

# MÓDULO PROFESIONAL

## ENTORNOS DE DESARROLLO

UD 1 – Elementos de  
desarrollo de software

# CONTENIDO

- Conceptos básicos
  - programa
  - lenguaje de programación
  - algoritmo
- Algoritmos
  - diagramas de flujo
  - pseudocódigo
- Traductores: compiladores e intérpretes
- Código fuente, código objeto y código ejecutable
- Clasificación de los lenguajes de programación
- Características de los lenguajes más difundidos
- Desarrollo de software
  - Ciclo de vida
  - Metodología
  - Roles que interactúan en el desarrollo de software

# PROGRAMA

Es un conjunto ordenado de instrucciones que se dan a la computadora indicándole las operaciones o tareas que se desea realice.

(Introducción a la Informática, Alberto Prieto, McGrawHill)

# PROGRAMADOR

La tarea de un **programador informático** es escoger qué órdenes constituirán un programa de ordenador, en qué orden se deben llevar a cabo y sobre qué datos hay que aplicarlas para que el programa lleve a cabo la tarea que debe resolver.

## **Ejecutar un programa**

Por "ejecutar un programa" se entiende hacer que el ordenador siga todas sus órdenes, desde la primera hasta la última.

# SOFTWARE

Es imposible operar el mundo moderno sin **software**.

Como disciplina, la **ingeniería el software** ha progresado mucho en un corto periodo de tiempo.

¿Pero, el software qué comprende?

# SOFTWARE



- **Programas:** Instrucciones que cuando se ejecutan proporcionan las características, función y desempeño buscados
- **Datos:** este componente incluye los datos necesarios para manejar y probar los programas y las estructuras requeridas para mantener y manipular estos datos.
- **Documentos:** este componente describe la operación y uso del programa.

# LENGUAJE DE PROGRAMACIÓN

Para especificar las órdenes que debe seguir un ordenador, es decir construir un programa, lo que se usa es un lenguaje de programación.

# LENGUAJE DE PROGRAMACIÓN

Se trata de un lenguaje **artificial** diseñado expresamente para crear **algoritmos** que puedan ser llevados a cabo por el ordenador

## **artificial**

Por *artificial* entendemos lo que no ha evolucionado a partir del uso entre humanos, sino que ha sido creado expresamente, en este caso para ser usado con los ordenadores.



# PROGRAMA

Las siguientes son un conjunto de instrucciones de un programa escrito en el **lenguaje de programación Java**. Este conjunto de instrucciones recibe el nombre de **código fuente**.



```
/*  
    El programa Hola Mundo muestra un saludo en la pantalla  
*/  
public class HolaMundo {  
    public static void main(String[] args){  
        //Muestra "Hola Mundo!"  
        System.out.println("Hola Mundo!");  
    }  
}
```

# PROGRAMA

Normalmente, el **conjunto de instrucciones** de un programa se almacena dentro de un **conjunto de archivos**.

Estos archivos los edita el programador (vosotros) para crear o modificar el programa. Para los programas más sencillos basta con un único archivo, pero para los más complejos puede ser necesario más de uno.

# ALGORITMO

Los **algoritmos** son secuencias precisas de instrucciones para resolver un problema.

No debe confundirse **algoritmo** con **programa**. Ya que un programa es la codificación de un algoritmo en un lenguaje de programación.

Un algoritmo puede implementarse en distintos lenguajes de programación

# ALGORITMO

Veamos lo que sería un algoritmo para freír un huevo

- 1 *Poner aceite en la sartén*
- 2 *Colocar la sartén en el fuego*
- 3 *Romper el huevo haciendo caer el contenido en la sartén*
- 4 *Tirar las cáscaras a la basura*
- 5 *Poner sal en la yema*
- 6 *Si el huevo está sólido ir a 7, si no esperar*
- 7 *Servir el huevo, fregar la sartén*
- 8 *Fin*

# ALGORITMO

Las características fundamentales que debe cumplir todo algoritmo son:

- **Un algoritmo debe ser preciso** e indicar el orden de realización de cada paso.
- **Un algoritmo debe estar bien definido.** Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- **Un algoritmo debe ser finito.** Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

# ALGORITMO

En un algoritmo se plasman las tres partes fundamentales de una solución informática: **entrada**, **proceso** y **salida**.

Un algoritmo describe una transformación de los datos de entrada para obtener los datos de salida a través de un procesamiento de la información.

Por ejemplo, en el cálculo de la edad de una persona, conociendo su año de nacimiento, la definición del algoritmo, quedaría de la siguiente manera:

- Entrada: información del año de nacimiento y el actual.
- Proceso: realizar la diferencia del año actual menos el año de nacimiento.
- Salida: visualización del resultado generado. Es decir, el resultado es la edad.

# ALGORITMO

Los algoritmos pueden ser representados simbólicamente por **diagramas de flujo** o en **pseudocódigo (falso lenguaje)**, dónde la secuencia de instrucciones se representa por medio de frases o proposiciones

1. Inicio
2. Leer A, B
3.  $S = A + B$
4. Escribir S
5. Fin

Pseudocódigo Algoritmo para determinar la suma de dos números cualesquiera.

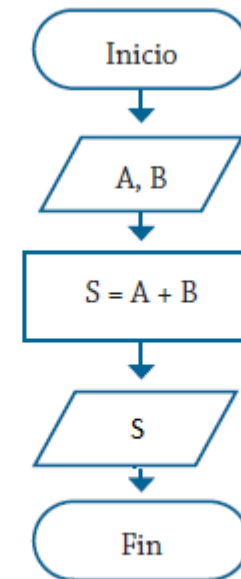


Diagrama de flujo Algoritmo para determinar la suma de dos números.

# ACTIVIDAD ALGORITMOS

## 1. Algoritmo de un cajero de supermercado

El objetivo de esta actividad es aprender a describir un proceso medianamente complejo como una lista enumerada de pasos individuales: un **algoritmo**.

Supone que eres un cajero de un supermercado y que un cliente ha puesto todos los productos en la cinta transportadora.

Describe el algoritmo para atenderle como una lista ordenada de frases cortas en **lenguaje natural**, suponiendo que el pago es en efectivo. Ten en cuenta dentro del algoritmo caso de que un producto no marca ningún precio al pasar el código de barras.



# ACTIVIDAD ALGORITMOS

## 2. Algoritmo para discriminar pares e impares

Supone que deseas hacer un programa que diga si la suma de dos números escritos por el usuario usando el teclado es par o impar.

Describe el algoritmo que debe seguir el ordenador, descrito como una lista de frases cortas en **lenguaje natural**.

No has de mencionar instrucciones ni variables, sólo las tareas generales que hay que hacer, de manera similar a los resultados de los ejercicios anteriores.

# ACTIVIDAD ALGORITMOS

## 3. Algoritmo para adivinar un número

Supone que deseas hacer un programa que se inventa un número y vosotros debéis adivinar.

Describe el algoritmo que debe seguir el ordenador, descrito como una lista de frases cortas en **lenguaje natural**, para decidir si ha ganado o no.

No has de mencionar instrucciones ni variables, sólo las tareas generales que hay que hacer, de manera similar a los resultados de los ejercicios anteriores.

# ALGORITMOS. DIAGRAMA DE FLUJO







Un **diagrama de flujo** es una representación gráfica de un algoritmo, el cual muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema .

Los pasos son representados por varios tipos de bloques y el flujo de ejecución es indicado por flechas que conectan los bloques.

# ALGORITMOS. DIAGRAMA DE FLUJO

Los símbolos utilizados han sido normalizados por las organizaciones **ANSI** (American National Standard Institute) y por **ISO** (International Standard Organization).

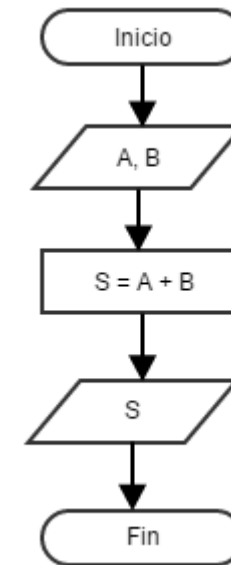
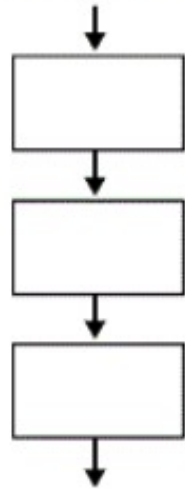
Algunos de los más utilizados son:

	Terminal (representa el comienzo, "inicio", y el final, "fin" de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa
	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.).
	Entrada/Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida").
	Decisión (indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Conector, sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la terminación del primero y otro al comienzo del segundo. Se refiere a la conexión de partes del diagrama en la misma página

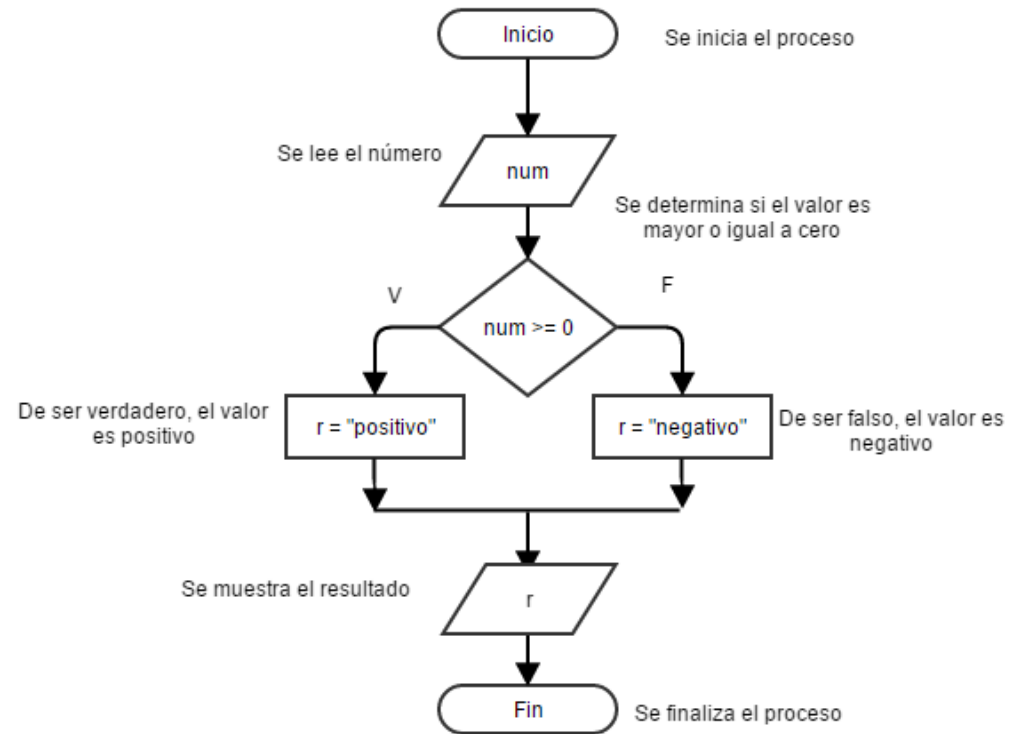
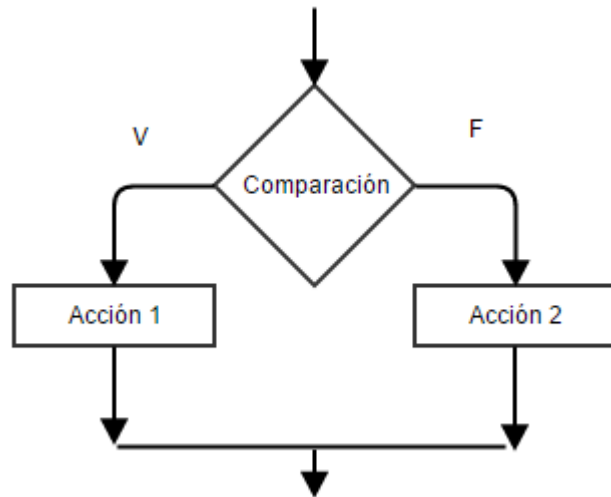
# DIAGRAMA DE FLUJO: ESTRUCTURAS SECUENCIALES

Algoritmo para obtener la suma de dos números cualesquiera.

**Secuencia**

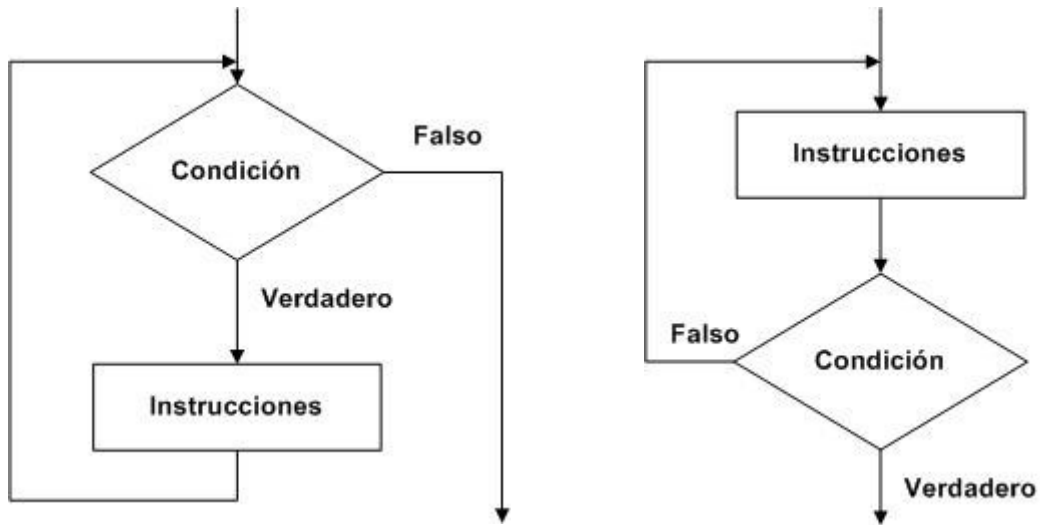


# DIAGRAMA DE FLUJO: ESTRUCTURAS SELECTIVAS

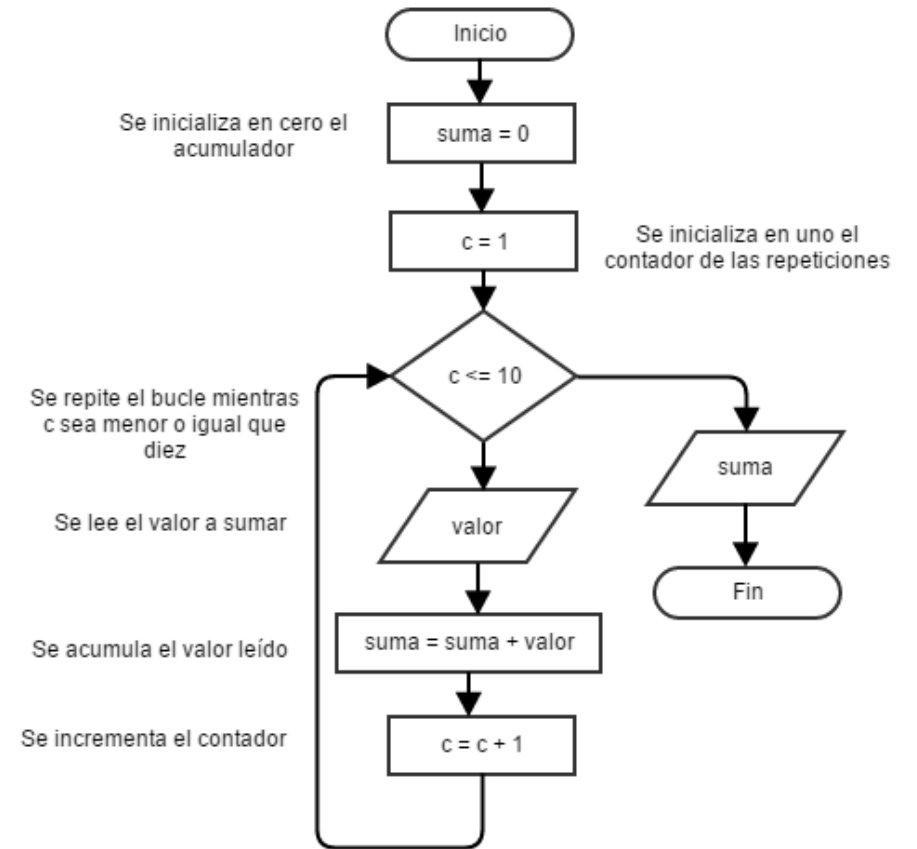


Algoritmo para determinar si un número es positivo o negativo.

# DIAGRAMA DE FLUJO: ESTRUCTURAS REPETITIVAS



Mientras: repite mientras la condición se cumpla  
Hasta: repite hasta que la condición se cumpla



Algoritmo para obtener la suma de diez cantidades

# DIAGRAMA DE FLUJO

Su correcta construcción es sumamente importante porque a partir del mismo se escribe un programa en algún lenguaje de programación.

Si el diagrama de flujo está completo y correcto el paso del mismo a un lenguaje de programación es relativamente simple y directo



# DIAGRAMAS DE FLUJO

**Construcción:** para la construcción de un diagrama de flujo se tienen que seguir los tres pasos siguientes:

1. **Análisis:** identificar datos de entrada y salida, grupo de pasos, puntos de toma de decisión, etc. y ordenarlos de manera cronológica.
2. **Construcción:** organizarlo gráficamente. Normalmente la organización es
  - De arriba hacia abajo
  - De izquierda a derecha
3. **Prueba:** se basa en hacer pruebas para ver si funciona.



# ACTIVIDAD DIAGRAMAS DE FLUJO

1. Se requiere obtener el **área de una circunferencia**. Realiza el diagrama de flujo correspondiente.
2. Desarrolla un algoritmo que permite leer dos valores distintos, determina **cual es el mayor** y lo escribe. Realiza el diagrama de flujo.
3. Desarrolla un algoritmo que permita leer un número cualquiera y escriba si dicho número es **par o impar**. Realiza el diagrama de flujo
4. Realiza el diagrama de flujo para determinar si un **año es bisiesto**
5. Se requiere un algoritmo que dado un número positivo, disminuya su valor hasta llegar a cero (**cuenta regresiva**)
6. Se requiere un algoritmo para obtener la **edad promedio de un grupo de N alumnos**. Realiza el diagrama de flujo correspondiente

# ALGORITMOS. SOLUCIÓN

## EJERCICIO 6

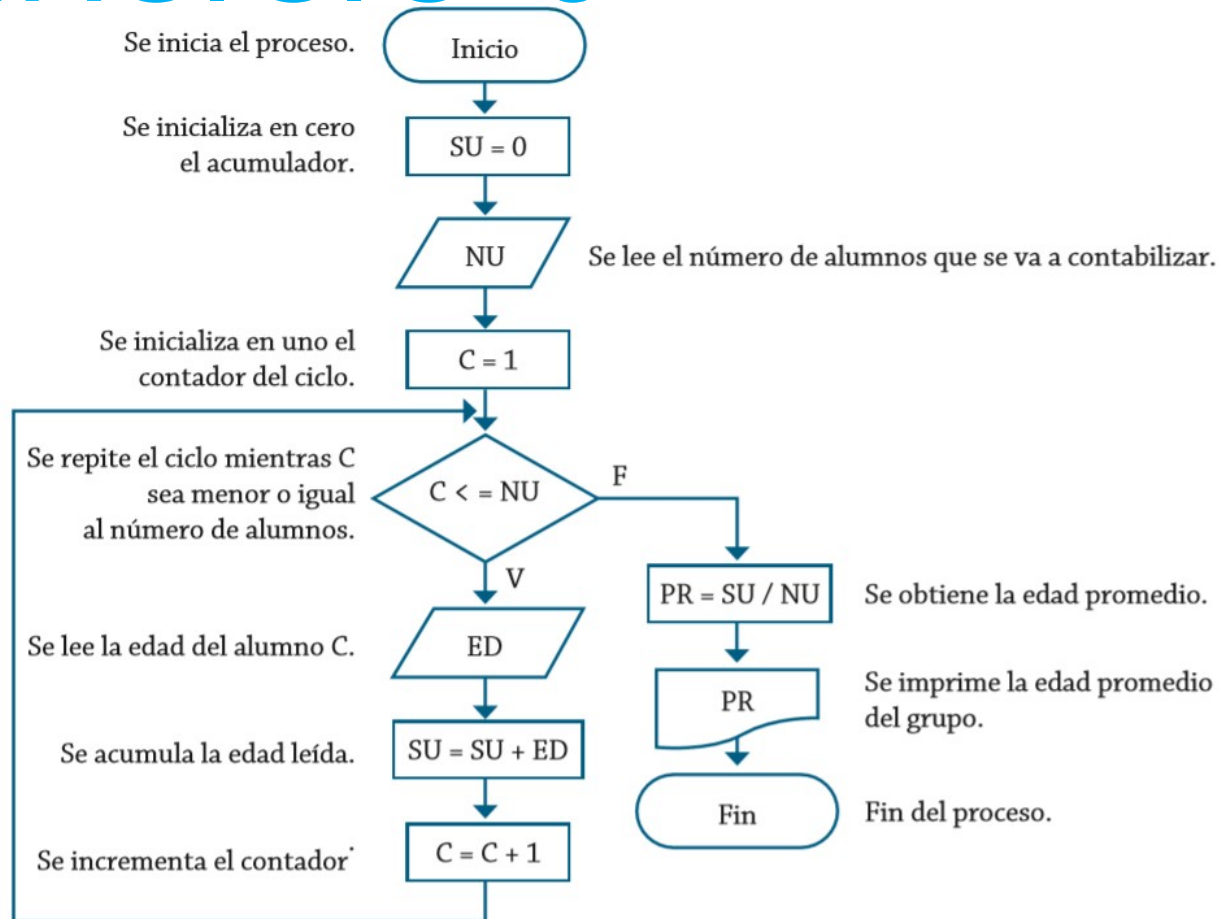
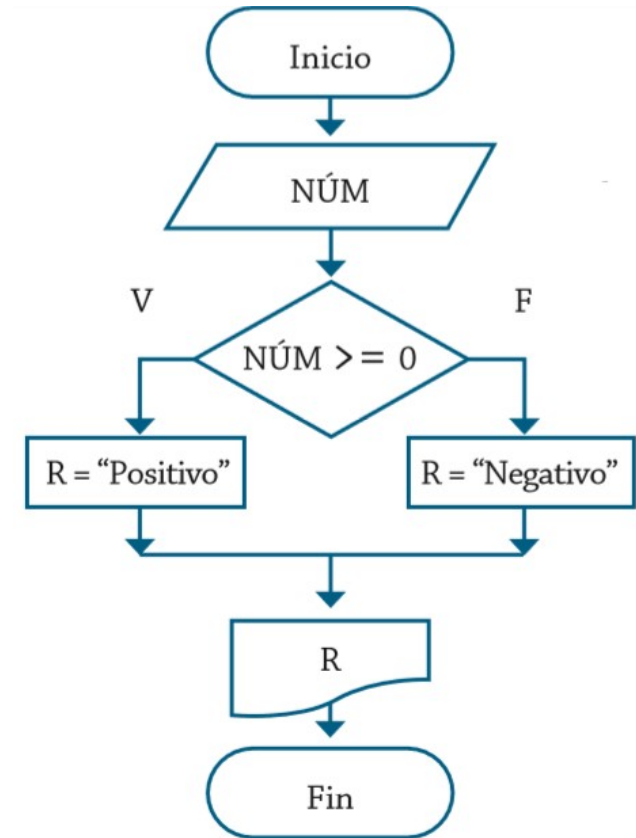
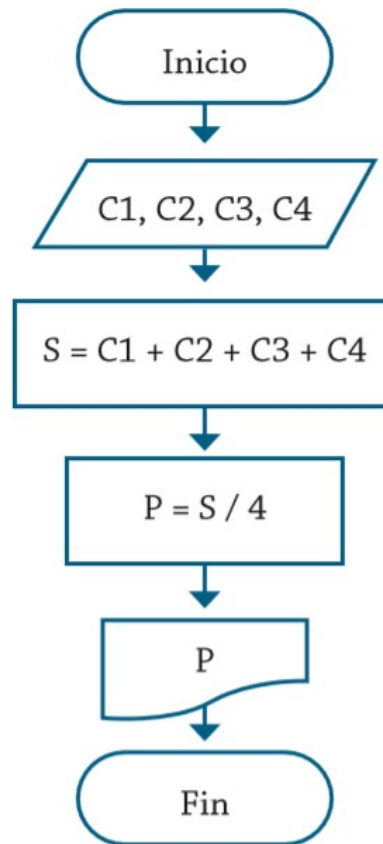


Diagrama de flujo con ciclo Mientras. Algoritmo para obtener la edad promedio de N alumnos

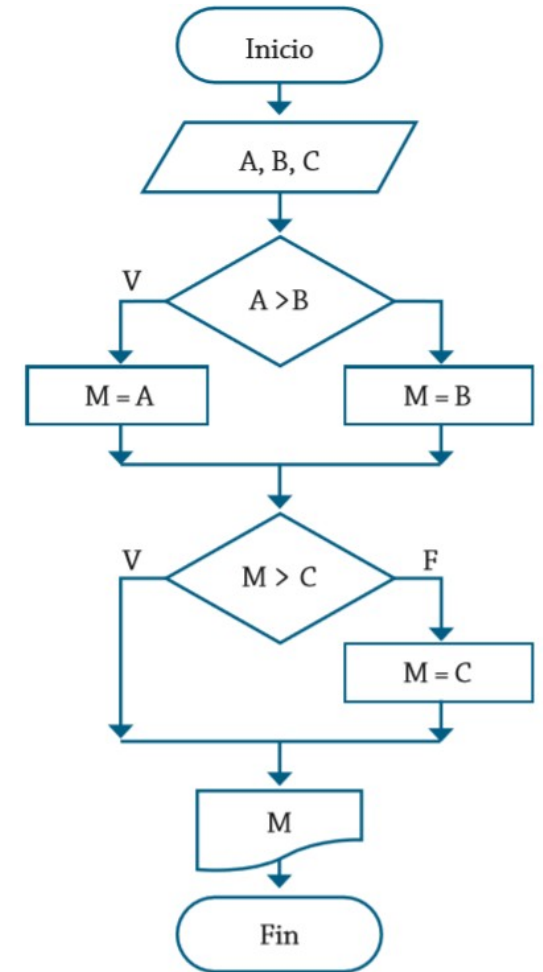
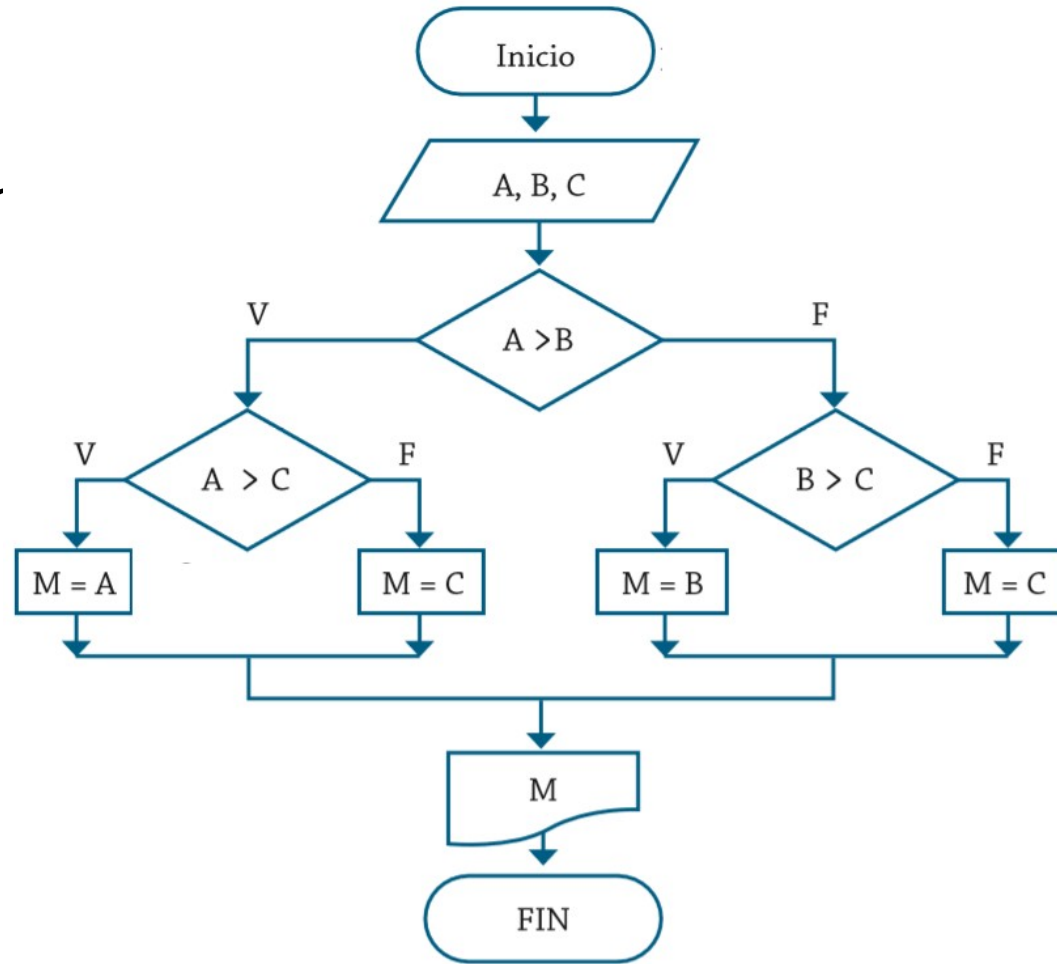
# ACTIVIDAD DIAGRAMAS DE FLUJO

¿cuál es el algoritmo?



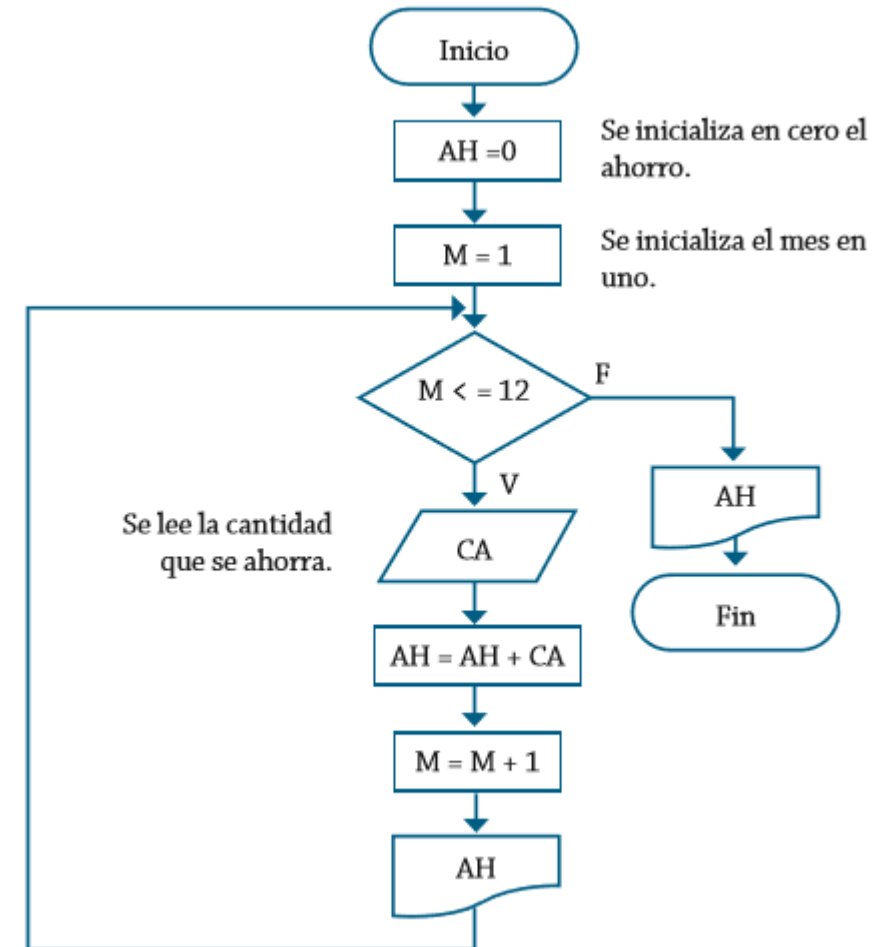
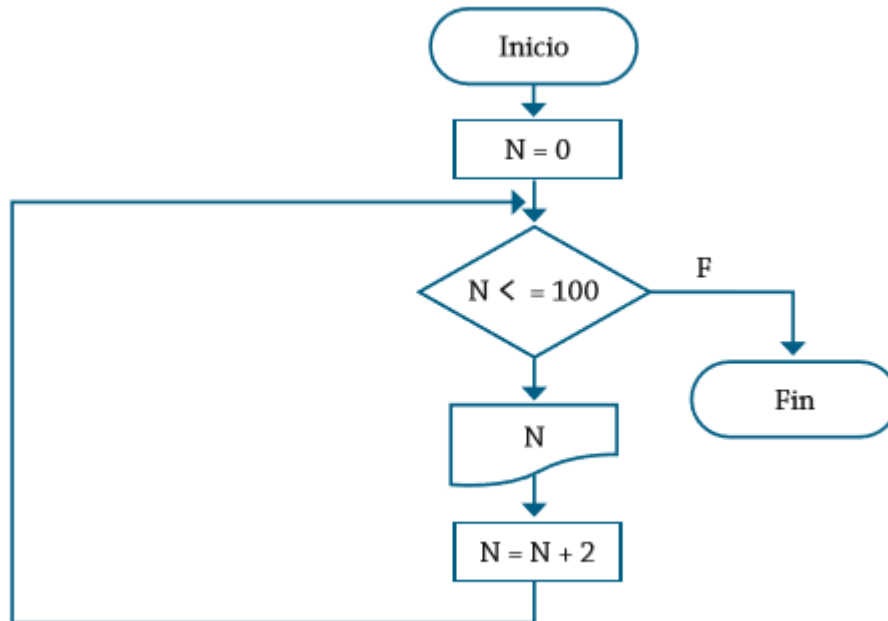
# ACTIVIDAD DIAGRAMAS DE FLUJO

¿cuál es el algoritmo?



# ACTIVIDAD DIAGRAMAS DE FLUJO

¿Cuál es el algoritmo?



# ALGORITMOS. PSEUDOCÓDIGO

Los programas deben ser escritos en un lenguaje que pueda entender el ordenador, pero no olvidemos que nuestra forma normal de expresar algo es en lenguaje natural. De la aproximación entre ambos surge una herramienta para la descripción de algoritmos: el **pseudocódigo**.

Es por tanto un lenguaje algorítmico que permite representar las construcciones básicas de los lenguajes de programación, pero a su vez, manteniéndose próximo al lenguaje natural.

# ALGORITMOS. PSEUDOCÓDIGO

## **pseudocódigo**

El pseudocódigo es un lenguaje informal de alto nivel que usa las convenciones y la estructura de un lenguaje de programación, pero que está orientado a ser entendido por los humanos.



# ALGORITMOS. PSEUDOCÓDIGO

La ventaja del pseudocódigo es que en su uso, en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico.

Además es fácil de aprender y utilizar, es independiente del lenguaje de programación que se vaya a utilizar. Facilita el paso del programa al lenguaje de programación.

# ALGORITMOS. PSEUDOCÓDIGO

Ejemplo de pseudocódigo para el problema del cálculo del valor absoluto de un número

INICIO

    LEER numero

    SI número  $< 0$  ENTONCES

        absoluto = -numero

    EN OTRO CASO

        absoluto = numero

    FINSI

    ESCRIBIR absoluto

FIN

# ALGORITMOS. PSEUDOCÓDIGO

No existe una sintaxis estándar para el pseudocódigo, se utiliza una mezcla de lenguaje natural (utilizando como base la lengua nativa del programador) y una serie de símbolos, términos y otras características propias de los lenguajes de programación de alto nivel.

La escritura de pseudocódigo exige normalmente la indentación (sangría en el margen izquierdo) de diferentes líneas.

# ACTIVIDAD PSEUDOCÓDIGO

1. Se requiere obtener el **área de una circunferencia**. Escribe el pseudocódigo
2. Desarrolla un algoritmo que permite leer dos valores distintos, determina **cual es el mayor** y lo escribe. Escribe el pseudocódigo
3. Desarrolla un algoritmo que permita leer un número cualquiera y escriba si dicho número es **par o impar**. Escribe el pseudocódigo
4. Escribe el pseudocódigo para determinar si un **año es bisiesto**
5. Se requiere un algoritmo que dado un número positivo, disminuya su valor hasta llegar a cero (**cuenta regresiva**). Escribe el pseudocódigo
6. Se requiere un algoritmo para obtener la **edad promedio de un grupo de N alumnos**. Escribe el pseudocódigo

# LENGUAJES DE PROGRAMACIÓN

Igual como hay muchas lenguas diferentes, también hay muchos **lenguajes de programación**, cada uno con sus características propias, que los hacen más o menos indicados para resolver un tipo de tareas u otras.

Todos, sin embargo, tienen una **sintaxis** muy definida, a seguir para que la computadora interprete correctamente cada orden que se le da.

```
std::cout << "Hola, mundo!" << std::endl;
<?php echo 'Hola, mundo!'; ?>
trace("Hola, mundo!");
Hola, mundo!
printf("Hola, mundo!\n");
System.out.println("Hola, mundo!");
document.writeln("Hola, mundo!");
```

# LENGUAJES DE PROGRAMACIÓN

Es exactamente lo mismo que ocurre con las lenguas del mundo: para expresar los mismos conceptos, el español y el inglés usan palabras y normas de construcción gramatical totalmente diferentes entre sí.



**¡Hola Mundo!**



HELLO  
WORLD!

# LENGUAJES DE PROGRAMACIÓN

Las siguientes son un conjunto de instrucciones de un programa escrito en el **lenguaje de programación Java**. Este conjunto de instrucciones recibe el nombre de **código fuente**.



```
/*  
    El programa Hola Mundo muestra un saludo en la pantalla  
*/  
public class HolaMundo {  
    public static void main(String[] args){  
        //Muestra "Hola Mundo!"  
        System.out.println("Hola Mundo!");  
    }  
}
```

El código fuente estará escrito en un lenguaje de programación determinado, elegido por el programador, como pueden ser: C, C++, C#, Java, Perl, Python, PHP.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

Para **crear un programa** lo que se hará será crear un archivo y escribir a un fichero cuyo serie de instrucciones que se quiere que el ordenador ejecute. Estas instrucciones deberán seguir unas pautas determinadas en función del lenguaje de programación elegido.

Además, deberían seguir un orden determinado que dará sentido al programa escrito. Para empezar a programar bastará con un **editor de texto simple**.



# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

Una vez se ha terminado de escribir el programa, el conjunto de archivos de texto resultantes, donde se encuentran las instrucciones, se dice que contienen el **código fuente**. Este código fuente puede ser desde un nivel muy alto, muy cerca del lenguaje humano, hasta un nivel más bajo, más cercano al código de las máquinas, como el código ensamblador.

```
class Triangle {  
    ...  
    float surface()  
        return b*h/2;  
}
```

ALTO NIVEL

```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

BAJO NIVEL

```
0001001001000101  
0010010011101100  
10101101001...
```

CÓDIGO MÁQUINA

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

El **lenguaje máquina** es el único que entiende el ordenador, es su "lenguaje natural". En él solamente se pueden utilizar dos símbolos: el cero (0) y el uno (1).

Por ello, al lenguaje máquina también se le denomina lenguaje binario. La computadora solo puede trabajar con bits, sin embargo, para el programador no resulta fácil describir instrucciones tales como:

10100010

11110011

00100010

00010010

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE

Una dificultad añadida a los lenguajes binarios es el hecho de que son **dependientes de la máquina** –o mejor dicho, del procesador–, es decir, **cada procesador utiliza un lenguaje máquina distinto** –un juego de instrucciones distinto– que está definido en su propio hardware.

En consecuencia, un programa escrito para un tipo de procesador no se podrá usar en otro equipo que utilice un procesador distinto, ya que, el programa no será portable o transportable.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE

Igualmente, puesto que las instrucciones que se pueden escribir en un lenguaje ensamblador siempre están asociadas a las instrucciones binarias de un ordenador en concreto, los **lenguajes ensambladores también son dependientes del procesador.**

Sin embargo, los **lenguajes de alto nivel sí que son independientes del procesador**, es decir, un programa escrito en cualquier ordenador con un lenguaje de alto nivel podrá transportarse a cualquier otra computadora, con unos pequeños cambios o incluso ninguno.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

- La tendencia actual es hacer uso de **lenguajes de alto nivel**. Estos están diseñados para que las personas escriban y entiendan los programas de un modo más fácil que los lenguajes máquina y ensambladores, y por otro lado traducibles a código máquina.
- Pero esto hace aparecer un problema, y es que **los archivos de código fuente no contienen el lenguaje máquina que entenderá el ordenador**. Por lo tanto, resultan incomprensibles para el procesador.
- Para poder **generar código máquina** hay que hacer un **proceso de traducción** desde los mnemotécnicos que contiene cada archivo a las secuencias binarias que entiende el procesador.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

Un **traductor** es un programa que recibe como entrada un texto escrito en un lenguaje de programación concreto y produce, como salida, el lenguaje máquina equivalente. El programa inicial se denomina **código fuente** y el programa obtenido, **código objeto**.

El **código objeto** a veces no es directamente ejecutable.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

- El proceso llamado **compilación** es la traducción del código fuente de los archivos del programa en archivos en formato binario que contienen las instrucciones en un formato que el procesador puede entender.
- Por otro lado, existe un tipo de programas llamados **intérpretes**, los cuales también sirven para traducir el código fuente de un programa a código objeto, pero, su manera de actuar es diferente con respecto a la de un compilador.

# CÓDIGO FUENTE, CÓDIGO OBJETO, CÓDIGO EJECUTABLE.

## Código objeto

Además de todo lo visto hasta ahora, también hay que tener en cuenta que una aplicación informática suele estar compuesta por un conjunto de programas o subprogramas.

Por tanto, el código objeto de todos ellos deberá ser **enlazado** (unido) para obtener el deseado programa ejecutable.

Para ello, se utiliza un programa llamado **enlazador**, el cual generará y guardará, en disco, un archivo ejecutable. En Windows, dicho archivo tendrá extensión (.exe), abreviatura de *executable*.



# TRADUCTORES: COMPILADORES E INTÉRPRETES

Hay dos tipos de traductores: los **compiladores** y los **intérpretes**.

## Compiladores

Los compiladores son traductores que en un único proceso analizan todo el código fuente y generan el código objeto correspondiente almacenando el resultado.

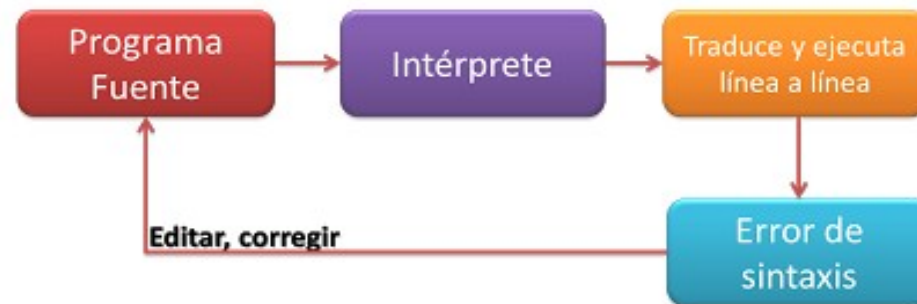
Dependiendo el compilador, el archivo con **código objeto generado puede ser directamente ejecutable o necesitar otros pasos previos a la ejecución**, tales como ensamblado, enlazado y carga.



# TRADUCTORES: COMPILADORES E INTÉRPRETES

## Interpretes

Los intérpretes son traductores que hacen en forma simultánea el proceso de traducción y el de ejecución. Su forma de trabajo es ir analizando bloques del programa fuente, generando el código máquina correspondiente, y ejecutándolo.



# TRADUCTORES: COMPILADORES E INTÉRPRETES

## Interpretes

No crea un archivo almacenable en memoria secundaria para su posterior uso.

Los intérpretes, durante la generación de código, buscan las instrucciones en lenguaje máquina que equivalgan al bloque de instrucciones fuente que se está traduciendo. En general, a cada instrucción de alto nivel le van a corresponder varias en lenguaje máquina.

# ACTIVIDAD

- Razona las ventajas y desventajas de compiladores e intérpretes

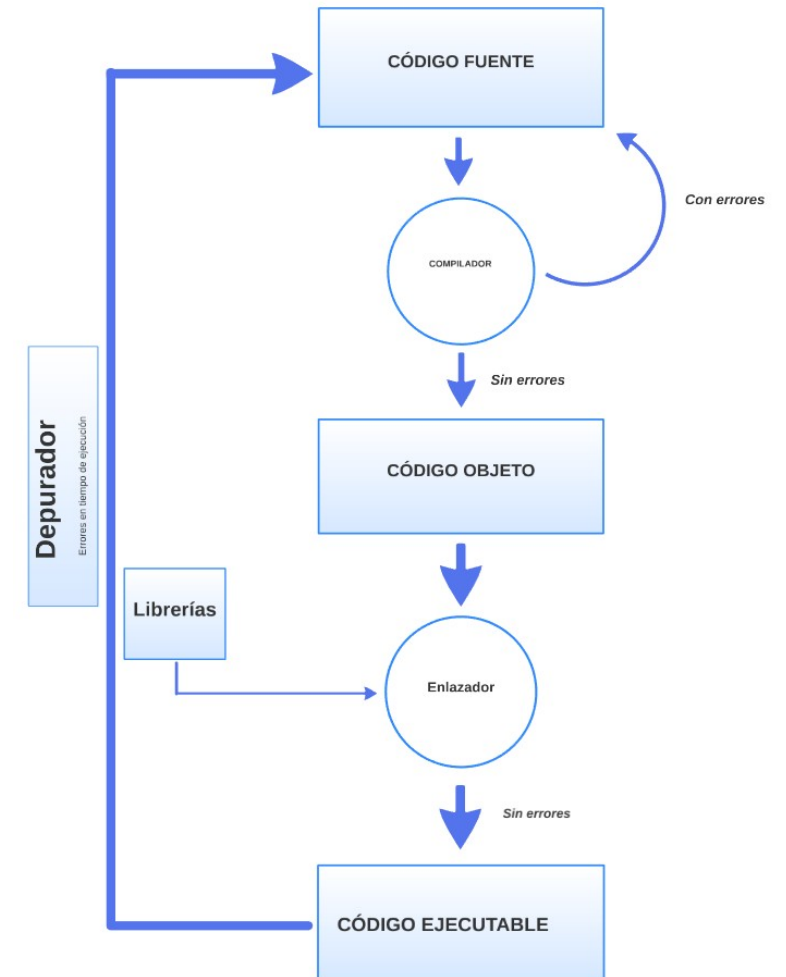
# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

## Compilación y enlace

Hay que tener en cuenta que una aplicación informática suele estar compuesta por un conjunto de programas o subprogramas.

Por tanto, el código objeto de todos ellos deberá ser enlazado (unido) para obtener el deseado programa ejecutable.

Para ello, se utiliza un programa llamado **enlazador**, el cual generará y guardará, en disco, un archivo ejecutable. En Windows, dicho archivo tendrá extensión (.exe), abreviatura de *executable*.



# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

## Fases:

- **Análisis lexicográfico:** se leen de manera secuencial todos los caracteres de nuestro código fuente, buscando palabras reservadas, operadores, caracteres de puntuación, identificadores y organizándolos todos en cadenas de caracteres denominados **lexemas**
- **Análisis sintáctico-semántico:** agrupa los lexemas en frases gramaticales. Con el resultado del análisis sintáctico se revisa la coherencia de las frases gramaticales, si su “significado” es correcto, si los tipos de dato son correctos, si los arrays tiene el tamaño y tipo adecuados, y así con todas las reglas gramaticales de nuestro lenguaje

# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

- **Generación de código intermedio:** una vez finalizado el análisis se genera una representación intermedia con el objetivo de facilitar la traducción a código objeto
- **Optimización de código** se revisa el código generado en el paso anterior optimizándolo para que el código resultante sea más fácil y rápido de interpretar por la máquina
- **Generación de código:** se genera código objeto de nuestro programa en un código de lenguaje máquina relocizable, con diversas posiciones de memoria sin establecer ya que no sabemos en qué parte de la memoria volátil se ejecutará nuestro programa.
- **Enlazador de librerías:** se enlaza nuestro código objeto con las librerías necesarias, produciendo en último termino el código final o código ejecutable.

# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

En el mercado existen aplicaciones informáticas, llamadas **Entornos Integrados de Desarrollo (EID)**, que incluyen a todos los programas necesarios para realizar todas las fases de puesta a punto de un programa; en el caso de C se necesita un editor, un preprocesador, un compilador y un enlazador.

Además, un EID suele proporcionar otras herramientas software muy útiles para los programadores, tales como: depuradores de código, ayuda en línea de uso del lenguaje, etc. Todo ello, con el fin de ayudar y facilitar el trabajo al programador.



# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

Un **depurador de código** permite al programador ejecutar un programa paso a paso, es decir, instrucción a instrucción, parando la ejecución en cada una de ellas, y visualizando en pantalla qué está pasando en la memoria del ordenador en cada momento, esto es, qué valores están tomando las variables del programa.

De esta forma, el programador puede comprobar si el hilo de ejecución del programa es el deseado. De no ser así, esto puede ser debido a diversas causas.

# PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE

Lo primero que hay que hacer es comprobar si el algoritmo se ha traducido correctamente, ya que, al programador se le puede haber pasado por alto alguna instrucción, o puede que haya puesto un signo más (+) donde debería ir un signo menos (-), etc.

En estos casos, corregir el error solamente afectará al código fuente del programa. Sin embargo, si la traducción del algoritmo es correcta, entonces el problema estará, precisamente, en el diseño de dicho algoritmo, el cual habrá que revisar, modificar y volver a codificar.

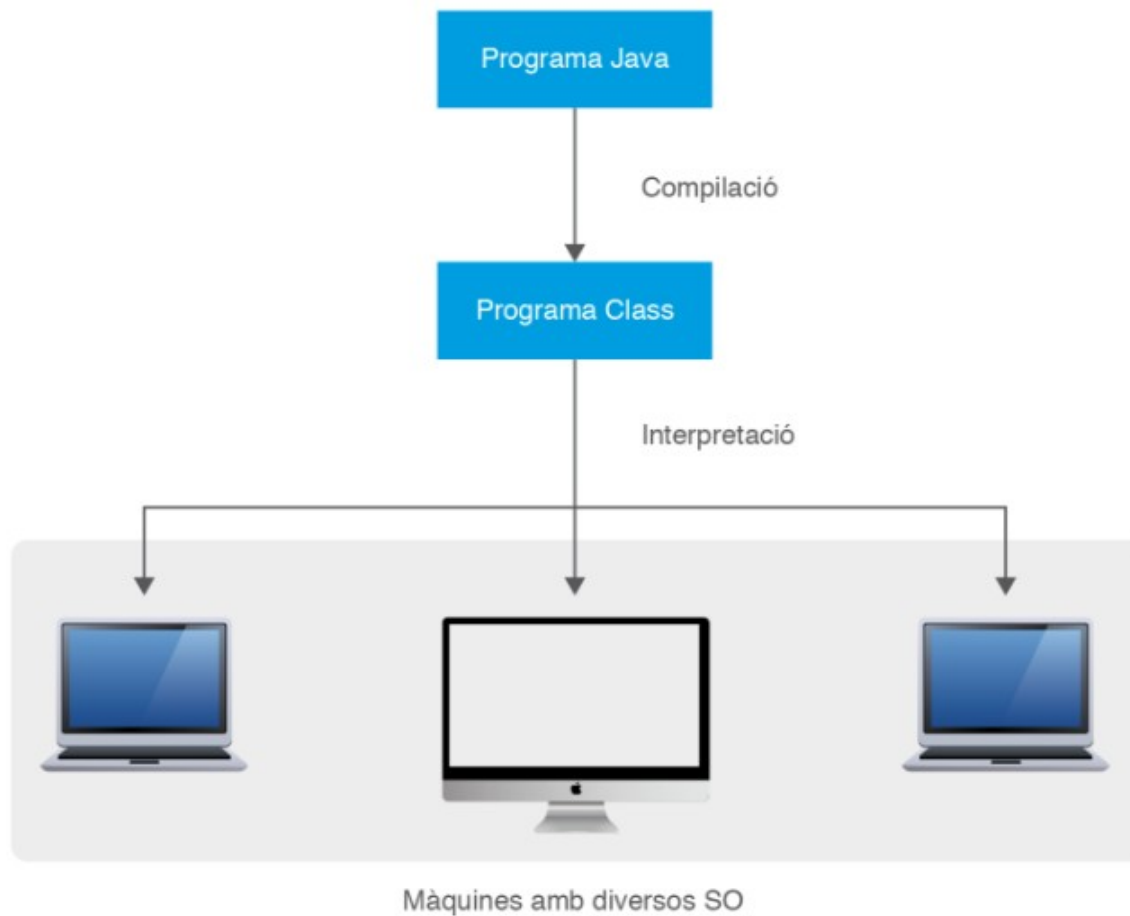
# MAQUINA VIRTUAL

Con la llegada de las máquinas virtuales la conversión de código fuente se realiza a lo que se denomina un **código Intermedio**, también llamado **bytecode**.

El objetivo es que el código intermedio, sea común para cualquier procesador, y que el código generado a partir del intermedio sea específico para cada procesador.

De hecho, la traducción del lenguaje intermedio al lenguaje máquina no se suele hacer mediante compilación sino mediante un **intérprete**.

# MAQUINA VIRTUAL



# LA MÁQUINA VIRTUAL JAVA

La **máquina virtual Java (JVM)** es el entorno en el que se ejecutan los programas Java.

Es un programa nativo, es decir, ejecutable en una plataforma específica, que es capaz de interpretar y ejecutar instrucciones expresadas en un código de bytes o (el bytecode de Java) que es generado por el compilador del lenguaje Java.

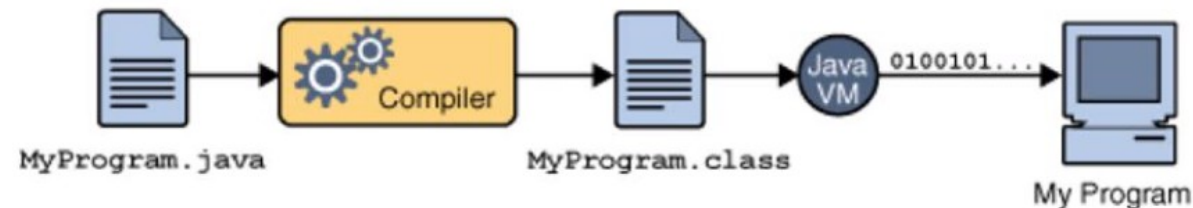
# LA MÁQUINA VIRTUAL JAVA

La gran ventaja de la JVM es que posibilita la **portabilidad de la aplicación a diferentes plataformas** y, así, un programa Java escrito en un sistema operativo Windows se puede ejecutar en otros sistemas operativos (Linux, Solaris y Apple OS X) con el único requerimiento de disponer de la JVM para el sistema correspondiente.

# MAQUINA VIRTUAL JAVA

Dado que **Java es un lenguaje interpretado**, las herramientas que se necesitan para crear y ejecutar un programa en lenguaje java son:

- Un **editor de texto** simple cualquiera que es la herramienta con la que escribimos el código.
- Un **compilador** del lenguaje Java, para generar *bytecode*.
- Un **intérprete** de Java, para poder ejecutar los programas.



# CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación se pueden **clasificar** según varios criterios. Algunos de ellos son:

- Según la **tarea a realizar**
- Según el **paradigma de programación**
- Según el **nivel de abstracción**
- Según la **manera de ejecutarse**



# CLASIFICACIÓN. MANERA DE EJECUTARSE

Según la manera de ejecutarse:

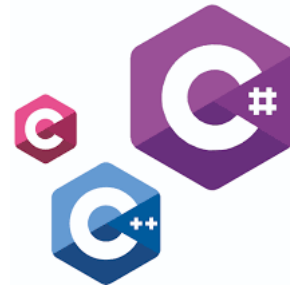
- Lenguajes **compilados**
- Lenguajes **interpretados**

# LENGUAJES COMPILADOS

Una vez escrito el **código fuente** de un programa, este se traduce por medio de un **compilador** en un archivo ejecutable para una determinada plataforma.

Ejemplos:

- Fortran
- C / C++ / C#
- Pascal / Object Pascal
- Haskell
- Go



**Go**

# LENGUAJES INTERPRETADOS

Por el contrario, un lenguaje de programación **interpretado** es aquel en el que las instrucciones se traducen o interpretan una a una.

El código es traducido por el **interprete** a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción. El proceso se repite cada vez que se ejecuta el programa el código en cuestión.

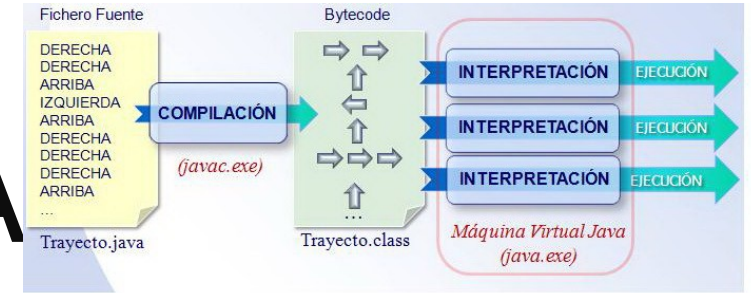
No se genera ningún código objeto ni ningún archivo ejecutable. Se trabaja directamente con el archivo de código fuente.

Ejemplos:

- JavaScript
- PHP
- Lisp
- Python 🥰



# APROXIMACIÓN MIXTA



Algunos lenguajes interpretados usan una aproximación híbrida

En este caso el compilador produce una representación intermedia del programa o **bytecode**, que después es ejecutado por un interprete.

A pesar de necesitar un proceso de compilación, estos lenguajes no se consideran realmente compilados y se **siguen clasificando como interpretados**.

Ejemplos:

- Java
- Groovy (bytecode compatible con el generado por Java)
- Kotlin (corre sobre la JVM)



# CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN

Según el nivel de abstracción, es decir, según el grado de cercanía a la máquina:

```
class Triangle {  
    ...  
    float surface()  
    return b*h/2;  
}
```

ALTO NIVEL

```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

BAJO NIVEL

```
0001001001000101  
0010010011101100  
10101101001...
```

CÓDIGO MÁQUINA

- **Lenguaje máquina:** El lenguaje máquina es el único que entiende la computadora digital, es su "lenguaje natural". En él solamente se pueden utilizar dos símbolos: el cero (0) y el uno (1). Por ello, al lenguaje máquina también se le denomina lenguaje binario. La computadora solo puede trabajar con bits, sin embargo, para el programador no resulta fácil escribir instrucciones tales como

```
10100010  
11110011  
00100010  
00010010
```

# CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN

Las instrucciones en lenguaje máquina dependen del hardware del ordenador y por lo tanto diferirán de un ordenador a otro. Código muy eficiente pero muy difícil de escribir.

Por esta razón, se inventaron lenguajes de programación más entendibles para el programador.

- Así, aparecieron los **lenguajes de bajo nivel**, también llamados **lenguajes ensambladores**, los cuales permiten al programador escribir las instrucciones de un programa usando abreviaturas del inglés, también llamadas palabras nemotécnicas, tales como: **ADD**, **DIV**, **SUB**, etc., en vez de utilizar ceros y unos. Por ejemplo, la instrucción:

**ADD a, b, c**

# CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por el ordenador sino que requiere una fase de traducción al lenguaje máquina. Los lenguajes ensambladores también son dependientes del procesador. Código muy eficiente y más fácil de escribir.

- **Lenguajes de alto nivel:** dada la complejidad de desarrollar estos programas se desarrollaron lenguajes de alto nivel independientes del procesador, diseñados para que las personas escriban y entiendan los programas de un modo más fácil que los lenguajes máquina y ensambladores, y por otro lado traducibles a código máquina.

# CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN

Un lenguaje de alto nivel permite al programador escribir las instrucciones de un programa utilizando palabras o expresiones sintácticas muy similares al inglés. Por ejemplo, en C se pueden usar palabras tales como: **case**, **if**, **for**, **while**, etc. para construir con ellas instrucciones como:

```
if ( numero > 0 ) printf( "El número es positivo", 163 );
```

Que traducido al castellano viene a decir que, si numero es mayor que cero, entonces, escribir por pantalla el mensaje: "El número es positivo".



# CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN

Otra característica importante de los lenguajes de alto nivel es que, para la mayoría de las instrucciones de estos lenguajes, se necesitarían varias instrucciones en un lenguaje ensamblador para indicar lo mismo.

De igual forma que, la mayoría de las instrucciones de un lenguaje ensamblador, también agrupa a varias instrucciones de un lenguaje máquina.

Por otra parte, un programa escrito en un lenguaje de alto nivel tampoco se libra del inconveniente que tiene el hecho de no ser comprensible para la computadora y, por tanto, para traducir las instrucciones de un programa escrito en un lenguaje de alto nivel a instrucciones de un lenguaje máquina, hay que utilizar otro programa que, en este caso, se denomina **compilador**.

# CLASIFICACIÓN. TAREA A REALIZAR

Según la manera de abordar la tarea a realizar, pueden ser:

- **Imperativos**, en los que se centra sobre **cómo** el ordenador debería hacerlo. Expresa como debe solucionarse un problema especificando una secuencia de acciones a realizar.
- **Declarativos**, en los que se enfoca en **qué** debe hacer el ordenador. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan solo se le indica al ordenador qué es lo que se desea obtener o que es lo que se está buscando y no cómo).

# PARADIGMAS DE PROGRAMACIÓN

¿ Qué entiendes por **paradigma**?

# PARADIGMAS DE PROGRAMACIÓN

Un **paradigma de programación** es un enfoque particular para la construcción de software, un estilo de programación que facilita la tarea de programación.

Algunos lenguajes soportan varios paradigmas, y otros sólo uno. Se puede decir que históricamente **han ido apareciendo para facilitar la tarea de programar según el tipo de problema a abordar.**

Entonces otra **clasificación** de los lenguajes de programación es atendiendo al paradigma de programación.

# CLASIFICACIÓN. PARADIGMA DE PROGRAMACIÓN

**Imperativo:** se expresa como debe solucionarse un problema especificando una secuencia de acciones a realizar.

- ▮ **Paradigma estructurado:** cualquier programa, por complejo y grande que sea, puede ser representado mediante tres tipos de estructuras de control: secuencia, selección, Iteración. No hay encapsulamiento de ningún tipo. Ejemplo: Basic
- ▮ **Paradigma procedural:** el código es básicamente un conjunto de procedimientos (funciones). Un procedimiento concreto es el inicial, y nuestro programa consiste en una secuencia de llamadas a procedimientos. Hay encapsulamiento de estado de procedimiento (variables locales). Ejemplos: C y Pascal
- ▮ **Paradigma orientado a objetos:** el código consiste en un conjunto de objetos que colaboran entre ellos. Cada objeto tiene una identidad, estado y comportamiento. Encapsulamiento a nivel objeto. Ejemplos: Smalltalk, Java, C#, C++, Ruby y Javascript

# CLASIFICACIÓN. PARADIGMA DE PROGRAMACIÓN

**Declarativo:** La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica al ordenador que es lo que se desea obtener o que es lo que se está buscando).

▮ **Paradigma lógico:** basados en lógica formal. Tiene como característica principal la aplicación de las reglas de la lógica para inferir conclusiones a partir de datos. Ejemplo: Prolog

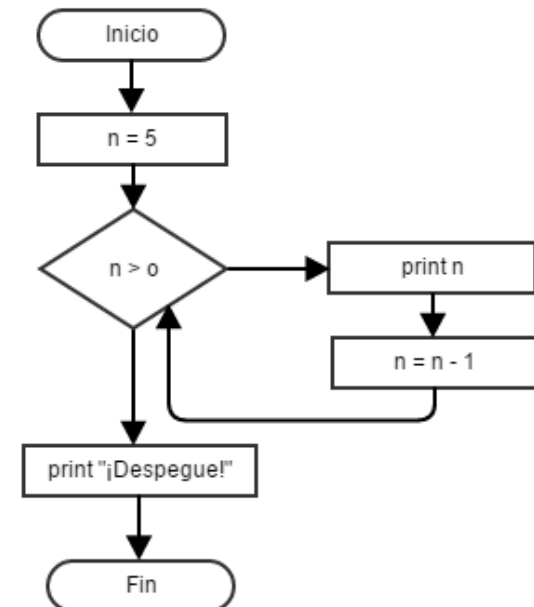
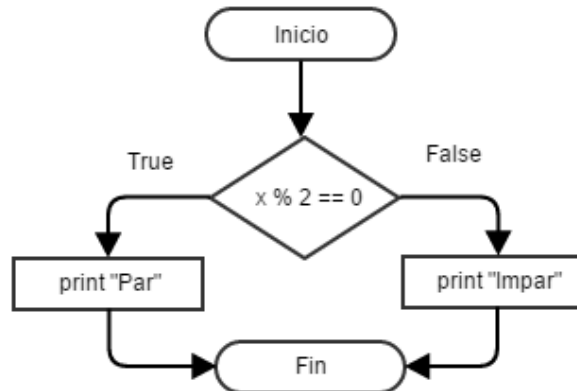
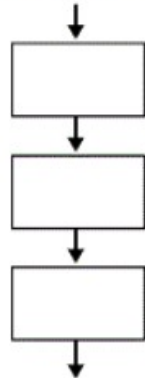
▮ **Paradigma funcional:** código basado en funciones entendiendo por función el concepto matemático de ella. La idea es que el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produzca el resultado deseado. Ejemplos: Scala, Lisp y Haskell

# EJEMPLO PROGRAMACIÓN ESTRUCTURADA

La idea es que cualquier programa, por complejo y grande que sea, puede ser representado mediante tres tipos de estructuras de control:

- Secuencia.
- Selección.
- Iteración.

Secuencia



## EJEMPLO PROGRAMACIÓN ESTRUCTURADA

Por otra parte se propone modular el programa creando porciones más pequeñas de programas con tareas específicas, que se subdividen en otros subprogramas, cada vez más pequeños.

La idea es que estos subprogramas típicamente llamados **funciones** o **procedimientos** deben resolver un único objetivo o tarea.



## EJEMPLO PROGRAMACIÓN ESTRUCTURADA

Imaginemos que tenemos que hacer una aplicación que

- registre los datos básicos del personal de una escuela, datos como pueden ser el nombre, el DNI,
- que calcule el salario de los profesores así como el de los administrativos, donde
  - el salario de los administrativos es el sueldo base (SUELDO\_BASE) \* 10
  - el salario de los profesores es el sueldo base (SUELDO\_BASE) + número de horas impartidas (numHoras) \* 12.

# EJEMPLO PROGRAMACIÓN ESTRUCTURADA

```
const float SOU_BASE = 1.000;

struct Administrativo
{
    string nombre;
    string DNI;
    float Salario;
}

struct Profesor
{
    string nombre;
    string DNI;
    int numHores;
    float salario;
}

void AssignarSalariAdministratiu (Administrativo administratiu1)
{
    administratiu1. salario = SOU_BASE * 10;
}

void AssignarSalariProfessor (Profesor profesor1)
{
    profesor1. salario = SOU_BASE + (numHores * 12);
}
```

## EJEMPLO PROGRAMACIÓN ORIENTADA A OBJETOS

La construcción de programas está basada en una abstracción del mundo real.

En un programa **orientado a objetos**, la abstracción no son procedimientos ni funciones sino los **objetos**.

Estos objetos son una representación directa de algo del mundo real, como un libro, una persona, un pedido, un empleado ...

# EJEMPLO PROGRAMACIÓN ORIENTADA A OBJETOS

Un **objeto** es una combinación de datos (llamadas **atributos**) y **métodos** (funciones y procedimientos) que nos permiten interactuar con él.

En este tipo de programación, por lo tanto, los **programas son conjuntos de objetos que interactúan entre ellos a través de mensajes** (llamadas a métodos).

# EJEMPLO PROGRAMACIÓN ORIENTADA A OBJETOS

```
class Trabajador {
    private:
        string nombre;
        string DNI;
    protected:
        static const float SOU_BASE = 1.000;
    public:
        string GetNom () {return this.nom;}
        void SetNom (string n) {this.nom = n;}
        string GetDNI () {return this.DNI;}
        void SetDNI (string dni) {this.DNI = dni;}
        virtual float salario () = 0;
}

class Administrativo: public Trabajador {
    public:
        float Salario () {return SOU_BASE * 10;}
}

class Profesor: public Trabajador {
    private:
        int numHores;
    public:
        float Salario () {return SOU_BASE + (numHores * 15);}
}
```

## EJEMPLO PROGRAMACIÓN LÓGICA

Tiene como característica principal la aplicación de las reglas de la lógica para inferir conclusiones a partir de datos.

Un programa lógico contiene una base de conocimiento sobre la que se llevan a cabo consultas.

La base de conocimiento está formada por hechos, que representan la información del sistema expresada como relaciones entre los datos y reglas lógicas que permiten deducir consecuencias a partir de combinaciones entre los hechos y, en general, otras reglas.

# EJEMPLO PROGRAMACIÓN LÓGICA

Ejemplo de desarrollo práctico del paradigma lógico

Determinaremos si debemos prescribir al paciente estar en casa reposando saber que se cumplen los siguientes hechos: malestar y  $39^{\circ}$  de temperatura corporal.

Reglas de la **base de conocimiento**:

- R1: Si fiebre, entonces estar en casa en reposo.
- R2: Si malestar, entonces ponerse termómetro.
- R3: Si termómetro marca una temperatura  $> 37^{\circ}$ , entonces fiebre.
- R4: Si diarrea, entonces dieta.

Si seguimos un razonamiento de encadenamiento hacia adelante, el procedimiento sería:

Indicar el motor de inferencia, los hechos: malestar y termómetro marca 39.

```
<Code> Base de hechos = {malestar, termómetro marca 39º}
```

El sistema identifica las reglas aplicables: R2 y R3. El algoritmo se inicia aplicando la regla R2, incorporando en la base de hechos "ponerse el termómetro".

```
Base de hechos = {malestar, termómetro marca 39º, ponerse termómetro}
```

Como no se ha solucionado el problema, continúa con la siguiente regla R3, añadiendo a la base de hechos "fiebre".

```
Base de hechos = {malestar, termómetro marca 39º, ponerse termómetro, fiebre}
```

Como no se ha solucionado el problema, vuelve a identificar un subconjunto de reglas aplicables, excepto las ya utilizadas. El sistema identifica las reglas aplicables: R1, incorporando a la base de hechos "estar en casa en reposo".

```
Base de hechos = {malestar, termómetro marca 39º, ponerse termómetro, fiebre, estar en
```



Como reposo está en la base de hechos, se ha llegado a una respuesta positiva a la pregunta formulada.



## EJEMPLO PROGRAMACIÓN LÓGICA

El paradigma es ampliamente utilizado en las aplicaciones que tienen que ver con la **Inteligencia Artificial**, particularmente en el campo de **sistemas expertos** y **procesamiento del lenguaje humano**.

Un sistema experto es un programa que imita el comportamiento de un experto humano. Por lo tanto contiene información (es decir una base de conocimientos) y una herramienta para comprender las preguntas y encontrar la respuesta correcta examinando la base de datos (un motor de inferencia).

# PARADIGMAS DE PROGRAMACIÓN

Algunos lenguajes de programación pueden soportar **múltiples paradigmas** de programación **Python** que soporta la orientación objetos y el paradigma procedural y en menor medida el funcional.



Por otro lado, algunos lenguajes han sido diseñados para soportar un **único paradigma de programación**, ese es el caso de **Smalltalk** que soporta únicamente la programación orientada a objetos

**SMALLTALK-80**

o **Haskell** que solo soporta la programación funcional



# THE TOP PROGRAMMING LANGUAGES 2019



Rank	Language	Type	Score
1	Python	🌐 🖥 ⚙	100.0
2	Java	🌐 📱 🖥	96.3
3	C	📱 🖥 ⚙	94.4
4	C++	📱 🖥 ⚙	87.5
5	R	🖥	81.5
6	JavaScript	🌐	79.4
7	C#	🌐 📱 🖥 ⚙	74.5
8	Matlab	🖥	70.6
9	Swift	📱 🖥	69.1
10	Go	🌐 🖥	68.0

IEEE Spectrum ranking

Sep 2019	Sep 2018	Change	Programming Language
1	1		Java
2	2		C
3	3		Python
4	4		C++
5	6	⬆	C#
6	5	⬇	Visual Basic .NET
7	8	⬆	JavaScript
8	9	⬆	SQL
9	7	⬇	PHP
10	10		Objective-C



TIOBE Index

# THE TOP PROGRAMMING LANGUAGES 2019

1 JavaScript

2 Java



3 Python

4 PHP

5 C++

5 C#

7 CSS

8 Ruby

9 C

10 TypeScript



Worldwide, Sept 2019 compared to a year ago:

Rank	Change	Language
1		Python
2		Java
3		Javascript
4		C#
5		PHP
6		C/C++
7		R
8		Objective-C
9		Swift
10		Matlab

[The RedMonk Programming Language Rankings: June 2019 of Programming Language](#)

[PYPL Popularity](#)

# LENGUAJES DE PROGRAMACIÓN MEJORES PAGOS

Top 8 Highly Paid Programming Languages to Learn in 2019

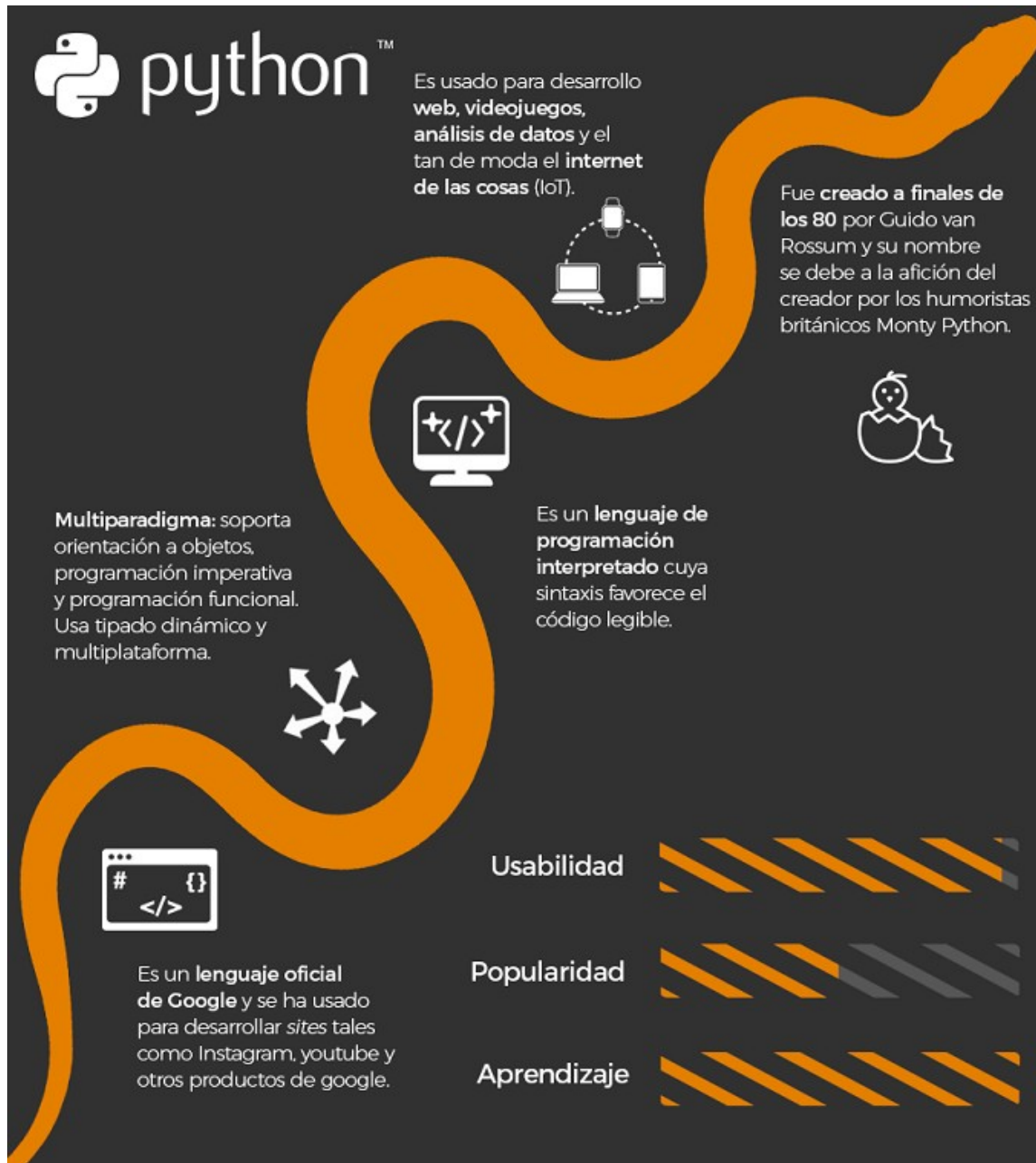
1. Go
2. Ruby
3. Objective C
4. C#
5. Python 🍷
6. Java
7. Swift
8. Kotlin

# CARACTERÍSTICAS DE LOS LENGUAJES MÁS DIFUNDIDOS

Existen muchos lenguajes de programación diferentes cada uno de estos lenguajes tiene una serie de particularidades que lo hacen diferente del resto.

Los lenguajes de programación más difundidos son aquellos que más se utilizan en cada uno de los diferentes ámbitos de la informática.

En el **ámbito educativo**, por ejemplo, se considera un lenguaje de programación muy difundido aquel que se utiliza en muchas universidades o centros educativos para la docencia de la iniciación a la programación. Por ejemplo, **python**



Python apareció en 1991.

El creador de Python, **Guido Van Rossum**, es científico en computación, trabajó para **Google** durante 7 años (2005-2012) y luego se incorporó a



```
print("Hola Mundo");
```

# EL LENGUAJE C

```
// necesario para utilizar printf()
# include <stdio.h>

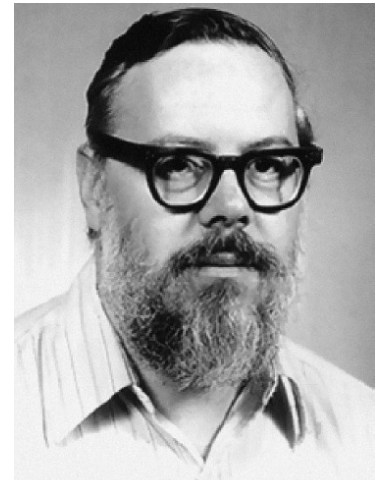
int main(void) {
    printf("Hola Mundo\n");
    return 0;
}
```

**C** es un lenguaje de programación orientado a la implementación de sistemas operativos, concretamente UNIX.

Fue desarrollado entre los años 1969 y 1972 por **Dennis Ritchie**

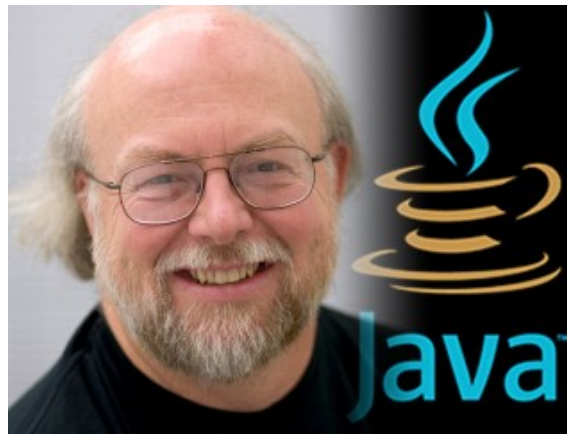
C es uno de los lenguajes más eficientes en cuanto a código resultante y es el lenguaje más popular para crear **software de sistemas**.

Es un lenguaje muy versátil, ya que trabaja tanto a bajo como a alto nivel y permite un **alto control sobre la máquina**.



*Dennis M. Ritchie*





```
/*  
    El programa Hola Mundo muestra un saludo en la pantalla  
*/  
public class HolaMundo {  
    public static void main(String[] args){  
        //Muestra "Hola Mundo!"  
        System.out.println("Hola Mundo!");  
    }  
}
```

- Es un lenguaje de programación de propósito general **orientado a objetos**
- Apareció en el año 1996 originalmente desarrollado por **James Gosling**, de Sun Microsystems (posteriormente adquirida por Oracle en 2010)
- La última versión estable es la 10.0.1 (abril de 2018)
- Su sintaxis deriva en gran medida de C y C++.
- Las aplicaciones de Java son compiladas a bytecode, que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

# SOFTWARE DE SISTEMAS Y SOFTWARE DE APLICACIÓN

Una **aplicación** es un tipo de programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo.

Esto lo diferencia principalmente de otros tipos de programas (**sistema**) que realizan tareas más avanzadas y no pertinentes al usuario común, como los sistemas operativos (que hacen funcionar al ordenador, como Windows, Mac o Linux), las utilidades (que realizan tareas de mantenimiento o de uso general), y los lenguajes de programación (con el cual se crean los programas informáticos).

# ACTIVIDAD

1. Para los lenguajes de programación mostrados en los ranking de las diapositivas anteriores busca a que paradigma corresponden

# ACTIVIDAD

Argumenta tu respuesta a las preguntas abajo formuladas con el máximo de información que ofrezcan las grandes consultorías a través de sus sitios web, así como a partir de las ofertas de trabajo publicadas específicamente para programadores en función de los lenguajes más solicitados.

2. ¿En el mundo profesional (local), consideras que se utiliza más la programación estructurada o la programación orientada a objetos?
3. ¿Cuál sería tu recomendación en el caso de que te consulten qué tipo de programación utilizar?

# ACTIVIDAD

4. Programación orientada a objetos vs programación estructurada.  
Puntos fuertes y débiles de cada paradigma.

# DESARROLLO DE SOFTWARE

Toda aplicación informática, ya sea utilizada en un ordenador de sobremesa o un ordenador portátil, o sea utilizada en dispositivos móviles como un teléfono móvil de última generación o una tableta, ha seguido un **procedimiento planificado y desarrollado detalle por detalle para su creación.**

Este irá desde la concepción de la idea o de la funcionalidad que deberá satisfacer esta aplicación hasta la generación de uno o varios ficheros que permitan su ejecución exitosa.

# DESARROLLO DE SOFTWARE: CICLO DE VIDA

Para convertir esta concepción de una idea abstracta en un producto acabado que sea eficaz y eficiente habrá muchos más pasos, muchas tareas que hacer.

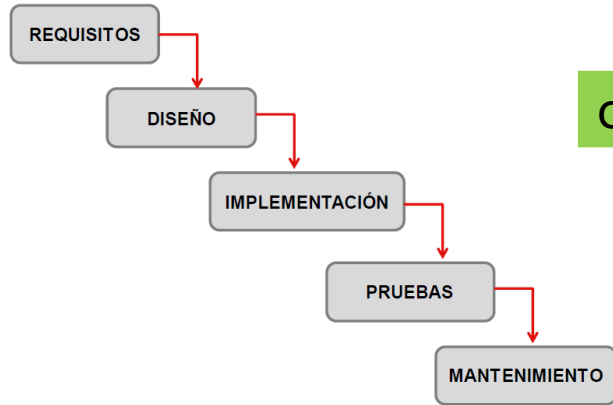
Este conjunto de etapas en el desarrollo de software corresponde al concepto de [ciclo de vida](#) del software.

No en todos los programas ni en todas las ocasiones el proceso de desarrollo llevará fielmente las mismas etapas; no obstante son unas directrices muy recomendadas.

- Análisis
- Diseño
- Codificación
- Pruebas
- Implantación
- Mantenimiento

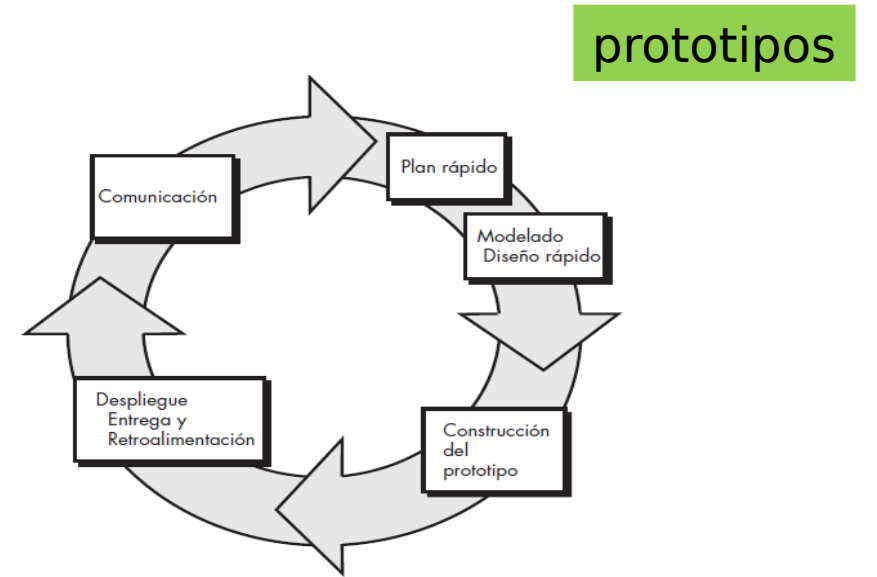
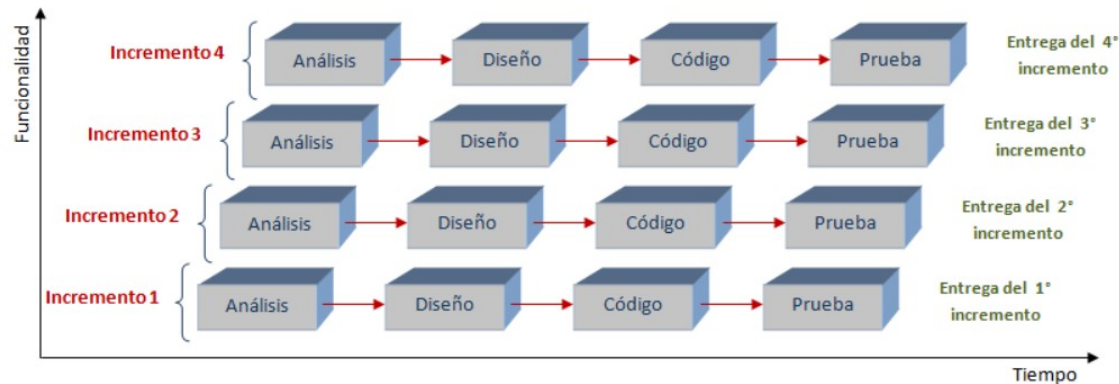
# DESARROLLO DE SOFTWARE: CICLO DE VIDA

Algunos modelos de ciclo de vida



cascada

incremental



prototipos



espiral



# DESARROLLO DE SOFTWARE: FASES

Hay **distintos modelos de ciclo de vida**. No obstante a ello las **etapas/fases** comunes a casi todos los modelos de ciclo de vida son las siguientes:

**Análisis (qué se quiere)** Define los requisitos del software que hay que desarrollar. Inicialmente esta etapa comienza con una entrevista con el cliente, que establecerá lo que quiere o lo que necesita.

Aunque el cliente crea que sabe lo que el software tiene que hacer, es necesaria una buena habilidad y experiencia para reconocer requisitos incompletos, ambiguos, contradictorios o incluso innecesarios.

Se crea un informe de requisitos del sistema (ERS) acompañado de otros diagramas que modelan los datos (diagrama entidad relación) y componentes del sistema a construir (según el paradigma serán los diagramas a construir: diagramas de clase en el paradigma OO o diagramas de flujo de datos en el paradigma estructurado)

# DESARROLLO DE SOFTWARE: FASES

**Diseño** (cómo se va a hacer) uno de los objetivos principales de esta etapa es establecer las consideraciones de recursos del sistema, tanto físicos como lógicos. Se define por tanto el entorno que requerirá el sistema. Y se crean diagramas que modelan el funcionamiento del mismo.

Se especificará también el formato de la información de entrada y salida, las estructuras de datos y la división modular (arquitectura general del software)

**Se traducen los requisitos de la etapa de análisis en una representación de software.**

# DESARROLLO DE SOFTWARE: FASES

## Codificación

El software que implemente todo lo analizado y diseñado anteriormente en un lenguaje de programación concreto, haciéndolo de una forma lo más modular posible para facilitar el posterior mantenimiento o manipulación por parte de otros programadores.

**Se traduce el análisis y diseño en una forma legible por la máquina**

# DESARROLLO DE SOFTWARE: FASES

## Pruebas

Con una doble funcionalidad las pruebas buscan confirmar que la codificación ha sido exitosa y el software no contiene errores a la vez que se comprueba que el software hace lo que debe hacer, **que no es necesariamente lo mismo.**

En general, las pruebas las realiza, idílicamente personal diferente al que realizó la codificación, con una amplia experiencia en desarrollo de software. Estas personas son capaces de saber en qué condiciones un software puede fallar de antemano sin un análisis previo.

# DESARROLLO DE SOFTWARE: FASES

**Implantación** del software en el entorno donde los usuarios finales lo utilizarán.

Cabe destacar que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de ambas aplicaciones durante un proceso de adaptación.

# DESARROLLO DE SOFTWARE: FASES

## Mantenimiento

Son muy escasas las ocasiones en las que un software no vaya a necesitar de un mantenimiento continuado. En esta fase de desarrollo de software **se arreglan los fallos o errores o se agregan nuevas funcionalidades** que suceden cuando el programa ha sido implementado.

Cuando el mantenimiento que hay que realizar consiste en una ampliación puede que sea necesario analizar los requisitos, diseñar la ampliación, codificarla, probarla, implementarla, y por supuesto, dar soporte de mantenimiento sobre la misma.

# DESARROLLO DE SOFTWARE: METODOLOGÍA

Un objetivo de décadas en el desarrollo de software ha sido encontrar **procesos y metodologías** que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del software

# DESARROLLO DE SOFTWARE: METODOLOGÍA

Una **metodología** es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo de software.

Existen numerosas propuestas metodológicas adaptándose unas mejor que otras a los distintos paradigmas.



# DESARROLLO DE SOFTWARE: METODOLOGÍA

Años atrás imperaban las metodologías **tradicionales** que se centran especialmente en el control del proceso, estableciendo las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán.

Como puede ser el caso de **Métrica v.3** que es una metodología de planificación, desarrollo y mantenimiento de sistemas de información desarrollada por el Gobierno de España que cubre tanto desarrollos **estructurados** como **orientados a objetos**.

# METRICA V3.0

Métrica v3.0 ha sido desarrollada por el Ministerio de Administraciones Públicas. Se trata de una metodología para la **planificación, desarrollo y mantenimiento** de los sistemas de información de una organización.

- Planificación de Sistemas de Información (PSI)
- Desarrollo de Sistemas de Información (DSI)
- Mantenimiento de Sistemas de Información (MSI)

# METRICA V3.0

## Desarrollo de sistemas de Información

1. Estudio de viabilidad del sistema (EVS)
2. Análisis del sistema de información (ASI)
3. Diseño del sistema de información (DSI)
4. Construcción del sistema de información (CSI)
5. Implantación y aceptación del sistema (IAS)

# DESARROLLO DE SOFTWARE: METODOLOGÍA

Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros.

A partir del 2001, surge un punto de inflexión, con el surgimiento de otro tipo de metodologías conocidas como las metodologías **ágiles** para el desarrollo de software.

# DESARROLLO DE SOFTWARE: METODOLOGÍA

Las metodologías ágiles cuales dan mayor valor a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

Este enfoque está mostrando su efectividad en proyectos pequeños/medianos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

# DESARROLLO DE SOFTWARE: ROLES

A lo largo del proceso de desarrollo de software debemos realizar, como ya hemos visto anteriormente diferentes y diversas tareas. Es por ello que el personal que interviene en el proceso es diverso. Los roles no son necesariamente rígidos y es habitual que participen en varias fases del proceso de desarrollo:

- **Jefe de proyecto:** dirige todo el proyecto de desarrollo de software. Se encarga de entregar un producto de **calidad**, manteniendo el coste dentro de las limitaciones del **presupuesto** del cliente y entregar el proyecto a **tiempo**.
- **Analista funcional:** participa en la etapa de análisis. Se encarga del análisis de requisitos del software a construir.
- **Diseñador de software:** Participa de la etapa de diseño. Diseña la solución a desarrollar a partir de los requisitos.

# DESARROLLO DE SOFTWARE:

## ROLES

- **Arquitecto:** conoce y investiga frameworks, y tecnologías, revisando que todo el proceso se lleva a cabo de la mejor forma y con los recursos más apropiados.
- **Analista programador:** comúnmente llamado desarrollador, domina una visión más amplia de la programación. Participa en la etapa de diseño y codificación
- **Programador:** escribe código fuente del software. Participa solo en la codificación del software