

# Introducción a la programación

Sitio: [Centros - Cádiz](#)  
Curso: Programación  
Libro: Introducción a la programación

Imprimido por: Barroso López, Carlos  
Día: martes, 21 de mayo de 2024, 23:23

# Tabla de contenidos

## 1. Programa informático

## 2. Algoritmo

2.1. Pseudocódigo

2.2. Diagrama de flujo

2.3. Diagrama N-S

## 3. Lenguajes de programación

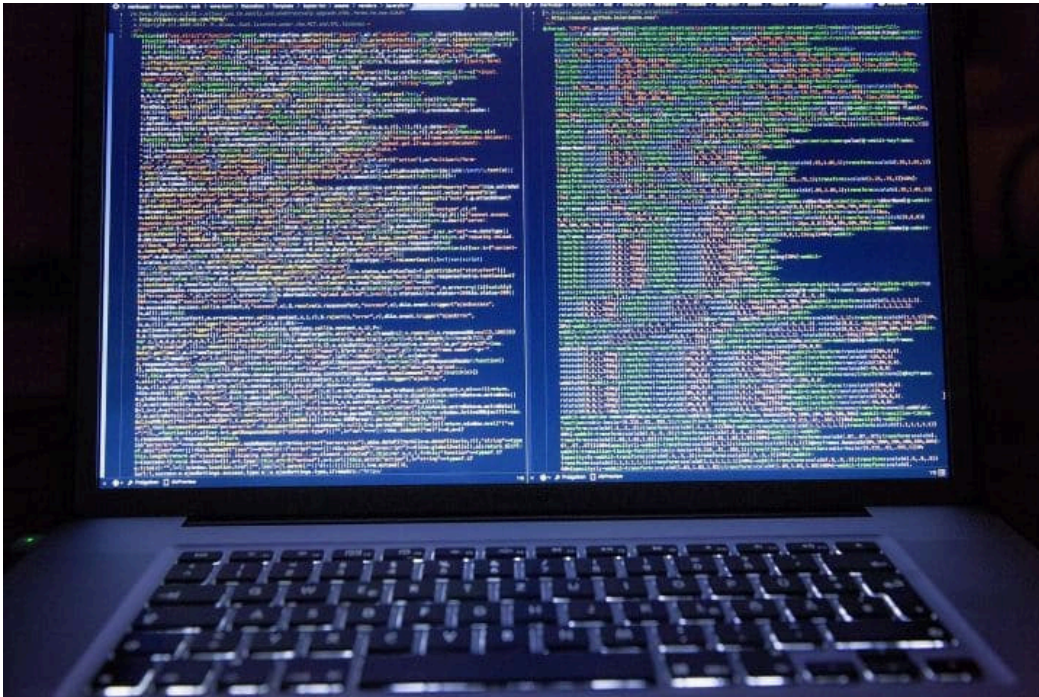
3.1. Traducción a código máquina

3.2. Lenguajes de alto y bajo nivel

3.3. Lenguajes + populares

## 1. Programa informático

La *programación* consiste en la creación de un conjunto de instrucciones u órdenes que un ordenador puede ejecutar, denominado **programa** o aplicación informática.



Un ordenador únicamente puede interpretar y ejecutar programas escritos en lenguaje o **código máquina**. Este lenguaje es fuertemente **dependiente del procesador** que se trate y **complejo** de utilizar.

```

-u 100 1a
OCFD:0100 BA0B01      MOV    DX,010B
OCFD:0103 B409      MOV    AH,09
OCFD:0105 CD21      INT     21
OCFD:0107 B400      MOV    AH,00
OCFD:0109 CD21      INT     21
-d 10b 13f
OCFD:0100                48 6F 6C 61 2C
OCFD:0110                6E 20 70 72 6F 67
OCFD:0120 20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67
OCFD:0130 72 61 6D 61 20 68 65 63-68 6F 20 65 6E 20 61 67
OCFD:0140 73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20
OCFD:0150 57 69 68 69 69 70 65 64 69-61 24

```

*Ej.: Direcciones de memoria (azul), instrucciones máquina (rojo), instrucciones en ensamblador (violeta).*

Como solución, se han desarrollado **lenguajes de alto nivel** independientes del procesador, **más comprensibles**, y que son traducibles a código máquina.

Existen multitud de lenguajes con distintas características y orientados a diferentes aplicaciones.



El propósito de un programa es **resolver un problema** a través del ordenador, para ello, es necesario realizar los siguientes pasos:

1. Análisis del problema.
2. Diseño del **algoritmo**: secuencia ordenada de pasos que conducen a la solución del problema.
3. Implementación en un *lenguaje de programación*.

## 2. Algoritmo

El **programador** es ante todo una persona que resuelve problemas, por lo que se hace necesario aprender a resolverlos de un modo riguroso y sistemático. El eje central de esta metodología es el concepto de **algoritmo**.

Un algoritmo se puede definir mediante la descripción abstracta de las **operaciones** que debe realizar el ordenador, el **orden** en que deben ejecutarse y los **datos** a manipular, y que conduce a la solución del problema.

El diseño de algoritmos es una **labor creativa**, aunque existen técnicas de desarrollo que facilitan la labor.

### 1. Diseño de algoritmos

Definición:

*Un algoritmo es una **secuencia no ambigua, finita y ordenada de pasos para resolver un problema**.*

Los algoritmos son **independientes del lenguaje de programación** en que posteriormente se implementen (programa), y de la computadora donde se ejecuten.



El **diseño** requiere de creatividad y conocimientos de técnicas de programación. El diseño de un buen algoritmo que resuelva un problema concreto es lo que requiere de un **mayor esfuerzo**, y facilitará el resto de pasos para el desarrollo del programa.

## 2. Características

Las características que debe cumplir un buen algoritmo son:

1. **Preciso:** debe indicar de forma clara y sin ambigüedades el orden de realización de cada paso.
2. **Finito:** con un comienzo y un final.
3. **Definido:** siempre obtiene los mismos resultados para los mismos datos de entrada.
4. **Legible:** fácil de leer y entender.
5. **Portable:** debe adaptarse con facilidad a cualquier lenguaje de programación y entorno.
6. **Robusto:** siempre obtiene una respuesta, sean cuales sean los datos de entrada o circunstancias en las que se ejecute.
7. **Eficaz (fiable):** obtiene resultados exactos y precisos.
8. **Eficiente:** debe ser lo más rápido y consumir los menos recursos posibles.

El equilibrio entre estas dos últimas hace que sea **efectivo**.

## 3. Técnicas descriptivas

Existen diferentes **herramientas para representar algoritmos** de forma independiente al lenguaje de programación que se vaya a utilizar.

Las técnicas descriptivas más utilizadas son:

1. **Pseudocódigo**
2. **Diagramas de flujo**
3. **Diagramas N-S**

## 2.1. Pseudocódigo

### 1. Definición y Características

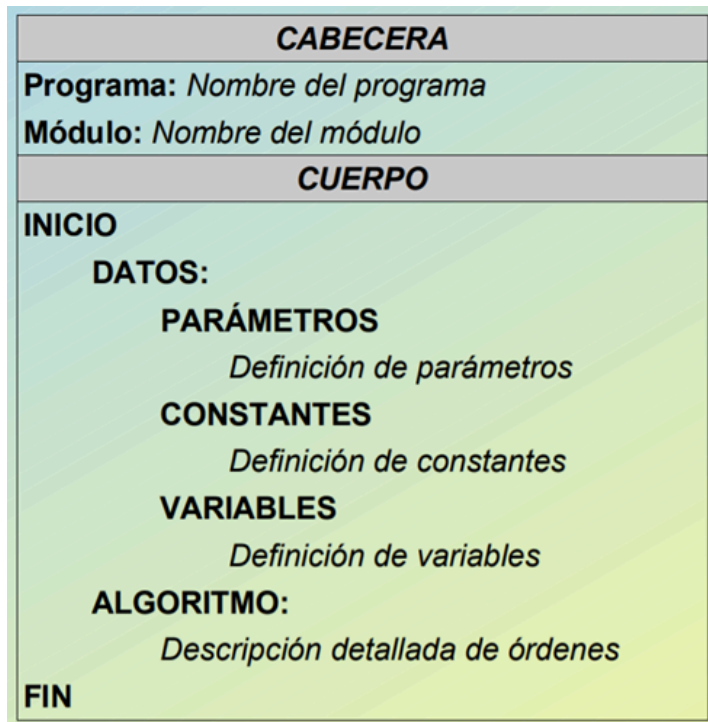
El pseudocódigo es un **lenguaje intermedio** entre el lenguaje natural y el lenguaje de programación.

Esta **notación**, aunque está sujeta a unas determinadas reglas, permite una gran **flexibilidad** a la hora de expresar acciones, lo que facilita el desarrollo del algoritmo.

### 2. Ventajas

- Es **sencillo** de aprender y de utilizar, debido a que utiliza palabras y una sintaxis parecida al lenguaje natural.
- **Facilita** la posterior **codificación del algoritmo** a un lenguaje de programación.

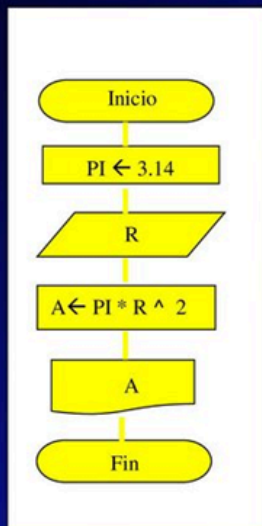
### 3. Estructura general



### 4. Ejemplo: "Calcular superficie de una circunferencia"

Diagrama de flujo / Pseudocódigo / Lenguaje de programación (C)

## Ej: Documentación Externa



Inicio  
**PI ← 3.14**  
**Leer R**  
**A ← PI \* R ^ 2**  
**Mostrar A**  
Fin

- Calcular la superficie una circunferencia teniendo como dato conocido, únicamente el radio del mismo

```
# include <conio.h>
# define PI 3.14
float R, A;
void main ()
{
    clrscr();
    printf("Radio: ");
    scanf ("%f", &R);
    A= PI * R * R;
    printf("\n Area: %f",
        A);
    getch();
}
```






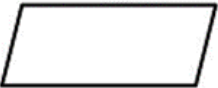
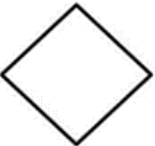
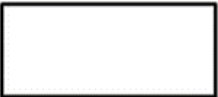
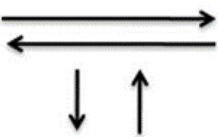
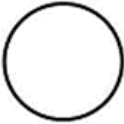
## 2.2. Diagrama de flujo

Representa gráficamente la **secuencia lógica y detallada de las operaciones** que se van a realizar para la resolución del problema.

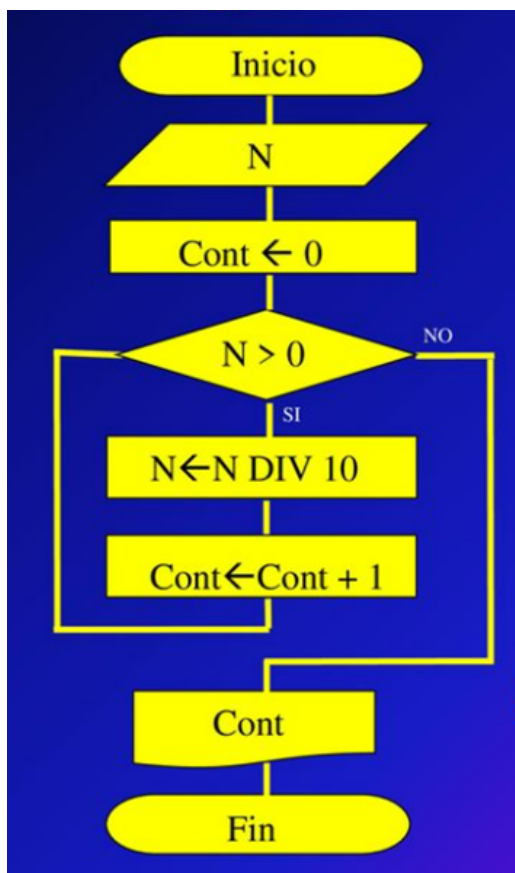
### 1. Reglas

1. Debe tener un **Inicio** en la parte superior, y un **Fin** en la inferior.
2. La secuencia de operaciones debe seguir el orden en el que se van a ejecutar, de **arriba-abajo** y de **izda-dcha**.
3. Todos los elementos deben estar conectados por **líneas rectas**, sin cruces entre ellas.
4. Debe guardar una cierta **simetría**.
5. Las expresiones utilizadas deben ser breves, e independientes del lenguaje.

### 2. Símbolos

SÍMBOLO	NOMBRE	ACCIÓN
	Terminal	Representa el inicio o el fin del diagrama de flujo.
	Entrada y salida	Representa los datos de entrada y los de salida.
	Decisión	Representa las comparaciones de dos o mas valores, tiene dos salidas de información falso o verdadero
	Proceso	Indica todas las acciones o cálculos que se ejecutaran con los datos de entrada u otros obtenidos.
	Líneas de flujo de información	Indican el sentido de la información obtenida y su uso posterior en algún proceso subsiguiente.
	Conector	Este símbolo permite identificar la continuación de la información si el diagrama es muy extenso.

### 3. Ejemplo



## 2.3. Diagrama N-S

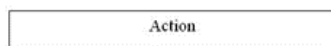
Estos diagramas **combinan** la descripción textual del **pseudocódigo** con la **representación gráfica del diagrama de flujo**.

### 1. Tipos de bloques

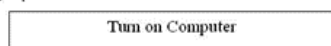
#### i. Bloque secuencial o de proceso

Se ejecuta la acción indicada en el bloque y se pasa al siguiente.

Standard Process Block:

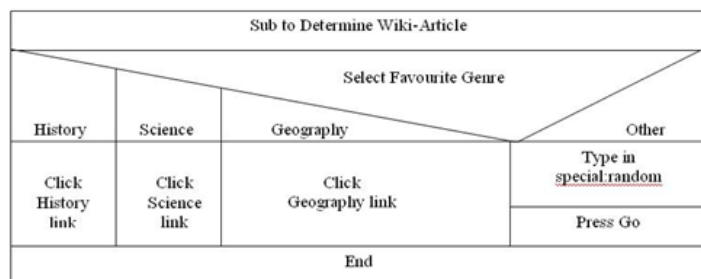
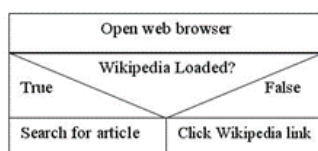


Example|



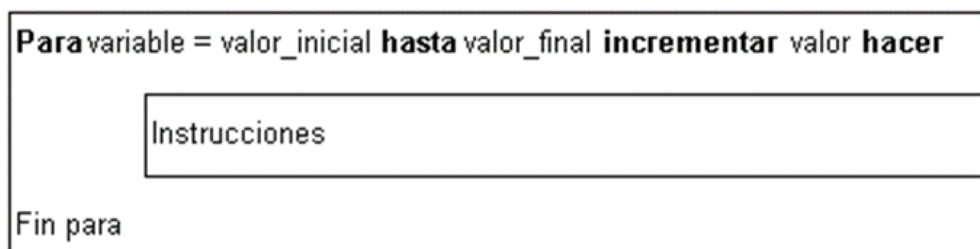
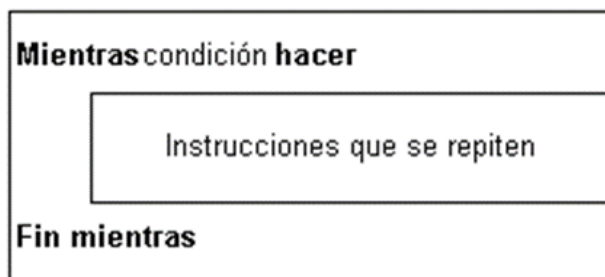
#### ii. Bloque condicional o de decisión

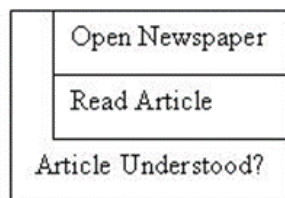
Se ejecuta una acción u otra dependiendo de que se cumpla o no una condición, o del valor tomado por una variable o expresión: simple, doble o múltiple.



#### iii. Bloque de repetición

Repetir un bloque o un conjunto de bloques, siendo controlado por una determinada condición: *Mientras*, *Hacer-Mientras*, *Para*





## 2. Ejemplo

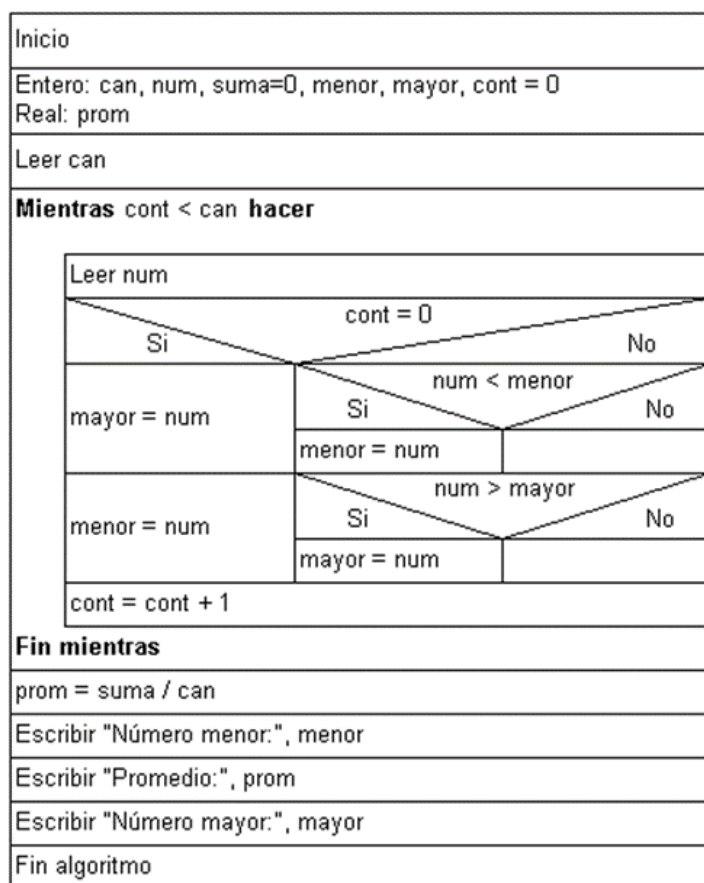
- Datos de entrada: cantidad de números, número

- Datos de salida: promedio, mayor y menor

- Procesos:

o  $\text{suma} = \text{suma} + \text{número}$

o  $\text{promedio} = \text{suma} / \text{cantidad de números}$



### 3. Lenguajes de programación

Para escribir un programa hay que utilizar un lenguaje de programación. Un lenguaje de programación es un tipo de lenguaje que permite al programador **expresar las instrucciones** que quiere que el ordenador ejecute.

#### Sintaxis

Cada lenguaje dispone de una sintaxis que indica cuál es la forma correcta de escribir un programa en dicho lenguaje.

#### Semántica

Además de la sintaxis, un lenguaje debe estar dotado de una semántica (significado), la cual debe estar detallada, sin ambigüedades, para cada una de las **palabras reservadas** que utilice el lenguaje.

C++	Java	Python
<pre>bool checkDivisibility(string num){     int length = num.size();     if(length == 1 &amp;&amp; num[0] == '0')         return true;     if(length % 3 == 1){         num += "00";         length += 2;     }     else if(length % 3 == 2){         num += '0';         length += 1;     }      int sum = 0, p = 1;     for(int i = length - 1;         i &gt;= 0; i--){         int group = 0;         group += num[i--] - '0';         group += (num[i--] - '0') * 10;         group += (num[i] - '0') * 100;         sum = sum + group * p;         p *= (-1);     }      sum = abs(sum);     return (sum % 13 == 0); }</pre>	<pre>static boolean checkDivisibility(     String num){     int length = num.length();     if(length == 1 &amp;&amp; num.charAt(0) == '0')         return true;     if(length % 3 == 1){         num += "00";         length += 2;     }     else if(length % 3 == 2){         num += "0";         length += 1;     }      int sum = 0, p = 1;     for(int i = length - 1; i &gt;= 0; i--){         int group = 0;         group += num.charAt(i-- - '0';         group += (num.charAt(i-- - '0') * 10;         group += (num.charAt(i) - '0') * 100;         sum = sum + group * p;         p *= (-1);     }      sum = Math.abs(sum);     return (sum % 13 == 0); }</pre>	<pre>def checkDivisibility(num):     length = len(num)     if(length == 1 and num[0] == '0'):         return True     if(length % 3 == 1):         num = str(num) + "00"         length += 2     elif(length % 3 == 2):         num = str(num) + "0"         length += 1      sum = 0     p = 1     for i in range(length - 1, -1, -1):         group = 0         group += ord(num[i]) - ord('0')         i -= 1         group += (ord(num[i]) - ord('0')) * 10         i -= 1         group += (ord(num[i]) - ord('0')) * 100         sum = sum + group * p         p *= (-1)      sum = abs(sum)     return (sum % 13 == 0)</pre>

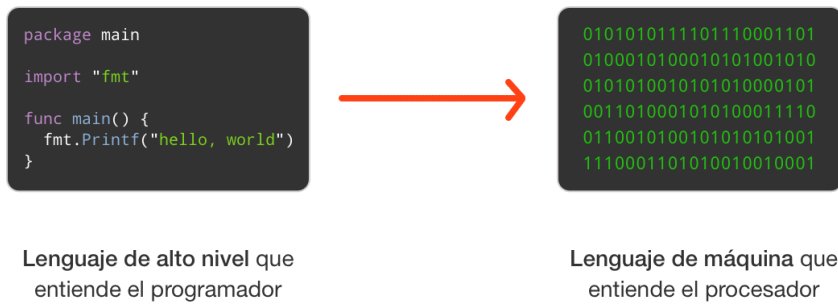
Ej.: Programa escrito en 3 lenguajes de programación diferentes.

Por tanto, un lenguaje de programación es un **conjunto predefinido de palabras y símbolos** que se utilizan siguiendo unas reglas sintácticas, para representar los algoritmos y datos que conforman un programa.





### 3.1. Traducción a código máquina



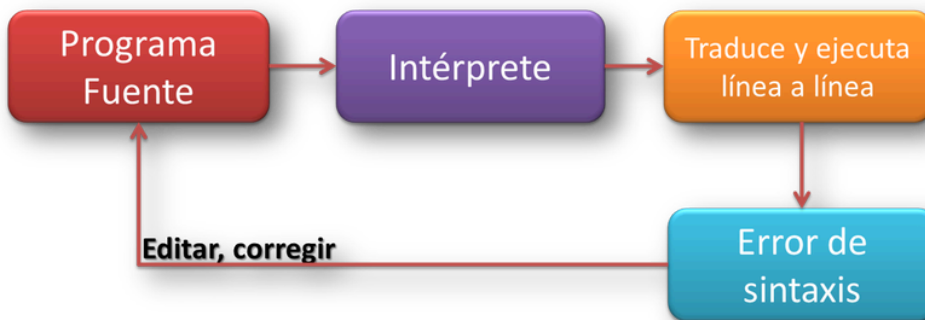
Al crear un programa en un **lenguaje de alto nivel**, las instrucciones utilizadas no son directamente entendibles por el ordenador que las va a ejecutar.

Es necesario un **traductor**, los hay de tres tipos:

#### 1. Tipos de traductores

##### i. Intérprete

**Traduce y ejecuta** un programa fuente instrucción a instrucción, realizando esta acción cada vez que se ejecuta dicho programa.



El intérprete recibe tanto el código a interpretar como los datos con los que se debe ejecutar.



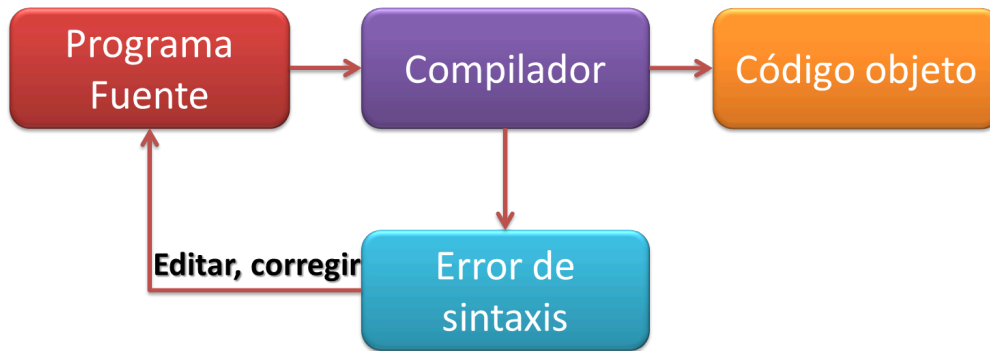
- **Ejecución inmediata** del programa.
- Pero de forma **más lenta**.

Ej.: PHP, Ruby, Python o JavaScript

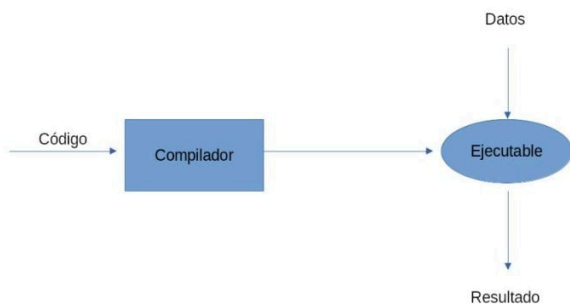
##### ii. Compilador

Traduce el programa completo generando un programa en **lenguaje máquina** (*código objeto*), ejecutable por el procesador.

El compilador analiza en profundidad el código, buscando errores e ineficiencias, intentando eliminarlas. Es un proceso más largo y complejo que el realizado por el intérprete.



El fichero ejecutable puede ahora recibir los datos y procesarlos.



- Ejecución **no inmediata**, requiere la traducción completa del programa.

- Ejecución **más rápida** (lenguaje máquina).

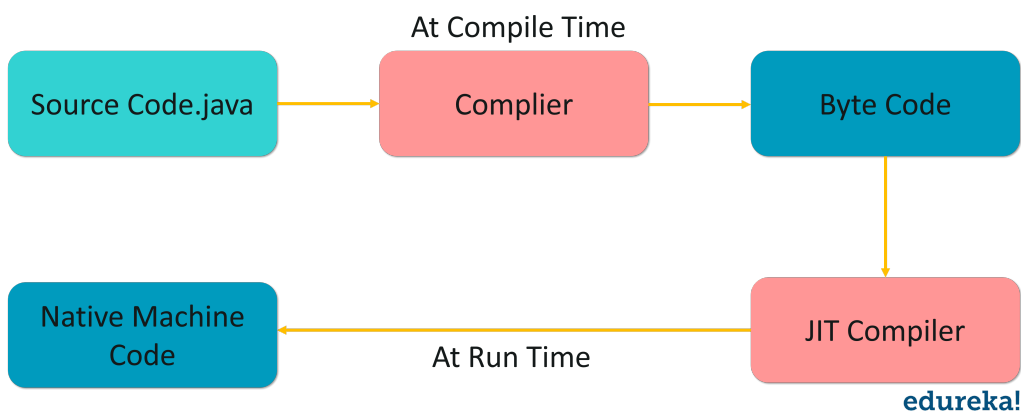
Ej.: C, C++, Haskell, Rust o Go

### iii. Compilación JIT (Just in Time, Justo a tiempo)

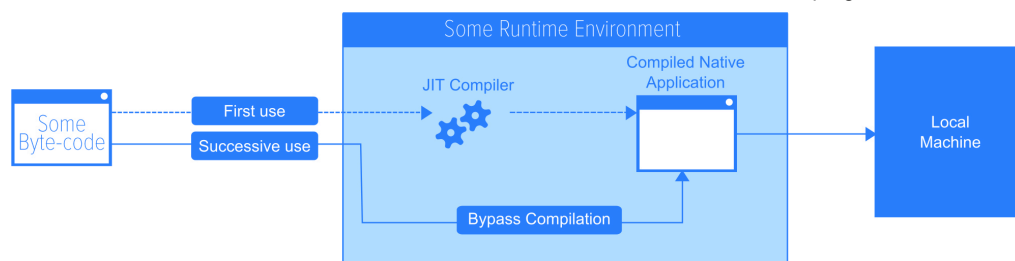
La compilación JIT es un método para **mejorar el rendimiento** de los programas interpretados. Mientras se ejecuta el programa interpretado, el compilador JIT determina el **código utilizado con más frecuencia** y lo compila en código máquina.

Por tanto, la compilación JIT o traducción dinámica, es una compilación que se realiza durante la ejecución de un programa.

Este tipo de compilación la utilizan lenguajes como **Java o C#** que generan un **código intermedio** al ser compilados, el cual debe ser posteriormente interpretado para poder ejecutarse.







- Ejecución no totalmente inmediata.
- Genera un **código más optimizado**, dado que la compilación se realiza en tiempo de ejecución.

Algunos ejemplos de JIT son:

- Java: **JVM** (*Java Virtual Machine*)
- C#: **CLR** (*Common Language Runtime*)
- Android: **DVM** (*Dalvik Virtual Machine*) o **ART** (*Android RunTime*) en versiones más recientes

## 2. Características

### i. Portabilidad

- *Lenguaje interpretado*

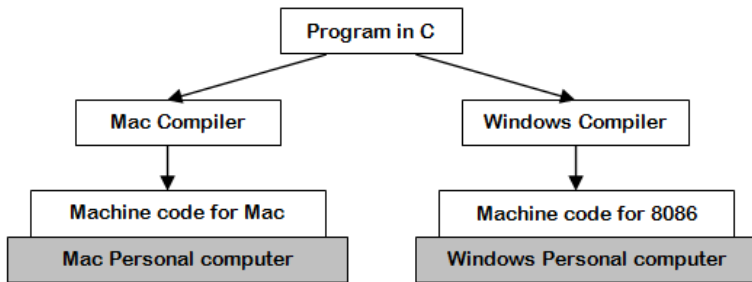
Se podrá ejecutar en cualquier sistema que disponga de un intérprete.

- *Lenguaje compilado*

Sólo podrá ejecutarse en el sistema para el que se compiló.

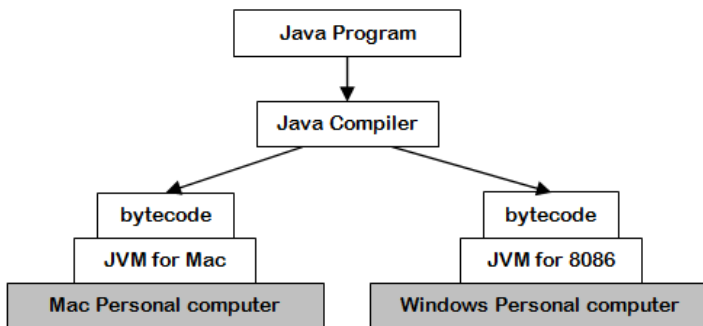
El conjunto de instrucciones de código máquina que entienden diferentes procesadores puede ser distinto. También los programas suelen llamar a rutinas del sistema operativo que son distintas según el sistema.

Estas diferencias obligan a crear un compilador especializado en cada tipo de máquina y sistema operativo aunque el lenguaje de alto nivel sea el mismo.



- *Lenguaje en código intermedio o bytecode*

Podrá ejecutarse en cualquier sistema que disponga de la correspondiente **máquina virtual**, es decir, el intérprete que permite ejecutar dicho lenguaje.



## ii. Rendimiento

- Los **lenguajes compilados** suelen ser más rápidos en ejecución y consumir menos recursos que los interpretados.
- Los **lenguajes interpretados** suelen ser más potentes y flexibles: ejecución de trozos de código, uso de variables no definidas, etc.
- Los **lenguajes intermedios** son más eficientes que los interpretados, manteniendo la potencia y flexibilidad de éstos.

## 3.2. Lenguajes de alto y bajo nivel

Los lenguajes de programación se pueden clasificar según lo específico o general que es respecto a la arquitectura del sistema donde se utiliza (**nivel de abstracción**) en: *lenguajes de alto y bajo nivel*

### 1. Lenguajes de bajo nivel

Son aquellos que están más próximos al lenguaje de la máquina (binario), y tienen un control directo sobre el hardware.

Es por ello que, los programas escritos en estos lenguajes:

- **No necesitan compiladores o intérpretes**, se ejecutan directamente en el procesador.
- **Adaptados** a la arquitectura del **procesador** (juego de instrucciones), por tanto, no se pueden migrar o utilizar en otras máquinas.

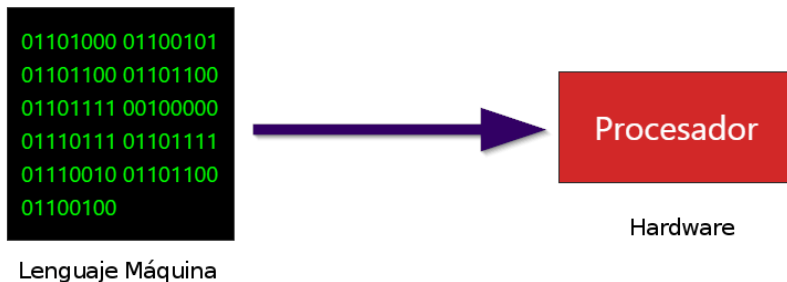
Son lenguajes que aprovechan mejor las características del hardware, pero alejados de la forma de razonar de las personas, lo que los hace **difíciles de entender y utilizar**.

Existen dos tipos de lenguajes de bajo nivel: *máquina y ensamblador*

#### a. Lenguajes máquina

Escritos en **binario o hexadecimal**, describen operaciones básicas que el procesador para el que están desarrollados puede ejecutar de forma directa, por tanto, su ejecución es inmediata, no necesitan traducción.

Fue el *primer lenguaje de programación* utilizado, y es el único lenguaje que el procesador "entiende" directamente.



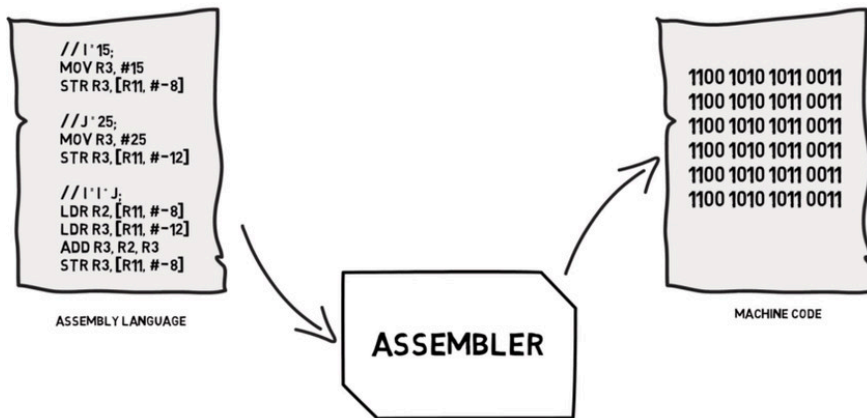
#### b. Lenguajes ensamblador

Permiten escribir instrucciones máquina utilizando una **notación simbólica** o nemotécnica, en lugar de códigos binarios.

En consecuencia, los programas escritos en ensamblador se traducen a código máquina de forma inmediata.

Características:

- Utilizan **códigos de operación** de 3 o 4 letras para indicar la operación a realizar (ej.: ADD para sumar), que suelen ser abreviaturas en inglés.
- Emplean además direcciones simbólicas de memoria (**identificadores**).
- Aceptan el uso de **comentarios**.
- Una instrucción en ensamblador puede dar lugar a varias instrucciones máquina (menos líneas de código).
- Siguen siendo dependientes de la arquitectura del procesador, difícilmente portables, algo más legibles, pero aún alejados del lenguaje humano.



Se utilizan para programar:

- **Tareas críticas** de los SO o de *aplicaciones en tiempo real*.
- Controladores de dispositivos (**drivers**).

Entre otros.

0089814E	57	PUSH EDI	
0089814F	6A 00	PUSH 0	
00898151	FF95 57040000	CALL DWORD PTR SS:[EBP+457]	GlobalAlloc
00898157	8938	MOV DWORD PTR DS:[EAX],EDI	
00898159	8D78 04	LEA EDI,[EAX+4]	
0089815C	8D85 77040000	LEA EAX,[EBP+477]	
00898162	57	PUSH EDI	
00898163	50	PUSH EAX	
00898164	E8 46000000	CALL Decompress	
00898169	57	PUSH EDI	
0089816A	E8 EB000000	CALL PrepareImage	
0089816F	89C6	MOV ESI,EAX	
00898171	31C0	XOR EAX,EAX	
00898173	83EF 04	SUB EDI,4	
00898176	8B0F	MOV ECX,DWORD PTR DS:[EDI]	
00898178	57	PUSH EDI	
00898179	F3:AA	REP STOS BYTE PTR ES:[EDI]	
0089817B	5F	POP EDI	
0089817C	57	PUSH EDI	
0089817D	FF95 5B040000	CALL DWORD PTR SS:[EBP+45B]	
00898183	8BBD 67040000	MOV EDI,DWORD PTR SS:[EBP+467]	
00898189	31C0	XOR EAX,EAX	
0089818B	83EF 04	SUB EDI,4	
0089818E	8B0F	MOV ECX,DWORD PTR DS:[EDI]	
00898190	57	PUSH EDI	
00898191	F3:AA	REP STOS BYTE PTR ES:[EDI]	
00898193	5F	POP EDI	
00898194	57	PUSH EDI	
00898195	FF95 5B040000	CALL DWORD PTR SS:[EBP+45B]	
0089819B	85F6	TEST ESI,ESI	
0089819D	74 07	JE SHORT 008981A6	
0089819F	897424 1C	MOV DWORD PTR SS:[ESP+1C],ESI	
008981A3	61	POPAD	
008981A4	FFEO	JMP EAX	
008981A6	61	POPAD	
008981A7	B8 FFFFFFFF	MOV EAX,-1	
008981AC	C2 0800	RETN 8	

## 2. Lenguajes de alto nivel

Los lenguajes de alto nivel están **más próximos al lenguaje humano** (natural o matemático) y más alejados de la máquina. Por lo tanto, resultan **más fáciles de entender** y son más flexibles a la hora de programar.

**Abstraen la complejidad de la máquina** al programador, siendo independientes de la máquina donde se ejecuten.

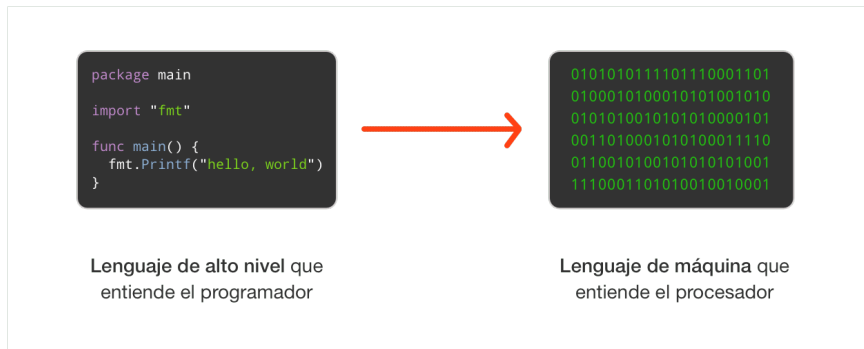
Para ello, utilizan entre otros elementos:

- Reglas sintácticas parecidas al lenguaje natural.
- Expresiones aritmético-lógicas.
- Identificadores lógicos.

*Características:*

- Priman la **facilidad de uso** sobre la eficiencia del programa. Generan menos líneas de código, aunque son algo menos eficientes al no estar optimizados a nivel de la máquina.
- Al ser el código más sencillo y compresible, **reduce costes** de desarrollo y mantenimiento.
- Al ser independiente de la máquina, el mismo código es válido para diferente hardware y SO (**portabilidad**).
- Requiere ser traducido al lenguaje máquina mediante un intérprete o compilador (**no es directamente ejecutable** por el procesador).

Ejemplos: *Pascal, C, C++, Java, etc.*



**Fortran** fue el primer lenguaje de alto nivel que salió al mercado, desarrollado por IBM.

```

PROGRAM TRIVIAL
  INTEGER I
  I=2
  IF(I .GE. 2) CALL PRINTIT
  STOP
END
SUBROUTINE PRINTIT
  PRINT *, 'Hola Mundo'
  RETURN
END

```

**Vídeo: Lenguajes de alto y bajo nivel**

**Comparativa****Lenguajes de bajo nivel**

Gestión de memoria directa.

Dependen del hardware.

Ejecución más rápida.

Difícil de leer y escribir.

Poco apoyo y difíciles de aprender. Gran comunidad detrás.

**Lenguajes de alto nivel**

Necesitan ser interpretados o compilados.

Son independientes del hardware.

Menor rendimiento.

Sintaxis flexible y fácil de leer.

### 3.3. Lenguajes + populares

Algunos de los lenguajes de programación más conocidos son:

#### 1. Fortran



(IBM Mathematical **Formula Translating System**)

Primer lenguaje de alto nivel, fue desarrollado por IBM y está orientado a aplicaciones científicas.

```
PROGRAM par_impar
IMPLICIT NONE

INTEGER :: a=11
LOGICAL :: espar

espar= ( MOD(a,2) == 0 )

IF ( espar ) THEN
    WRITE(*,*)"El numero ",a," es par"
ELSE
    WRITE(*,*)"El numero ",a," es impar"
END IF

STOP
END PROGRAM par_impar
```

#### 2. Lisp



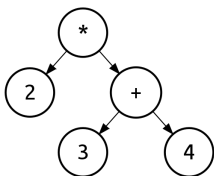
(List Processing)

Diseñado para una fácil manipulación de cadenas de datos, fue utilizado para la traducción automática de textos.

La capacidad de LISP para calcular con **expresiones simbólicas** en lugar de números lo hace conveniente para para la representación de conocimiento, utilizado en aplicaciones de **Inteligencia Artificial (IA)**.

Todo el código del programa es escrito como *expresiones S*, o listas entre paréntesis.

Ej.: (\* 2 (+ 3 4)) *equivale a* 2\*(3+4)



```
; LISP
; ¡Hola,mundo!

(print "Hola, mundo!")

; Factorial de n

(defun factorial (n)
  (if (= n 0) 1
      (* n (factorial (- n 1)))))
```

### 3. Cobol



(**CO**mmun **B**usiness-**O**riented **L**anguage, *Lenguaje Común Orientado a Negocios*)

COBOL se utiliza principalmente en **sistemas de gestión** con grandes volúmenes de datos, como son: aplicaciones comerciales, financieras o administrativas para empresas y gobiernos.

Fue diseñado para escribir programas autodocumentados, mediante separación en divisiones. Buscaba acercarse lo más posible al inglés evitando el uso de símbolos.

```
*> COBOL
*> Multiplicando 2 números

IDENTIFICATION DIVISION.
PROGRAM-ID. VERBS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM1 PIC 9(9) VALUE 10.
01 NUM2 PIC 9(9) VALUE 10.
01 NUMC PIC 9(9).

PROCEDURE DIVISION.
COMPUTE NUMC = (NUM1 * NUM2).
DISPLAY "NUMC:" NUMC
STOP RUN.
```

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
VIEW GHIL.BRYAN.CNTL (AMORTCAL) - 01.02 Columns 00001 00072
Command ==> Scroll ==> CSR
000252 * CALCULATE SIMULATED AMORT POSITION
000253
000254 COMPUTE WS-PAYMENT-AMOUNT =
000255 (WS-CURR-LIMIT) *
000256 WS-MTH-INT-RATE *
000257 (((CP-1 + WS-MTH-INT-RATE) ** WS-TERM) /
000258 (((CP-1 + WS-MTH-INT-RATE) ** WS-TERM) - CP-1)).
000259
000260 COMPUTE WS-AMORT-BALANCE = WS-PAYMENT-AMOUNT *
000261 (CP-1 - (CP-1 + WS-MTH-INT-RATE) ** (WS-TERM-1))
000262 / (WS-MTH-INT-RATE).
000263
000264 MOVE WS-AMORT-BALANCE TO FD-AMORT-AMOUNT.
000265 DISPLAY 'AMORT AMT ' FD-AMORT-AMOUNT.
000266
000267 BB200-999-EXIT.
000268 EXIT.
000269 EJECT
000270 *
```

### 4. Basic

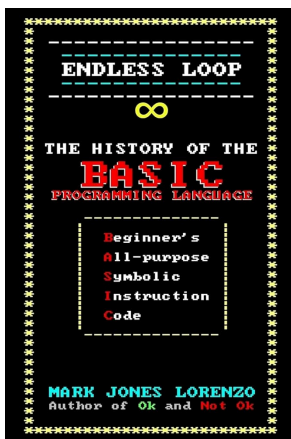




(Beginners' All-purpose Symbolic Instruction Code, Código simbólico de instrucciones de propósito general para principiantes)

Originalmente fue desarrollado como una **herramienta de enseñanza**. Es fácil de aprender, lo que lo hizo muy popular.

Visual Basic y Visual Basic .NET son evoluciones de este lenguaje.



```
PRINT ";Hola Mundo!"
```

```
10 INPUT "Cuál es su nombre:"; NN$
20 PRINT "Bienvenido al 'asterisquero' ";NN$
25 PRINT
30 INPUT "con cuántos asteriscos inicia [Cero sale]:"; N
40 IF N<=0 THEN GOTO 200
50 AS$=""
60 FOR I=1 TO N
70   AS$=AS$+"*"
80 NEXT I
90 PRINT "AQUI ESTAN:"; AS$
100 INPUT "Desea más asteriscos:";SN$
110 IF SN$="" THEN GOTO 100
120 IF SN$<>"S" OR SN$<>"s" THEN GOTO 200
130 INPUT "CUANTAS VECES DESEA REPETIRLOS [Cero sale]:"; VECES
140 IF VECES<=0 THEN GOTO 200
150 FOR I=1 TO VECES
160   PRINT AS$;
170 NEXT I
180 PRINT
185 REM A repetir todo el ciclo (comentario)
190 GOTO 25
200 END
```

## 5. Pascal



Creado por el profesor suizo *Niklaus Wirth* en 1970 con **finés académicos**.

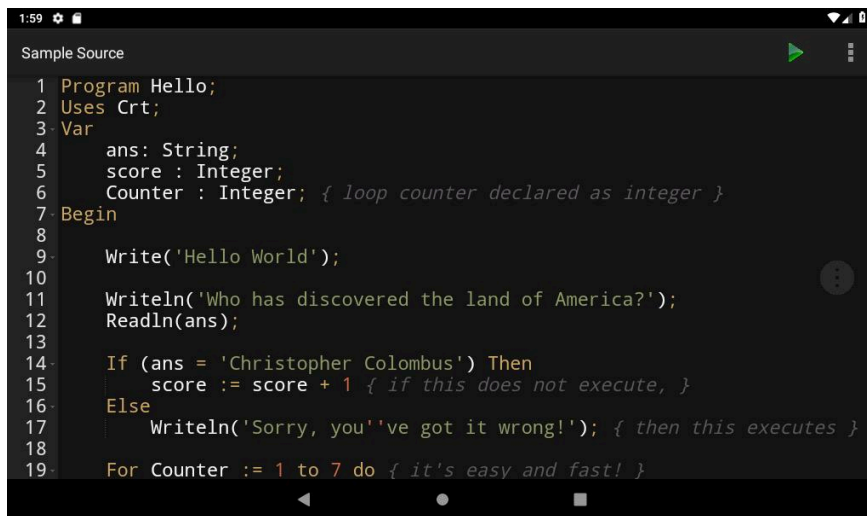
Es un lenguaje sencillo y completo, permite adquirir buenas prácticas de programación, siendo ideal para iniciarse en el mundo de la programación.

Con el tiempo Pascal se popularizó, y ha influenciado a lenguajes tan conocidos con C/C++ o Java.

Este lenguaje se basa en el paradigma de la programación estructurada, el cual se basa en:

- Tipos y estructuras de datos
- Estructuras de control básicas: secuencia, selección e iteración
- Uso de subrutinas o subprogramas

El famoso libro "*Algoritmos + Estructuras de datos = Programas*" de *Wirth*, es una obra referente para enseñar tanto técnicas de algoritmia como los fundamentos de la programación estructurada.



```

1 Program Hello;
2 Uses Crt;
3 Var
4   ans: String;
5   score : Integer;
6   Counter : Integer; { loop counter declared as integer }
7 Begin
8
9   Write('Hello World');
10
11  Writeln('Who has discovered the land of America?');
12  Readln(ans);
13
14  If (ans = 'Christopher Colombus') Then
15    score := score + 1 { if this does not execute, }
16  Else
17    Writeln('Sorry, you've got it wrong!'); { then this executes }
18
19  For Counter := 1 to 7 do { it's easy and fast! }

```

## 6. C



C es un lenguaje de **propósito general** desarrollado por *Dennis Ritchie* a principios de los años 70 en los *Laboratorios Bell*, como evolución del anterior lenguaje B.

Características:

- Es un lenguaje de alto nivel, pero con funcionalidades de bajo nivel, como son: *operaciones a nivel de bits, uso de punteros para gestión de memoria, etc.*
- Genera programas muy eficientes.

Esto lo hace muy **adecuado** para la programación de **software de sistema** (SO, drivers, etc.).

El núcleo del SO Linux está programado con este lenguaje, salvo pequeñas secciones de código escritas con el lenguaje ensamblador.

```

/*
 * C Program to Print "Hello World"
 */
#include <stdio.h>

int main(){
    printf("Hello World");
    return 0;
}

//Site : BTechgeeks.com

```

```

/* Suma de n números */

#include <stdio.h>
int main() {
    int num=0,suma=0;

    do {
        suma=suma+num;
        printf("un número: ");
        scanf("%d",&num);
    } while(num>=0);
    printf("suma es: %d",suma);
    return 0;
}

```

## 7. C++



Evolución del lenguaje C que incluye soporte para la **Programación Orientada a Objetos (POO)**.

En este sentido, C++ es un lenguaje híbrido, debido a que permite programar de forma estructurada y orientada a objetos.

```

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}

```

## 8. Ada



Desarrollado bajo encargo del *Departamento de Defensa de los Estados Unidos* en el año 80.

Lenguaje que destaca por la seguridad, con una filosofía orientada a la reducción de errores comunes y difíciles de descubrir. Para ello se basa en un tipado muy fuerte y en chequeos en tiempo de ejecución.

Ada se usa principalmente en entornos en los que se necesita una gran **seguridad y fiabilidad**, como por ejemplo, los **sistemas en tiempo real**: industria aeroespacial, aeronáutica, control del tráfico aéreo, entre otros.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
  Put_Line ("¡Hola, mundo!");
end Hello;
```

## 9. Java



Desarrollado por *Sun Microsystems* (adquirida posteriormente por Oracle) a mediados de los años 90.

Es el lenguaje **orientado a objetos** más popular.

Características:

- Su sintaxis deriva en gran medida de C/C++.
- El código Java es compilado a *bytecode*, que puede ejecutarse en cualquier **máquina virtual de Java (JVM)** sin importar la arquitectura de la computadora donde se aloja.
- Dispone de recolector de basura (libera memoria no usada) y una amplia **biblioteca de clases**.

The screenshot shows an IDE window titled 'HolaMundo.java'. The code is as follows:

```
1
2 public class HolaMundo {
3
4     public static void main(String[] arg){
5         System.out.println("Hola Mundo");
6     }
7
8 }
```

Below the code editor, there is a 'Console' tab showing the output of the program:

```
<terminated> HolaMundo [Java Application] C:\Program Files\Java\jre1.8.0_
Hola Mundo
```

## 10. Python

Lenguaje interpretado **multiparadigma, multipropósito y multiplataforma**.

Destaca por su legibilidad y facilidad de uso.

Es uno de los lenguajes más populares actualmente, utilizado para desarrollar aplicaciones de todo tipo, como por ejemplo: *Instagram, Netflix, Spotify, etc.*

[Reiniciar tour para usuario en esta página](#)