa. Si decidiésemos usar un identificador para poder alojar en una celdilla de memoria la temperatura, que vamos tomando mediante un equipo de medida, deberíamos usar una variable, ya que el valor de esta irá cambiando de cuando en cuando.

Algunos lenguajes de programación, como por ejemplo Pascal, exigen que declaremos todas las variables que vamos a emplear al principio del programa, mientras que otros permiten declarar variables en cualquier punto del programa.

- *Variables locales y globales*

Las **locales** tienen valor sólo dentro de un procedimiento o función, pero no fuera de él. Por tanto en distintos procedimientos podremos tener variables con el mismo nombre sin que interfieran entre si, es decir, tomando distintos valores.

Las variables **globales** tienen valor en todo el programa y no únicamente dentro de un procedimiento. Por tanto una variable global estará definida en todos los procedimientos de que conste el programa. Debe evitarse emplear el mismo nombre para una variable global que para una local, o éstas interferirán entre si.

Veamos algunos ejemplos de cómo se declaran variables en distintos lenguajes de programación:

Lenguaje	Declaración	Significado
Pascal	Var voltaje: Integer	La expresión "Var" le indica a Pascal que estamos declarando una variable. El nombre de la variable es "voltaje". La expresión "Integer" le indica Pascal que "voltaje" representa un dato numérico de tipo entero.
Visual Basic	Dim interruptor as boolean	La expresión "Dim"..."as" le indica a Visual Basic que estamos declarando una variable. El nombre de la variable es "interruptor". La expresión "boolean" le indica Visual Basic que "interruptor" representa un dato de tipo booleano.
MSWLogo	Haz "temperatura 23	La expresión "haz" le indica a MSWLogo que estamos declarando una variable global. El nombre de la variable es "temperatura" y su valor es 23. A diferencia de Pascal o Visual Basic en MSWLogo no podemos definir una variable global sin darle un valor.

- *Utilización de variables en MSWLogo.*

Variables globales:

Ya que éste es el lenguaje con el que vamos a trabajar a lo largo del curso, vamos a estudiar más en detalle como emplear las variables y luego haremos algunas actividades.

MSWLogo utiliza la forma:

Haz "*mivariable unvalor*

para asignarle un valor a una variable, donde **Haz "** es la orden para declarar una variable. m*ivariable* representa el nombre de la variable y *unvalor* el valor que le estamos asignado.

El valor de la variable se puede mostrar en cualquier otro punto del programa mediante la orden

Rotula :*mivariable*

¡Observa que los dos puntos van delante de la variable!

Variables Locales:

Las variables son locales en el procedimiento en que se encuentran. Las variables en Logo tienen un alcance dinámico; una variable local en un procedimiento no está disponible en los subprocedimientos invocados por ese procedimiento.

Se declaran en la primera línea de un procedimiento de la siguiente manera:

Para nombreprocedimiento **:**nombrevariable1 **:**nombrevariable2

Como ves podemos declarar más de una variable a la vez, el nombre de cada variable debe ir precedido del signo **:**

El contenido de estas variables sólo tiene existencia mientras que se ejecuta el procedimiento que las genera.

También se pueden generar variables locales mediante la orden

Local nombrevariable

Las variables creadas con LOCAL no tienen valor inicial; se les debe asignar un valor (p.e.con HAZ) antes de que el procedimiento intente leer su valor.

Practiquemos con el siguiente ejemplo:

```
Haz "numero1 100
Haz "numero2 200
Rotula :numero1
Av 200
Rotula :numero2
AV 100
BP
AV :numero1
GD 90
AV :numero2
```

Como puedes observar en MSWLogo no es necesario declarar que tipo de variable vamos a emplear (numérica, caracter o booleana). Podemos acceder a una variable simplemente con poner dos puntos seguidos del nombre de la variable.

Otro ejemplo: variables locales y globales

Crea el siguiente procedimiento

```
para variables :vl1

    local "vl2

    haz "vl2 10

    escribe [valor de vl1] escribe :vl1
    escribe [valor de vl2] escribe :vl2

    haz "vg 100

    escribe [valor de vg] escribe :vg
fin
```

En este procedimiento creamos tres variables. Dos de ellas locales (vl1 y vl2) y una global (vg). Durante el procedimiento se muestra el valor de las variables.
Una vez has ejecutado el procedimiento prueba a escribir directamente en la línea de comandos:

escribe :vl1
escribe :vg

En el primer caso obtendrás como respuesta:

vl1 No tiene valor

ya que vl1 es una variable local y no existe fuera del procedimiento en que fue definida.

Cometamos algunos errores:

1. Olvidando los dos puntos:

   Haz "numero3 300
   AV numero3

   Si escribes numero3 sin los dos puntos delante MSWLogo no lo toma como una variable y obtendrás el siguiente mensaje de error:

   No se cómo numero3

   De igual forma si tecleas:

   Rotula numero3

   MSWLogo no escribirá 300 sino que directamente escribirá "numero3" , es decir, el nombre de la variable en vez de su valor.

2. Variables de tipo caracter:

   Haz "saludo HOLA

   No podemos emplear está expresión directamente para declarar una variable de tipo caracter. Si lo intentas obtendrás el siguiente mensaje de error:

   No se cómo HOLA

   Si quieres introducir texto en una variable este debe ir entre corchetes de la siguiente forma

   Haz "saludo [HOLA]

   ¡Compruébalo!

- **Actividad 9:** Crea un procedimiento llamado rectángulo para dibujar un rectángulo. Define dos variables llamadas lado1, lado2,. Dales inicialmente el valor que quieras y empléalas para que la tortuga avance creando cada uno de los lados.

- *Cambiando el valor de una variable.*

Tenemos dos maneras para cambiar el valor de una variable ya definida.

1º Igual que hacemos para definirla. **Compruébalo**:

```
Haz "n1 100
Rotula :n1

Haz "n1 200
Rotula :n1

Haz "n1 :n1+:n2
Rotula :n1
```

Nota: en MSWLogo no podemos emplear expresiones del estilo :n1=:n2+:n3 para cambiar el valor de una variable, ya que son tomadas como una **comparación** y no como una **asignación.**

2º Modificándola en un procedimiento. **Compruébalo:**

Vamos a crear un procedimiento que haga avanzar a la tortuga un número que nosotros queramos y además muestre lo que avanza.

```
Para avanzar :numero
AV :numero
Rotula :numero
Fin
```

Para ejecutar el procedimiento escribe

```
avanzar 100
avanzar 20
```

Como puedes ver ni siquiera hace falta definir una variable con la orden **haz** en un procedimiento. Basta con poner el nombre del procedimiento seguido de la variable o variables (precedidos de dos puntos) que vamos a utilizar en él. De esta forma sí podemos crear variables "vacías" (sin valor inicial).

- **Actividad 10:** Modifica el anterior procedimiento "rectángulo" de forma que podamos especificar el tamaño de los lados.

- *Un pequeño problema..*

  Intenta mostrar el contenido de dos variables simultáneamente. Podemos pensar en la siguiente forma:

  ```
  Haz "a 2
  Haz "b 3
  Muestra :a :b
  ```

  ¿Qué sucede?.
  Bien, podemos intentarlo de la siguiente manera:

  ```
  Haz "a 2
  Haz "b 3
  Muestra [:a :b]
  ```

  ¿Qué sucede?. Tampoco vale...

  La solución es lo siguiente:

  ```
  Haz "a 2
  Haz "b 3
  Muestra (lista :a :b)
  ```

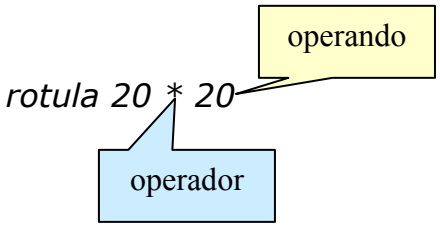
  ¡Compruébalo!

  La razón es la siguiente. [:a :b] es una lista con dos palabras ":a" y ":b". En cambio (lista :a :b) CONSTRUYE una lista con los valores contenidos en :a y :b.

Operandos y operadores

Para conseguir que un programa haga algo útil, aparte de almacenar y recuperar valores de la memoria también será necesario manipular esos valores con el fin de obtener resultados. Supón que tienes que calcular el total de una factura. Deberías tomar los importes parciales almacenados en varias celdillas de memoria y luego sumarlos para obtener un total.

Con el fin de manipular los datos o tomarlos como base para actuar de alguna manera se crean expresiones que pueden ser de distintos tipos: aritméticas, relacionales y lógicas principalmente. En dichas expresiones intervienen dos elementos fundamentalmente: los operandos y los operadores. El ejemplo más simple lo constituiría una multiplicación en logo:



Si ejecutas esta instrucción obtendrás el resultado de la operación. Compruébalo...

Como hemos dicho antes los operadores se agrupan básicamente en tres categorías: aritméticos, relacionales y lógicos.

Veamos algunos de los más habituales en cada una de estas categorías:

Aritméticos		Relacionales		Lógicos	
+	Suma	<	Menor	And	Si
-	Resta	>	Mayor	Or	O
*	Multiplicación	<>	Distinto	Not	Negación
/	División	=	Igualdad		

- **Actividad 11:** Vamos a practicar algunos operadores de MSWLogo. Crea una variable llamada a1 y asígnale el valor 100. Crea una variable llamada a2 y asígnale el valor 200.Intenta hacer las siguientes operaciones:

  - Muestra el resultado de a1*a2
  - Muestra el resultado de a2/a1
  - Haz que almacene en una variable llamada "total" la suma de a1+a2.

2.4. Control de flujo de un programa.

Actualmente todos los ordenadores funcionan en base a una arquitectura conocida como Neumann, llamada así en honor a su creador, el matemático Louis von Neumann.

Lo revolucionario de la idea de Neumman, propuesta en los años 40 cuando todavía no existían microprocesadores, está en la forma de instruir al ordenador acerca de lo que debe hacer. Hasta ese momento los ordenadores eran máquinas con un programa fijo encargado de realizar una tarea concreta, de tal manera que la única entrada por parte de los usuarios eran los datos que, tras ser procesador, generaban unos resultados. Es como funciona hoy en día una calculadora. La idea de von Neumman era radicalmente distinta: el programa ejecutado por el ordenador no sería un concepto fijo, sino que podría introducirse en su memoria igual que se introducían hasta ese momento los datos.

De esta forma los ordenadores comenzaron a utilizar la arquitectura von Neumman y se convirtieron en máquinas multipropósito.

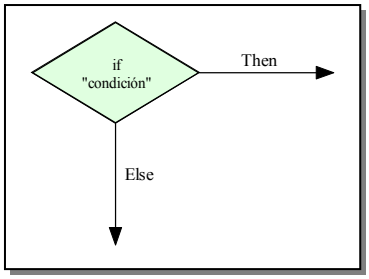
- Ejecución del código.

Cuando se pone en marcha la ejecución de un programa, el proceso que sigue el ordenador es, básicamente, el siguiente: se toma una instrucción, se procesa y ejecuta, se toma la siguiente instrucción y así sucesivamente.

Si un programa se ejecutase siempre secuencialmente, de principio a fin, lo cierto es que sería complicado lograr algo más que resultados simples.

Para lograr que un programa pueda ser útil, este debe ser flexible, ejecutarse de distinta forma en función de algunas variables. Son necesarias dos estructuras básicas en programación: los condicionales y los bucles. A continuación aprenderemos algo más sobre ellos.

Estructuras condicionales



Todos los lenguajes de programación cuentan con una estructura condicional simple, con una salida afirmativa y otra negativa como la representada en la figura.

Planteamos una condición, si ésta se cumple (then) el programa ejecuta una serie de instrucciones, en caso contrario (else) el programa ejecuta otras distintas.

La sintaxis común de esta estructura condicional, expresada en código, suele ser la siguiente:

Lenguaje	Declaración	Significado

Pascal	**If** "condición" **then**
	"Acción1"
	else
	"Acción2"
		Si se cumple la condición se ejecutan unas sentencias (Acción1) en caso contrario (else) se ejecutarán otras diferentes (Acción2).

Actionscript	**If (**"condicion**") {** "Acción1**"}** **else {** "Acción2"**}**	Vemos que la sintaxis es igual que en Pascal pero en este caso no es necesaria la instrucción **then .**
MSWLogo	**Sisino** "condición" **[**"Acción1"**]** **[**"Acción2"**]**	La sintaxis es nuevamente similar a los casos anteriores. En logo las sentencias que deseemos utilizar irán entre corchetes.
MSWLogo	**Si** "condición" **[**"Acción1"**]**	Ejecuta un grupo de acciones en caso de que la sentencia sea cierta y no ejecuta nada si la sentencia es falsa.

Ejemplo de un programa en pascal:

```
Program Edades;
Var
  edad : integer ;
begin
   WriteLn('Escribe tu edad : ') ;
   ReadLn(edad) ;
   if edad >= 18 then
          WriteLn('!Eres Mayor de edad !')
   else
          WriteLn('!Eres Menor de edad !');
   WriteLn('Esta instrucción siempre se
ejecuta');
end.
```

Este programa determina si eres mayor de edad o no. La instrucción

```
ReadLn(edad) ;
```

sirve para introducir al edad por el teclado. A continuación el programa comprueba si se cumple al condición de que la edad sea mayor que 18 o no. En función de esta edad escribe:

```
WriteLn('!Eres Mayor de edad !')
```

O bien:

```
WriteLn('!Eres Menor de edad !');
```

- Las condicionales en MSWLogo.

  Veamos el mismo programa anterior realizado en MSWLogo.

```
para edades :edad

sisino :edad>18 [escribe [Eres mayor de edad] ] [escribe [Eres menor de edad]]
escribe "Adios

fin
```

- **Actividad 12:** Escribe un procedimiento llamado "cruzar" que dé a un peatón la orden de cruzar o no en función del estado de una variable llamada "luz".

- Estructuras de repetición.

Gracias a las estructuras condicionales ya podemos controlar qué sentencias de nuestro programa son ejecutadas en cada caso, de tal manera que dicha ejecución no es completamente secuencial. Con una condición (If, Si) podemos saltarnos la ejecución de una o más sentencias, pero en realidad no nos sirve si precisamos la realización reiterada de una o más sentencias.

Supón que deseamos realizar un programa para que la tortuga avance y gire hasta que se cumpla una determinada condición. ¿Podemos hacerlo simplemente con una condicional?. La respuesta es no. Necesitamos una nueva estructura conocida como bucle.

Un bucle estará formado por una o más sentencias que deben repetirse, para lo cual se delimitan con las órdenes adecuadas para hacer posible la ejecución reiterada. Generalmente un bucle siempre tiene asociada una expresión condicional cuyo resultado "verdadero" o "falso", condiciona que el bucle se siga ejecutando o no.

El tipo de bucle más común, existente en todos los lenguajes de programación, tiene una sintaxis similar a esta:

mientras condicional
sentencias

Veamos algunos ejemplos en distintos lenguajes:

Lenguaje	Declaración	Significado
Pascal	**while** <condición> **do** **begin** <instrucciones>; **end**;	Primero evaluamos la condición. Si ésta se cumple entramos en el bucle, en caso contrario no se ejecuta el bucle. Mientras la condición sea verdadera el bucle continuará indefinidamente a menos que "algo" en el interior del bucle modifique la condición haciendo que su valor pase a falso. Si la expresión nunca cambia de valor, entonces el bucle no termina nunca y se denomina *bucle infinito* lo cual no es deseable.

Pascal	**for** <contador>:=<expresión.1> **to** <expresión.2> **do**
	begin
	<instrucciones> ;
	end;
		Cuando se sabe de antemano el número de veces que deberá ejecutarse un ciclo determinado, ésta es la forma más conveniente.
		
		Al ejecutarse la sentencia **for** la primera vez, a contador se le asigna un valor inicial(expresion.1), y a continuación se ejecutan las instrucciones del interior del bucle, enseguida se verifica si el valor final (expresión.2) es mayor que el valor inicial (expresión.1); en caso de no ser así se incrementa contador en uno y se vuelven a ejecutar las instrucciones, hasta que el contador sea mayor que el valor final, en cuyo momento se termina el bucle.

MSWLogo	**Haz.mientras** [listainstrucciones] [condición]	Ejecuta una lista de instrucciones mientras que se cumpla una determinada condición. Observa que en MSWLogo la condición es el segundo argumento.

- Los bucles en MSWLogo.

Veamos a continuación algunos de los tipos de bucles que podemos emplear en MSWLogo.

DESDE

Este bucle es similar al **FOR** que se emplea en Pascal o en Basic. Su estructura es la siguiente:

desde [variable valorinicial valorfinal incremento]
[instrucciones]

Con este bucle se emplea una variable local a la que fijamos un valor inicial y otro final. Cada vez que se ejecuta el bucle el valor inicial se incrementa en la cantidad que nosotros fijemos como incremento, hasta que se alcanza el valor final. A la variable que empleamos para irse incrementando y recorriendo una serie de valores se le denomina **CONTADOR**.

Ejemplo:

***desde** [i 2 7 1.5] [escribe :i]*

Con lo que logo mostrará el siguiente resultado:

2
3.5
5
6.5

Este bucle lo emplearemos cuando conocemos los valores iniciales y finales que debe recorrer el contador.

HAZ.MIENTRAS

Este bucle es similar al **While...DO** de Pascal. Ejecuta una lista de instrucciones (listainstrucción) mientras se cumpla la condición. Observa que la condición es el segundo argumento.

Haz.mientras [listainstrucción] [condición]

Ejemplo:

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

Abandonar un bucle

Ejemplo: La tortuga avanza lentamente hasta que presionamos la tecla p.

```
para bucle

        gd 90
        haz "x 0
        ponteclado [haz "x car leecar]
        ponfoco [Pantalla de MSWLogo]
        rotula [Pulsa la tecla p para parar]
        repite 100[
                si :x="p [alto]
                espera 10
                av 1]
fin
```

Veamos las primitivas que usamos en este procedimiento:

Ponteclado: Esta orden captura los eventos del teclado.
Ponfoco: Esta primitiva hace que MSWLogo controle la ventana que tiene el foco. La ventana se especifica por su título.
Leecar: Devuelve el código ASCII correspondiente a la última tecla pulsada.
Car: Devuelve el caracter correspondiente a un código ASCII.

Veamos una variación que nos permite detener y continuar el bucle a voluntad:

```
para bucle
gd 90
haz "x 0
ponteclado [haz "x car leecar]
ponfoco [Pantalla de MSWLogo]
rotula [Pulsa la tecla p para parar]
repite 1000[
        si :x="p [pausa haz "x 0]
        espera 10
        av 1]
fin
```

Veamos un par de fallos que podrían ser habituales a la hora de escribir bucles:

```
para bucle
gd 90
haz "x 0
ponteclado [haz "x car leecar]
ponfoco [Pantalla de MSWLogo]
rotula [Pulsa la tecla p para parar]
        repite 1000[
                si :x="p [pausa]
                espera 10
                av 1]
        fin
```

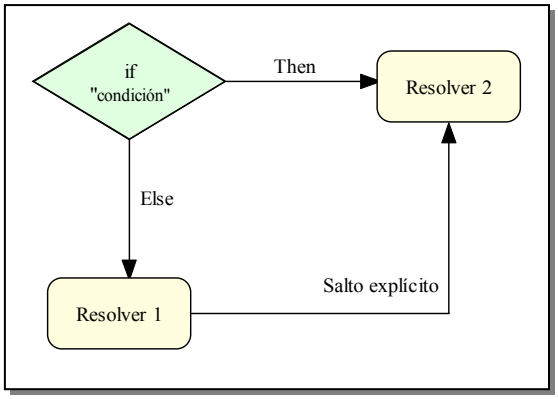
```
para bucle
gd 90
ponteclado [haz "x car leecar]
ponfoco [Pantalla de MSWLogo]
rotula [Pulsa la tecla p para parar]
        repite 1000[
                haz "x 0
                si :x="p [pausa]
                espera 10
                av 1]
        fin
```

¿Cuáles crees que son los fallos?

- **Actividad 13:** Escribe un procedimiento que en función de dos variables iniciales cuente desde una hasta otra con incrementos de dos en dos. La segunda variable ha de ser mayor que la primera. En caso contrario el programa dará un mensaje de error y se detendrá (para detener un procedimiento emplearemos el comando ALTO).

- Bifurcaciones.

Tanto los condicionales como los bucles, según hemos podido ver, lo que hacen es desviar la ejecución del programa desde un punto hasta otro. Es lo que se conoce como una bifurcación en el flujo de ejecución de un programa. Los saltos, no obstante, están controlados por el condicional o el propio bucle, no siendo nosotros, de forma explícita, los que indicamos a dónde hay que pasar el control del programa.



Los saltos explícitos, de un punto a otro del programa, pueden utilizarse en situaciones donde sea necesario cambiar de un proceso a otro totalmente distinto.

En MSWLogo esto podemos hacerlo de una forma muy simple. Basta con escribir el nombre del procedimiento al que queramos saltar.

- Comunicándonos con el usuario.

Ya hemos aprendido las principales estructuras para programar. Lo único que nos falta para "profesionalizar" nuestro programas es ver un par de instrucciones de MSWLogo que nos permitirán comunicarnos mejor con el usuario.

ESCRIBE

Ésta ya la conocemos pero de todas formas vamos a repasar su funcionamiento:

ESCRIBE objeto
ES objeto

(ESCRIBE objeto1 objeto2 ...)
(ES objeto1 objeto2 ...)

Esta primitiva escribe la entrada o entradas actuales. Todas las entradas se escriben en una sola línea, separadas por espacios y terminando en una nueva línea. Si la entrada es una lista, no escribe los corchetes.

Ejemplo:

```
escribe "Hola
Hola
escribe [Hola, cómo estás]
Hola, cómo estás
```

MENSAJE

MENSAJE título cuerpo

Detiene el proceso y despliega una ventana con un mensaje (cuerpo). En la barra de título aparece una etiqueta (título). El proceso anterior continúa cuando el usuario pulsa el botón ACEPTAR o CANCELAR.

título:(LISTA) Se usa como título de la ventana.
cuerpo:(LISTA) Texto del mensaje. La ventana se adapta al tamaño del mensaje.

Ejemplo:

mensaje [Bienvenida][Hola a todos os saludo desde LOGO]

LEELISTA

lista LEELISTA
lista LL

Lee una línea desde el lector de cadenas (inicialmente el terminal) y devuelve la línea como una lista. La línea se compone de sus elementos como si se hubiera tecleado entre corchetes LEELISTA no trata el punto y coma como un caracter de comentario.

lista:(LISTA).

Ejemplo:

rotula leelista
<Entrada por teclado (Hola, ¿cómo estás? <CR>) en la ventana de diálogo>
[Hola, ¿cómo estás?]

Podemos usar esta instrucción para almacenar valores introducidos por el usuario en una variable de la siguiente forma:

Haz "a **leelista**
Escribe :a

Este sistema tiene un inconveniente. La ventana de diálogo que se muestra al usuario no ofrece información sobre lo que esperamos que haga. Podemos mejorar el método de la siguiente forma:

PREGUNTABOX

datos PREGUNTABOX título cuerpo

Detiene el proceso y despliega una ventana con un mensaje (cuerpo). En la barra de título aparece título. El proceso anterior continua cuando el usuario pulsa uno de los botones SI o CANCELAR.

datos: Es una variable que contiene la respuesta del usuario. Devuelve una lista vacía si el usuario pulsó CANCELAR.

título: Se usa como etiqueta de la ventana.

cuerpo: Texto del mensaje. La ventana se adapta al tamaño del mensaje.

Ejemplo:

```
para sumar

    Mensaje [Sumar] [Este es un programa para sumar dos números]

    haz "n1 primero preguntabox [Sumar] [Introduce el primer número]

    haz "n2 primero preguntabox [Sumar] [Introduce el segundo número]

    haz "resultado :n1+:n2

    Mensaje [Resultado] :resultado

fin
```

La palabra "primero" antes de "preguntabox" es una función de MSWLogo que extrae el valor introducido por el usuario como un número. Debemos emplear esta instrucción ya que en caso contrario se interpretaría como una cadena de caracteres y no podríamos realizar operaciones aritméticas.

- **Actividad 14:** Modifica la actividad 13 para que las dos variables sean introducidas por el usuario mediante ventanas de Windows. Añade mensajes explicativos al programa.

- **Actividad 15:** Crea un programa que nos permita calcular la energía consumida por un electrodoméstico en función de su potencia y las horas de funcionamiento. Debe mostrar el resultado tanto en KWh como en julios. Emplea ventanas para entrar los datos y añade comentarios explicativos. (E=P·t)

- **Actividad 16:** Crea un juego para adivinar un número del 1 al 10. Una vez que se acierte el número nos mostrará los intentos. (Primitiva: AZAR)