DESARROLLO DE APLICACIONES WEB

Unidad 1

Introducción a la Programación

1r DAW

IES Enric Valor de Monòver

Departament d'informàtica

Professor: Joan Carbonell Picó

ÍNDEX

Unidad 1: Introducción a la Programación

1.- ALGORITMOS Y PROGRAMAS

1.1.-DEFINICIONES

Algoritmo: Secuencia finita de reglas o instrucciones que especifican un conjunto de operaciones, que al ser ejecutadas por un agente ejecutor (máquina real o abstracta), resuelve cualquier problema de un tipo determinado en un tiempo finito. Un algoritmo es independiente del lenguaje de programación. Es decir, una vez tengamos un algoritmo que resuelve un problema, podemos aplicarlo sobre cualquier lenguaje de programación.

Llenguatge de programació: És un llenguatge informàtic utilitzat per controlar el comportament d'una màquina, normalment un ordinador. Cada llenguatge té una sèrie de regles sintàctiques i semàntiques estrictes que cal seguir per escriure un programa informàtic, i que en descriuen l'estructura i el significat respectivament.

Programa informático: Conjunto de instrucciones que implementan un algoritmo. Una vez ejecutadas, las instrucciones realizarán una o varias tareas en un ordenador.

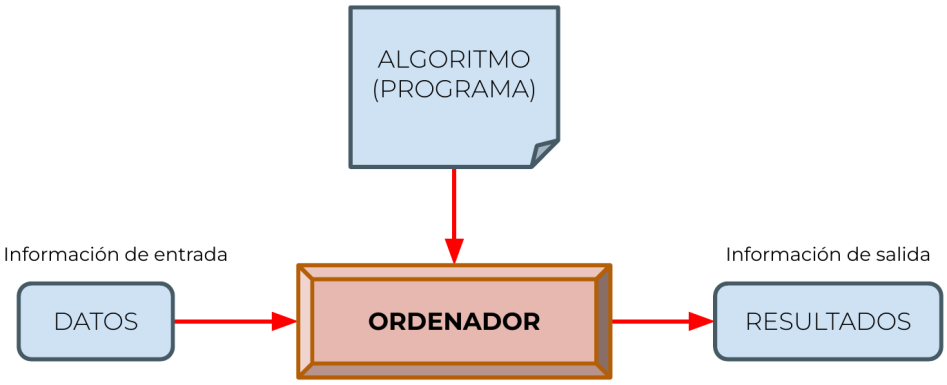
Programación: Es el proceso por el cual una persona desarrolla un programa valiéndose de una herramienta que le permita escribir el código (el cual puede estar en uno o varios lenguajes, como por ejemplo C++, Java y *Python entre otros) y de otra que sea capaz de “traducirlo” al que se conoce como lenguaje de máquina, el cual puede ser entendido por un microprocesador.

Software: Conjunto de programas, documentación y datos estrechamente relacionados para conformar una aplicación informática

1.2.-CARACTERÍSTICAS DE UNA ALGORITMO

- **Preciso**: Tiene que indicar la orden de realización de cada paso.
- **Definido**: Si se sigue un algoritmo dos vueltas, se tiene que obtener el mismo resultado cada vuelta.
- **Finito**: Tiene que finalizar en un número fenecido de pasas.

Además, un algoritmo recibe información de entrada, la procesa, y genera información de salida, como se indica en la siguiente figura:



No todos los problemas pueden resolverse utilizando un ordenador. Solo se podrán resolver aquellos problemas que tengan una solución mecánica. Es decir, aquellos que se resuelvan por medio de una secuencia de instrucciones determinada. Son los denominados **problemas computacionales o algorítmicos**. Como por ejemplo, los relacionados con el cálculo numérico, el tratamiento de palabras, o la representación gráfica.

Un ejemplo de problema computacional podría ser obtener el área de un cuadrado a partir de su lado. En cambio, un problema no computacional podría ser averiguar el color favorito de una persona a partir de su nombre.

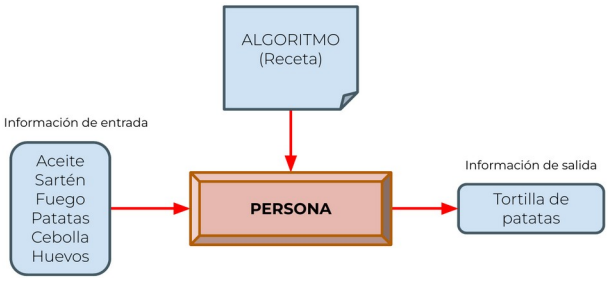
1.3.-EJEMPLO DE ALGORITMO.

A continuación vamos a ver un ejemplo de algoritmo para elaborar una tortilla de patatas. Tendremos unos datos de entrada, luego el algoritmo que procesa los datos de entrada, y finalmente unos datos de salida o resultado.

Datos de entrada: Huevo, aceite, patatas, cebolla, sartén, fuego.

Datos de salida: Tortilla de patatas.

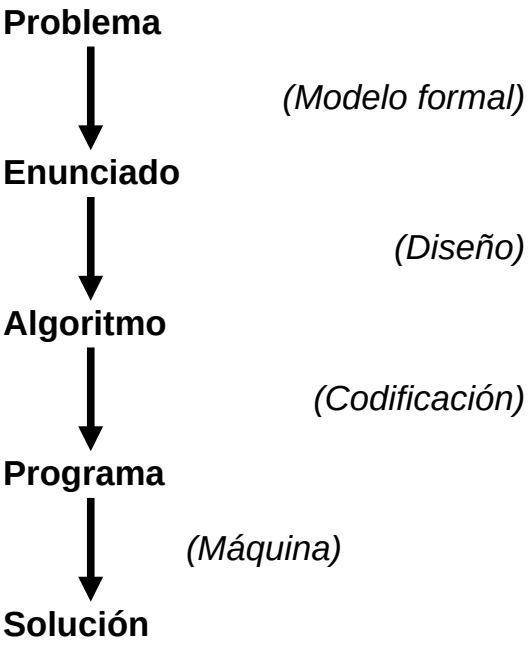
Algoritmo:



1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, freír las patatas y la cebolla.
4. Batir los huevos y añadir las patatas y la cebolla cuando estén al gusto.
5. Añadir los huevos a la sartén y darle la vuelta varias veces.

1.4.-CREACIÓN DE UN PROGRAMA

- Dado un problema intentaremos encontrar un modelo formal que nos permita representarlo como un enunciado.
- Mediante una técnica de diseño realizaremos un algoritmo que resuelva el problema
- Mediante un lenguaje de programación implementaremos el programa.
- Una vez ejecutado el programa por un agente ejecutor obtendremos un resultado que nos dará la solución del problema



1.5.-DATOS Y VARIABLES

Cuando vamos a implementar un algoritmo para resolver un problema, vamos a utilizar diferentes tipos de información como por ejemplo el tiempo, euros, cantidad de productos, etc.… Además, también tendremos que almacenar los resultados de los cálculos intermedios obtenidos. El tipo de información representan los datos y los resultados intermedios se almacenamos en variables.

- **Datos**: Información representada en un formato adecuado para su procesamiento.
- **Variables**: Símbolo empleado en programación para almacenar en memoria valores cambiantes.

Asociado a estos dos conceptos, tenemos que hablar de:

- **Tipo de datos**: Definición de un conjunto de valores válidos que pueden tomar unos datos y el conjunto de transformaciones que se pueden hacer. Además, los **tipos primitivos de datos** son el que ya están incorporados directamente dentro de un lenguaje de programación , y son usados como piezas básicas para construir de más complejos. Hay 4 que, de una manera y de otra todos los lenguajes soportan: enteros, reales, caracteres y booleanos.

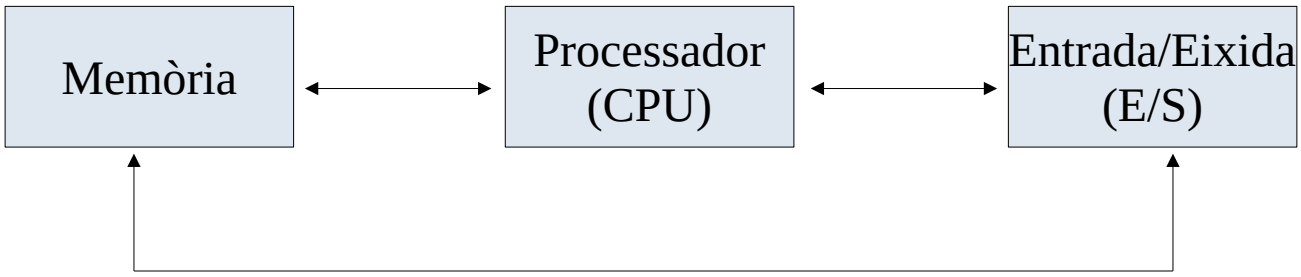
- **Operaciones**: Son las transformaciones que se pueden hacer sobre los tipos de datos. Normalmente, tan solo se pueden hacer operaciones entre datos del mismo tipo. Pero si que se puede dar el caso que una operación entre dos tipos de datos iguales doy como resultado un tipo de datos diferente. Además, algunas operaciones tampoco tendrían sentido. Las operaciones se pueden dividir en unarias y binarias, según el número de operandos que utilizan.

- **Constantes**: Al contrario que las variables almacena valores que nunca serán modificados por los algoritmos.

1.6.-COMPONENTES DE UN ORDENADOR

Como hemos dicho anteriormente, un algoritmo es un conjunto finito y ordenado de pasos, reglas o instrucciones para resolver un problema. El que tendremos que ver es de qué manera cada una de las órdenes de un programa estará vinculado a los componentes en que está formado un ordenador.

La estructura interna de un ordenador se divide en los siguientes componentes, todos ellos comunicados entre sí según la siguiente figura:



- **Procesador**: Parte de la máquina que ejecuta las órdenes de manipulación y transformación de datos.

- **Memoria**: Lugar donde están almacenadas todos los datos que tiene que tratar un programa. Es volátil y los datos desaparecen una vuelta finalizado el programa. Es por eso que otra posibilidad importante del ordenador, atendidas las limitaciones del sistema de memoria, es poder interactuar con el hardware de almacenamiento persistente de datos, como un disco duro.

- **Entrada/salida**: Permite el intercambio de datos con el exterior del ordenador, más allá del procesador y la memoria

A partir de esta descripción de las tareas que puede llevar a cabo un ordenador según los

elementos que lo componen, un ejemplo de programa para multiplicar dos números es mostrado a la siguiente tabla. Está expresado en lenguaje natural (aquel que empleamos los humanos para comunicarnos habitualmente). Comprobar como los datos tienen que estar siempre almacenadas a la memoria para poder operar.

ORDEN A DAR	ELEMENTO QUE LO EFECTÚA
1. Lee un número del teclado.	1. E/S (teclado)
2. Guarda el número a la memoria.	2. Memoria
3. Lee otro número del teclado.	3. E/S (teclado)
4. Guarda el número a la memoria.	4. Memoria
5. Recupera los números de la memoria y haz la multiplicación.	5. Procesador
6. Guarda el resultado a la memoria.	6. Memoria
7. Muestra el resultado a la pantalla.	7. E/S (pantalla)

2.- LENGUAJES DE PROGRAMACIÓN

Hemos visto que los programas son conjuntos de instrucciones que se proporcionan a un ordenador para completar una tarea. Estas instrucciones están escritas en un **lenguaje de programación** que escogemos, y de este modo creamos unos ficheros de texto llamados **código fuente**, escritos en el lenguaje escogido. Aquí tenéis dos ejemplos:

HOLA MUNDO EN C++

```
#include <iostream>
using namespace std;
int main() {
        cout << "Hola Mundo" << endl;
        return 0;
}
```

HOLA MUNDO EN JAVA

```
public class Hello {
        public static void main(String[] args) {
                System.out.println("Hola mundo");
        }
}
```

Ejemplo código ensamblador

```
L0:  MOV   R1, #a      ;
     MOV   R2, #b      ;

L1:  LD    R3, (R1)    ;
     CMP   R3, #0      ;
     BNE   L3          ;

L2:  MOV   R4, #1      ;
     JMP   L4          ;

L3:  MOV   R4, #0      ;

L4:  ST    (R2), R4    ;
     JMP   L1          ;
```

Ejemplo código binario

```
01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000
01101100 01100101 01100001 01110010
01101110 00100000 01100010 01101001
01101110 01100001 01110010 01111001
00101110 00100000 01001001 00100000
01101000 01101111 01110000 01100101
00100000 01111001 01101111 01110101
00100000 01100101 01101110 01101010
01101111 01111001 00100000 01101001
01110100 00100001
```

Por lo tanto, un lenguaje de programación es un lenguaje artificial diseñado expresamente para crear algoritmos que puedan ser llevados a cabo por el ordenador.

2.1.-TIPOS DE LENGUAJES

Cuando queremos escoger un lenguaje de programación específico, distinguimos entre lenguajes de **alto nivel** (cerca del lenguaje humano y, por lo tanto, más fáciles de entender por los programadores) y lenguajes de **bajo nivel** (cerca del lenguaje de máquina, y por tanto, más difíciles de entender por parte de los programadores pero más eficientes). Esta clasificación atiende a la proximidad a la máquina

2.2.-PROCESADORES DE LENGUAJES

Los ordenadores no pueden entender ninguno de los lenguajes de programación que utilizan los humanos para crear sus programas. Para que funcionan, sus instrucciones tienen que ser traducidas a un lenguaje que los ordenadores puedan entender. Este lenguaje se denomina **código máquina**, y está formado por bits (ceros y unos).

Si queremos traducir un determinado lenguaje de programación a código máquina, utilizamos una herramienta llamada compilador, aunque esta afirmación no es completamente cierta. Hay varios procesadores de lenguaje que se pueden utilizar, dependiendo del idioma en sí:

- **Compiladores**: Traducen el código escrito en un lenguaje concreto a código máquina, y generan un fichero ejecutable el procesador puede entender. El contenido de este fichero se denomina código objete. Por ejemplo, si compilamos un programa escrito en C++ en Windows, obtendremos un fichero .exe que podremos ejecutar.

- **Intérpretes**: Traducen desde el lenguaje de programación especificado al código máquina "sobre la marcha". En otras palabras, no generan ningún fichero ejecutable. Por lo tanto, cada vez que necesitamos ejecutar el programa, tenemos que tener disponible el fichero fuente. Este tipo de procesador de lenguaje es muy habitual en lenguajes como PHP o Python. De este modo, el tiempo de respuesta aumenta un poco, pero el programa se puede ejecutar en varias plataformas.

- **Máquinas virtuales**: Una solución intermedia entre compiladores e intérpretes es la que usan lenguajes como Java. Estos programas no se compilan en un código de máquina nativo (no hay ningún fichero .exe en Java, por ejemplo) y tampoco se interpretan. Java compila el código fuente y lo traduce a su propio código máquina intermedio. Después, ejecuta su

máquina virtual (JVM, *Java Virtual Machine*), que se encarga de interpretar y ejecutar este código cada vez que lo necesitamos. De este modo, no necesitamos tener el código fuente disponible antes de ejecutar el programa, y tampoco dependemos de una plataforma determinada (Windows, Linux, etc.). Solo necesitamos tener una JVM instalada a nuestro sistema para ejecutar nuestros programas Java.

3.- ENTORNOS DE DESARROLLO

Un **IDE** (*Integrated Development Environment* / Entorno Integrado de Desarrollo) es una herramienta que integra todo el que hace falta para generar programas de ordenador, de forma que el trabajo sea mucho más cómodo.

La utilización de estas herramientas agiliza increíblemente el trabajo del programador. Además, los IDE más modernos van más allá de integrar editor, compilador y lacero o intérprete, y aportan otras características que hacen todavía más eficiente la tarea de programar. Por ejemplo:

- Posibilidad de hacer resaltar con códigos de colores los diferentes tipos de instrucciones o aspectos relevantes de la sintaxis del lenguaje soportado, para facilitar la comprensión del código fuente.
- Acceso a documentación y ayuda contextual sobre las instrucciones y sintaxis de los lenguajes soportados.
- Detección, y en algunos casos incluso corrección, automática de errores de sintaxis en el código, de manera similar a un procesador de texto. Así, no hay que compilar para saber que el programa está mal.
- Apoyo simultáneo del desarrollo de lenguajes de programación diferentes.
- Un depurador, una herramienta muy útil que permite hacer una pausa a la ejecución del programa en cualquier momento o hacerla instrucción por instrucción, de forma que permite analizar como funciona el programa y detectar errores.
- En los más avanzados, sistemas de ayuda para la creación de interfaces gráficas.

En definitiva, usar un IDE para desarrollar programas es una opción muy recomendable. Aun así, hay que tener presente que son programas más complejos que un simple editor de texto y, como pasaría con cualquier otro programa, hay que dedicar cierto tiempo a familiarizarse con estos y con las opciones de que disponen.

Además, hay que tener en cuenta que uno en torno a desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. Qué quiere decir esto? Pues que si tenemos instalado un IDE y no tenemos instalado el compilador, no tenemos nada.

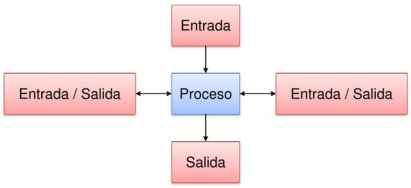
Para el lenguaje de programación Java existen múltiplos alternativas, siente los principales entornos de desarrollo NetBeans (que cuenta con el apoyo de la empresa Sun), Eclipse y JCreator. Los dos primeros son gratuitos, con apoyo de idiomas y multiplataforma (Windows, Linux, Guapos). Pero también hay otros como Geany y IntelliJ IDEA.

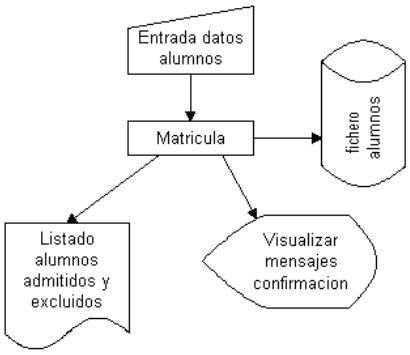
4.- REPRESENTACIÓN DE LOS ALGORITMOS

Los algoritmos se pueden representar de las siguientes formas:

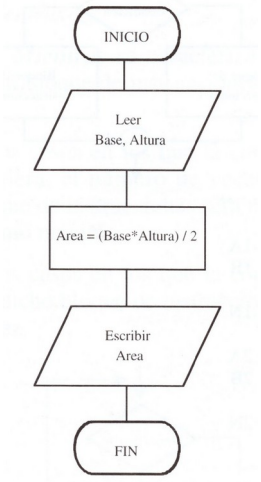
Gráfica | **Textual**

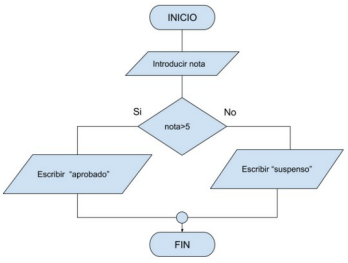
Organigrama
(flujo de sistema)





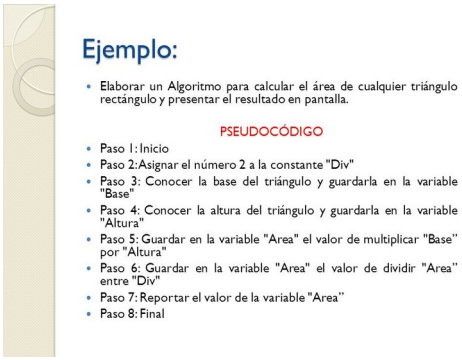
Ordinograma
(Representa de manera gráfica el orden de pasos y acciones de un algoritmo)





Pseudocódigo

```
Algoritmo Suma de dos números
Declaración
Variables
    A,B,Suma: Entero
Inicio
    Escribir “Suma de dos números...”
    Escribir “Introduzca Primer valor:  ”
    Leer  A
    Escribir “Introduzca Segundo valor: ”
    Leer  B
    Suma ← (A + B)
    Escribir “El Resultado la suma es: ”, Suma
Fin
```



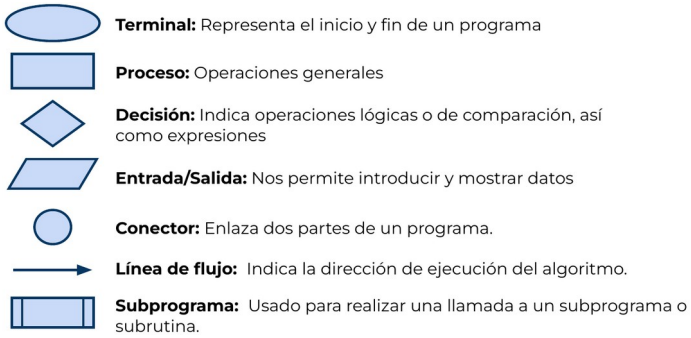
Podemos ver que para representar un algoritmo se suele utilizar algún método que permite independizarlo del lenguaje de programación utilizado. Esto nos permitirá codificar más tarde a

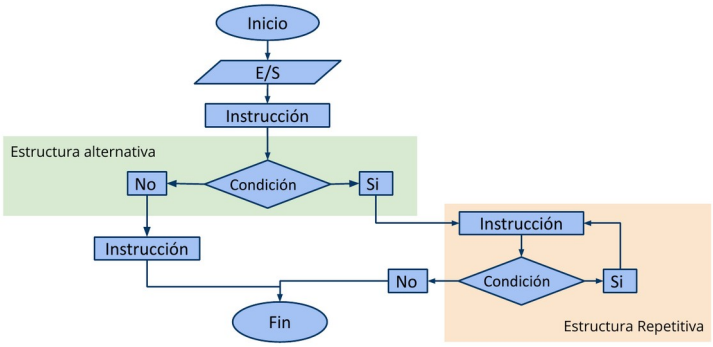
través de cualquier lenguaje de programación.

A continuación estudiaremos las diferentes técnicas más destacables que se utilizan para realizar algoritmos.

4.1.-ORDINOGRAMAS.

Los diagramas de flujos u ordinogramas utilizan símbolos gráficos estándar y escriben los pasos del algoritmo dentro de los símbolos. A través de las líneas de flujo se indican la secuencia en que se deben ejecutar.





4.2.-PSEUDOCÓDIGO.

Los lenguajes de descripción de algoritmos o pseudocódigos son un lenguaje para la descripción de algoritmos que procura mantener las propiedades de los diagramas de flujo: sencillez, claridad, normalización y flexibilidad.

Su notación se basa en el lenguaje natural. La estructura general del pseudocódigo está dividida en dos partes:

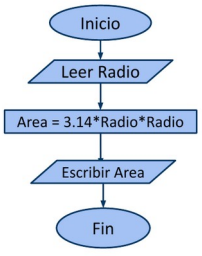
- Nombre y entorno del programa: tipos, nombre de las variables y constantes que se utilizan.
- Desarrollo ordenado y estructurado del algoritmo: esta segunda parte se corresponde con el diagrama de flujo.

Un ejemplo de algoritmo diseñado con Pseudocódigo sería:

```
PROGRAMA NombreDelPrograma
VARIABLES
    Declaración de variables
ALGORITMO
    Cuerpo del programa
FIN
```

Diseñar un algoritmo que obtenga el área de un círculo a partir de su radio

```
PROGRAMA AreaCirculoPrograma
VARIABLES
    Area, Radio REAL
ALGORITMO
    LEER Radio
    Area = 3.14 * Radio * Radio
    ESCRIBIR Area
FIN
```



Los algoritmos están formados por **datos** (se almacenan en variables y constantes), **expresiones** (combinación de constantes, variables, operadores, paréntesis y nombres de funciones.) e **instrucciones** (acciones básicas que implementan los algoritmos, como leer o escribir en variables.). Veamos cada uno de ellos:

- Como hemos dicho anteriormente, los **datos** que vamos a utilizar son los enteros, reales, caracteres y booleanos. Para facilitar la lectura del código, el tipo de datos de las variables lo vamos a escribir en mayúsculas. Por ejemplo:

```
PROGRAMA area
VARIABLES
    base, altura ENTERO
ALGORITMO
    .....
FIN
```

```
PROGRAMA datos
VARIABLES
    nombre, apellidos TEXTO
    mayorEdad BOOLEAN
ALGORITMO
    .....
FIN
```

- Las **expresiones** las podemos clasificar en aritméticas y lógicas. Las siguientes figuras representan las más utilizadas:

Operador	Operación	Operador	Operación
+	Suma o signo	+=	Suma y asignación
-	Resta o signo	-=	Resta y asignación
*	Multiplicación	*=	Multiplicación y asignación
/	División	/=	División y asignación
%	Módulo	%=	Módulo y asignación
++	Incremento en 1	--	Decremento en 1

Operador	Operación
==	Igual
!=	Distinto
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Además, las expresiones pueden agruparse a través de operadores lógicos. Este tipo de operadores se utilizan para hacer varias comprobaciones de forma conjunta, dichas expresiones pueden utilizar :

Operandos: constantes, variables y expresiones lógicas.

Operadores relacionales.

Operadores lógicos (**not**, **and**, **or**, **xor**)

Operador	Operación	Significado
!	NOT	Negación lógica
&&	AND	Conjunción lógica (Y lógico)
||	OR	Disyunción (O lógico)
^	XOR	Disyunción Exclusiva (XOR)

x	y	x && y	x || y	x ^ y	!x
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

Fíjate en las siguientes expresiones lógicas:

```
(valor >= 15) && (valor <= 20) //true si valor entre 15 y 20
(valor <= 15) && (valor >= 20) //false, condición imposible
(valor > 15) || (valor == 15) //true si valor es mayor o igual que 15
(valor > 15) || (valor < 15) //true para cualquier valor menos el 15
(valor >= 15) && (valor <= 20) || (valor > 30 ) //true si valor entre 15 y 20
o mayor que 30
!(valor > 15) //true si valor es menor o igual que 15
```

- Respecto a las **instrucciones**, las podemos clasificar en:

  Declarativas: anuncian el uso de variables y constantes, se debe indicar el nombre de la variable y el tipo de dato que contendrá, para diferencia el tipo lo pondremos en mayúsculas, ENTERO o REAL.

```
PROGRAMA area
VARIABLES
    base, altura, area ENTERO
ALGORITMO
    Cuerpo del programa
FIN
```

Primitivas: las ejecuta el procesador de manera inmediata; no dependen de ningún otro objeto o condición.

Entrada: Lee valores y los asigna a un identificador en memoria.

Salida: Envía datos a un dispositivo desde la memoria.

Asignación: Obtiene el resultado de una expresión y la guarda en un identificador en memoria.

Las primitivas que utilizaremos serán LEER y ESCRIBIR, y para diferenciarlas del resto del código, también las escribiremos en mayúsculas, con el contenido entre paréntesis.

Al utilizar la primitiva ESCRIBIR, podemos concatenar texto en lenguaje natural con los valores de las variables, para ello debemos utilizar el operador suma (+).

Ejemplo: ESCRIBIR(“El resultado de la suma es “+resultado). Si la variable ‘resultado‘ tiene guardado en su interior un 20, la salida del programa sería → El resultado de la suma es 20.

```
PROGRAMA area
VARIABLES
    base, altura, area ENTERO
ALGORITMO
    ESCRIBIR (“Introduce la base y la altura”)
    LEER (base, altura)
    area = base*altura
    ESCRIBIR (“El área del rectángulo es “+area)
FIN
```

ENTRADA	SALIDA
base = 3, altura = 4	El área del rectángulo es 12

Estructuras de control: Evalúan expresiones lógicas para controlar la ejecución de otras instrucciones o alterar el orden de ejecución normal. Tenemos las siguientes:

- Alternativas: Se ejecuta una instrucción o conjunto de instrucciones después de evaluar una condición. Pueden ser simples, dobles o múltiples

SIMPLE:

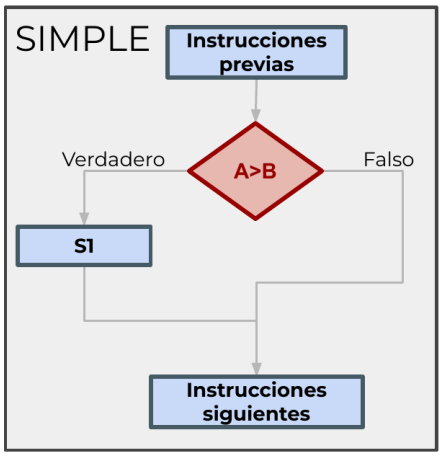
```
SI (condición )
    instrucción1
    instrucción2
instrucción3
```

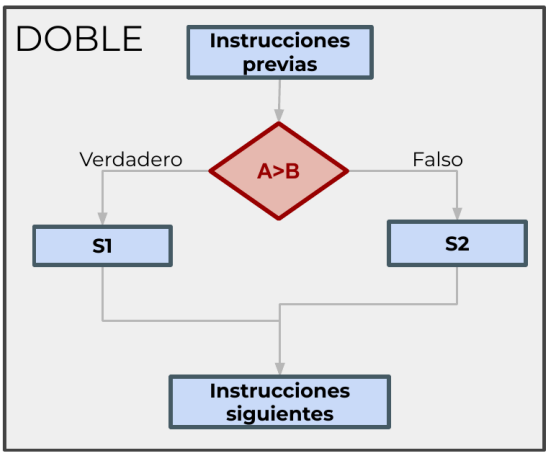
DOBLE:

```
SI (condición )
    instrucción1
    instrucción2
SINO
    instrucción3
instrucción4
```

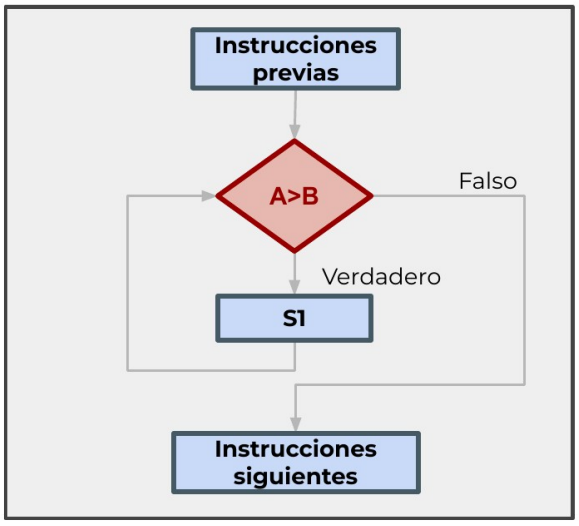
ANIDADA:

```
SI (condición1 )
    instrucción1
    instrucción2
SINO
    SI (condición2 )
        instrucción3
        instrucción4
    SINO
        instrucción5
instrucción6
```





- Repetitivas: Indica un bloque de código que puede ejecutarse más de una vez. Las siguientes figuras representan los tres bloques.



MIENTRAS

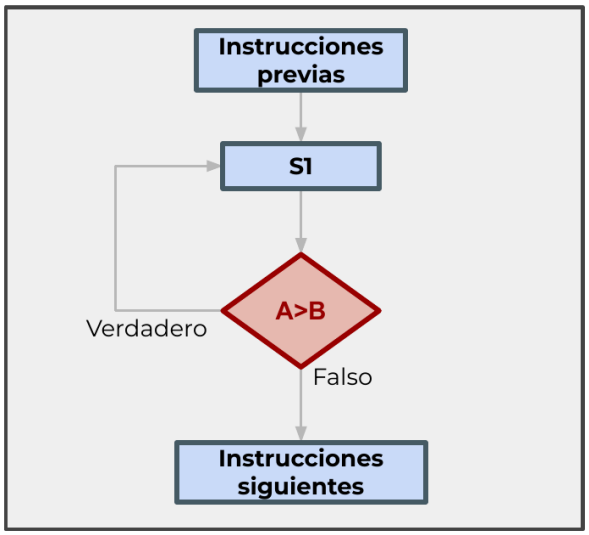
```
MIENTRAS:(condición )

HACER
    instrucción1
    instrucción2
    ........
    instrucciónN

instrucción4
```

✓ *Se ejecutará mientras la condición sea cierta.*

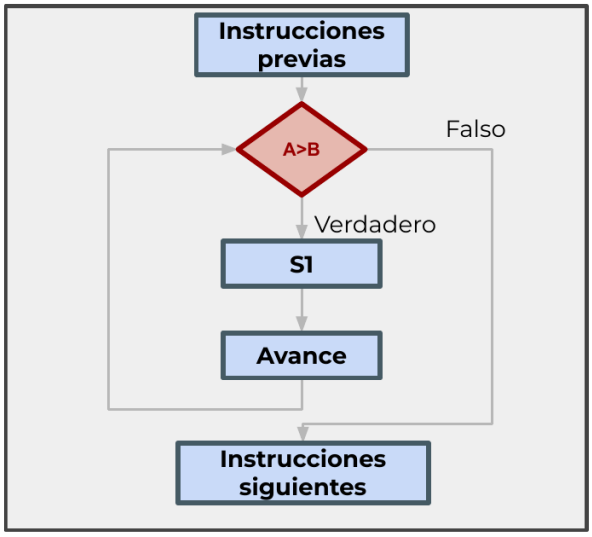
✓ *Es posible que nunca se ejecuten las instrucciones.*



REPETIR...MIENTRAS

```
REPETIR
    instrucción1
    instrucción2
    ........
    instrucciónN
MIENTRAS(condición)
instrucción4
```

- ✓ *Se ejecutará mientras se cumpla la condición.*
- ✓ *Las instrucciones se ejecutan como mínimo una vez.*



PARA...

```
PARA (Vi; Vf; Vc)
    instrucción1
    instrucción2
    ........
    instrucciónN
instrucción4
```

- ✓ *Se ejecutará mientras se cumpla la condición.*
- ✓ *Las instrucciones se ejecutan un número determinado de veces.*
- ✓ *Se indica el valor inicial(Vi), valor final(Vf) y el incremento(Vc).*

Para representar el código dentro de una estructura de control, lo escribiremos tabulando el código, es decir, dejando un margen por la izquierda. En caso de querer representar estructuras anidadas, podremos ir añadiendo tabulaciones a medida que las necesitemos.