

# Puesta en Producción Segura

Unidad 1.

Lenguajes de programación



*"El lenguaje de programación que elijas influirá tanto en la seguridad de tu software como la cerradura que pones en tu puerta."*

—Bruce Schneier (experto en criptografía y ciberseguridad)

# Objetivos

- Comprender la clasificación de los lenguajes de programación.
- Diferenciar entre lenguajes de bajo y alto nivel, interpretados y compilados.
- Conocer los principales paradigmas de programación y sus características.
- Conocer y valorar las ventajas de utilizar un entorno de desarrollo integrado (IDE).
- Entender el proceso de ejecución de software en los diferentes tipos de lenguajes.



# Contenidos

**01** Introducción a los lenguajes de programación.

**02** Lenguajes de bajo nivel y alto nivel.

**03** Lenguajes Interpretados y compilados.

**04** Paradigmas de programación.

**05** Código fuente y entornos de desarrollo integrados (IDE).

**06** Ejecución de software.



01

# Introducción a los lenguajes de programación.



# Definición de Lenguaje de programación

- Un lenguaje de programación es un lenguaje con unas reglas bien definidas que permiten elaborar un programa.
- Un programa es una secuencia de instrucciones que permiten controlar el comportamiento físico o lógico de un sistema informático.



# Clasificación de Lenguajes: alto y bajo nivel

Tenemos varias clasificación de los lenguajes de programación:

1. Según **nivel** del lenguaje de programación:
  - Lenguajes de bajo nivel.
  - Lenguajes de alto nivel.
  
2. Según **forma de ejecución**:
  - Lenguajes interpretados.
  - Lenguaje compilados



02

# Lenguajes de bajo nivel y alto nivel.

# Lenguajes de bajo nivel: lenguaje Máquina

El **lenguaje máquina** es el único lenguaje que entiende un procesador de un ordenador.

- Se trata de una secuencia de números binarios que representan las instrucciones del programa  
P.ej.: 100011 00011 01000 00000 00001 000100
- Cada instrucción es tan larga como marca la arquitectura del procesador  
Por lo general, 32 o 64 bits



# Lenguajes de bajo nivel: lenguaje Máquina

## Características

- Muy difícil de entender para humanos, pues es muy antinatural.
- Necesita muchas instrucciones para hacer tareas aparentemente simples.
- Completamente dependiente del procesador para el que se programa.



# Lenguajes de bajo nivel: lenguaje ensamblador

- El lenguaje **ensamblador** emplea mnemotécnicas para facilitar la escritura de instrucciones.  
P.ej.: MOV AL, 61h.  
Carga en el registro AL el número 61 hexadecimal.
- Más fácil de entender que el lenguaje máquina pero:
  - Muy dependiente del procesador empleado.
  - Es difícil elaborar programas complejos con él.
- Todavía se usa para programar drivers, etc.



# Lenguajes de bajo nivel: lenguaje ensamblador

-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 13#		
OCFD:0100	20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67	48 6F 6C 61 2C
OCFD:0110	72 61 60 61 20 68 65 63-68 6F 20 65 6E 20 61 73	
OCFD:0120	73 65 60 62 6C 65 72 20-70 61 72 61 20 6C 61 20	
OCFD:0130	57 69 6B 69 70 65 64 69-61 24	
OCFD:0140		

Hola,  
este es un progra  
ma hecho en as  
semblier para la  
Wikipedia\$

Fuente: [wikipedia.org](https://wikipedia.org)

# Lenguajes de alto nivel

Los **lenguajes de alto nivel** son más cercanos a los humanos (generalmente combinan palabras en inglés con expresiones matemáticas).

## Factorial.c

```
long factorial (int n) {  
    if (n == 0) {  
        return 1;  
    } else {  
        return(n * factorial(n-1));  
    }  
}
```

# Lenguajes de alto nivel

Los **lenguajes de alto nivel** suelen ser más independientes del procesador.

- En muchas ocasiones el mismo programa podría funcionar en Windows de 32 bits, Windows de 64 bits, en Linux, en MacOS, etc.
- En otras ocasiones, se necesitan ligeras modificaciones para adaptarlo.



# Bajo nivel vs alto nivel

Una instrucción en código ensamblador suele equivaler a una instrucción en código máquina:

- Pasar de una a otra es trivial.

Las instrucciones de los lenguajes de alto nivel son muy complejas en comparación con las del ensamblador

- 20 instrucciones sencillas en lenguaje C pueden equivalen a cientos o miles en ensamblador.



# Bajo nivel vs. Alto nivel

- El ordenador sólo entiende lenguaje máquina
- Los humanos generalmente escribimos en un lenguaje de alto

Nivel, los de Bajo nivel son muy difíciles para nosotros.

- Es más rápido y sencillo elaborar programas complejos con lenguajes de alto nivel. Pero:

**Para poder ejecutar las instrucciones del programa, es necesario traducir el lenguaje de alto nivel a lenguaje máquina** de ahí la necesidad de compilar los programas o utilizar intérpretes.





03

# Lenguajes Interpretados y compilados.



# Tipos de lenguajes

Nos encontramos una clasificación que tiene que ver cómo se ejecutan las instrucciones escritas en un lenguaje de programación.

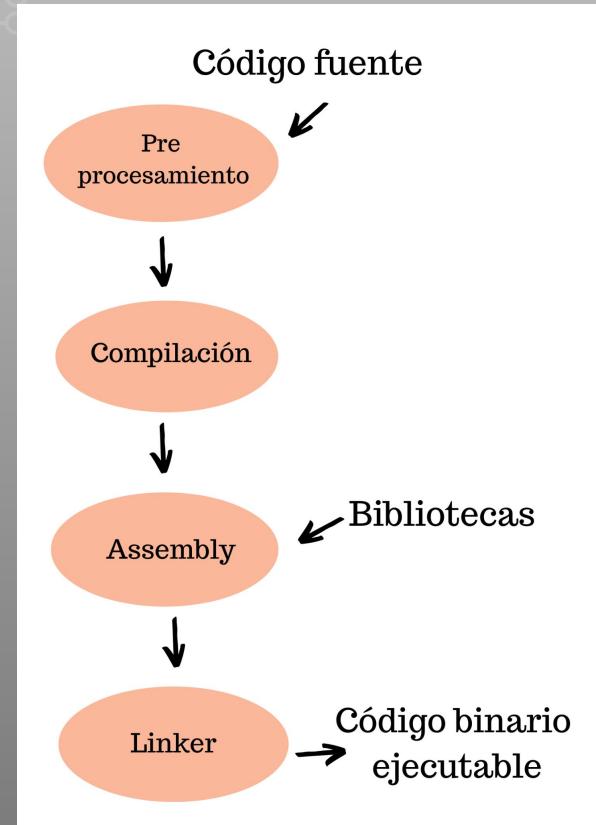
Nos encontramos con:

- **Lenguajes compilados.**
- **Lenguajes interpretados.**
- **Lenguajes híbridos**, parcialmente compilados o de Máquina Virtual.



# Lenguajes compilados

- Un **lenguaje compilado** es aquel que requiere que todas las instrucciones sean traducidas a lenguaje máquina antes de ser ejecutadas.
- A dicho proceso de traducción se le llama **compilación** y al programa que la realiza se le llama **compilador**.
- Aunque en realidad internamente se usan 3 programas uno tras otro: el **compilador**, el **ensamblador** y el **enlazador**.



Fuente: <https://www.aluracursos.com/>

## Terminal



```
$ cat holamundo.c
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
$ gcc -o holamundo holamundo.c //Genera el programa
ejecutable "holamundo"
$ ./holamundo //Ejecuta el programa "holamundo"
```

# Lenguajes interpretados

- Los **lenguajes interpretados** van traduciendo las instrucciones a lenguaje máquina una a una antes de ser ejecutadas
- Al programa que realiza este proceso se le llama **intérprete**
- Típico de los lenguajes de script de los SO
  - Bash, PowerShell, etc.
- Pero también hay otros lenguajes
  - Ruby, Python, JavaScript, etc.



# Lenguajes compilados vs lenguajes interpretados

## Lenguajes compilados

- La compilación tarda tiempo, pero la ejecución es más rápida.
- Se necesita crear un ejecutable para cada plataforma.
- Durante la compilación se previenen fallos de sintaxis y se optimiza el código.
- **Mayor carga para el programador, menor carga para la máquina.**

## Lenguajes compilados

- No requieren traducción previa, pero la ejecución es más lenta.
- El ciclo de desarrollo más rápido.
- Más fácil cometer fallos de sintaxis.
- **Menor carga para el programador, mayor carga para la máquina.**

# Lenguajes híbridos

- Los creadores de **Java** tenían como objetivo desarrollar un lenguaje de programación que funcionara en cualquier SO y arquitectura.
- La estrategia de Java se basa en compilar a un lenguaje intermedio cercano al lenguaje máquina llamado **bytecode**.
- Este lenguaje será interpretado después por la máquina virtual Java.



# Lenguajes híbridos

- Para ejecutar un programa en Java se necesita que la máquina tenga instalado el **JRE o Java Runtime Environment**.
- Para compilar un programa en Java se necesita que la máquina tenga instalado el **JDK o Java Development Kit**.
- Otros lenguajes que siguieron esta aproximación son C#, Elixir, etc.





04

# Paradigmas de programación

# Paradigmas de programación

**Paradigma de programación:** enfoque fundamental para estructurar y organizar el código de un programa informático.

- **Programación imperativa**

El programador instruye a la máquina cómo cambiar su estado.

- **Programación declarativa**

El programador simplemente declara las propiedades del resultado deseado, pero no cómo calcularlo.

- **Multiparadigma**

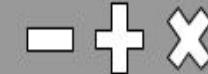


# Programación imperativa

**Programación imperativa:** El programador instruye a la máquina cómo cambiar su estado.

Tipos de programación imperativa:

- **Estructurada:** el flujo de control se define mediante bucles anidados, condicionales y subrutinas
- **Procedimental:** las instrucciones se engloban en procedimientos o funciones y éstas se llaman cada vez que haya que ejecutar la instrucciones.
- **Modular:** dividir un programa en módulos o subprogramas con el fin de hacerlo más manejable y legible.
- **Orientada a Objetos.**



## Programa estructurado, modular y procedimental

### Ejemplos

- PASCAL
- C
- BASIC
- COBOL
- FORTRAN
- Bash

```
def pedir_cantidad():
    """Pide al usuario cuántos números introducirá"""
    return int(input("¿Cuántos números quieres promediar? "))
def pedir_numero(indice):
    """Pide un número y lo devuelve"""
    return float(input(f"Introduce el número {indice}: "))
def calcular_promedio(cantidad):
    """Usa un bucle (estructurado) para sumar y calcular el promedio"""
    total = 0
    for i in range(1, cantidad + 1):
        total += pedir_numero(i)
    return total / cantidad if cantidad > 0 else 0
def mostrar_resultado(promedio):
    """Muestra el resultado"""
    print("El promedio es:", promedio)
def main():
    """Procedimiento principal"""
    cantidad = pedir_cantidad()
    promedio = calcular_promedio(cantidad)
    mostrar_resultado(promedio)
# Ejecución del programa
main()
```

# Programación orientada a objetos

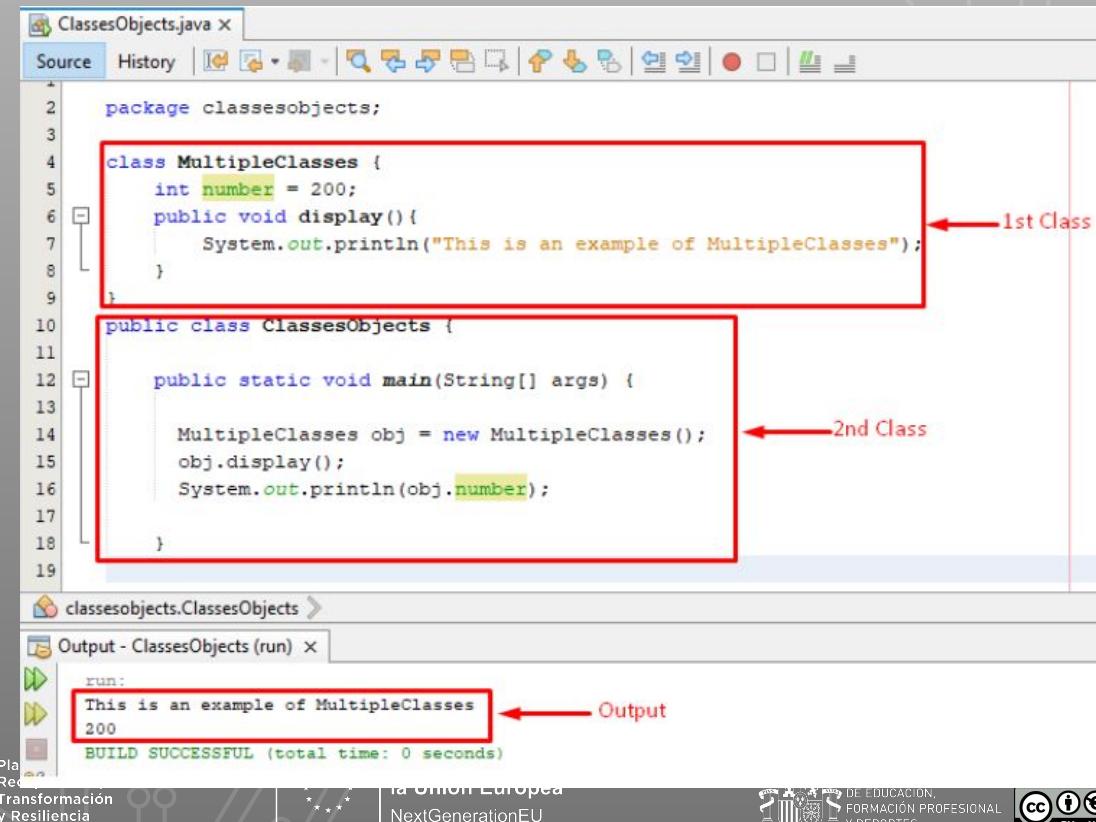
En la **programación orientada a objetos (POO)** se definen los conceptos de **clase** y **objeto**, que abstraen y encapsulan el código.

- Maximiza la cohesión y minimiza el acoplamiento.
- La **herencia** entre clases permite un mayor nivel de reutilización y extensibilidad del código.
- El **polimorfismo** permite mandar mensajes a objetos de distintas subclases como si fueran de su **superclase**.

# Programación orientada a objetos

## Ejemplos

- Visual Basic.NET
- SmallTalk
- JavaScript
- Java
- C++
- C#
- PHP
- Fortran 90/95



The screenshot shows an IDE interface with two tabs: 'Source' and 'History'. The 'Source' tab displays the following Java code:

```
1 package classesobjects;
2
3
4 class MultipleClasses {
5     int number = 200;
6     public void display() {
7         System.out.println("This is an example of MultipleClasses");
8     }
9 }
10
11 public class ClassesObjects {
12
13     public static void main(String[] args) {
14
15         MultipleClasses obj = new MultipleClasses();
16         obj.display();
17         System.out.println(obj.number);
18     }
19 }
```

Two red boxes highlight specific parts of the code:

- A red box surrounds the entire `MultipleClasses` class definition, labeled '1st Class' with a red arrow pointing to its right side.
- A red box surrounds the `main` method body and the `System.out.println` statement that prints the value of `obj.number`, labeled '2nd Class' with a red arrow pointing to its right side.

Below the code, the 'Output' tab shows the results of the program execution:

```
run:
This is an example of MultipleClasses
200
BUILD SUCCESSFUL (total time: 0 seconds)
```

A red box highlights the output text 'This is an example of MultipleClasses' and '200', labeled 'Output' with a red arrow pointing to its right side.

# Programación declarativa

**Programación declarativa:** El programador simplemente declara las propiedades del resultado deseado, pero no cómo calcularlo.

Subtipos:

- Funcional
- Lógica
- Otros (matemática, reactiva, etc.)



# Programación funcional

**Programación funcional:** El resultado deseado se declara como el valor de una serie de aplicaciones de función.

Ejemplos:

- LISP
- Erlang
- Scala
- Haskell
- Caml
- SQL

## Programación funcional SQL



```
-- Programación declarativa/funcional en SQL  
SELECT producto, AVG(venta) AS  
promedio_ventas  
FROM ventas  
GROUP BY producto  
HAVING AVG(venta) > 100;
```

# Programación lógica

**Programación lógica:** El resultado deseado se declara como la respuesta a una pregunta sobre un sistema de hechos y reglas.

Ejemplos:

- Prolog
- Mercury
- Datalog
- ECLIPSe
- Gödel

## Programa en Prolog



```
/* sintaxis: suma(X,Y,Z). */
/* semántica: X + Y = Z */
suma(0,X,X).
suma(s(X),Y,s(Z)):-suma(X,Y,Z).

/* sintaxis: producto(X,Y,Z). */
/* semántica: X * Y = Z */
producto(0,X,0).
producto(s(0),X,X).
producto(s(X),Y,G):-producto(X,Y,Z),suma(Z,Y,G).
```

?- producto (0,3,1).  
False

?- producto (4,3,12).  
True

# Multiparadigma

- Hoy en día los lenguajes han introducido elementos de diversos paradigmas, con lo cual es más difícil clasificarlos sólo en uno.
- **Java 8** introdujo los lambdas, que son programación funcional.
- **Scala** es funcional y orientado a objetos.
- **Ocaml** introdujo orientación a objetos a Cami.
- **Python** soporta programación imperativa, orientada a objetos y, en menor medida, funcional.





05

# Código fuente y entornos de desarrollo integrados (IDE)



# Código fuente

El **código fuente** es el conjunto de instrucciones escritas en un lenguaje de programación que le indica a una computadora cómo realizar una tarea específica.

Es el texto legible por humanos que los desarrolladores utilizan para crear software, aplicaciones y sitios web.

Hemos visto anteriormente los diferentes paradigmas y tipos de lenguaje de programación.



# Entornos Integrados de Desarrollo (IDE)

Un **IDE (Integrated Development Environment o entorno integrado de desarrollo)** es una herramienta que integra todo lo que hace falta para generar programas de ordenador, de forma que el trabajo sea mucho más cómodo.

Ejemplos:

- Visual Studio Code
- Netbeans
- Eclipse



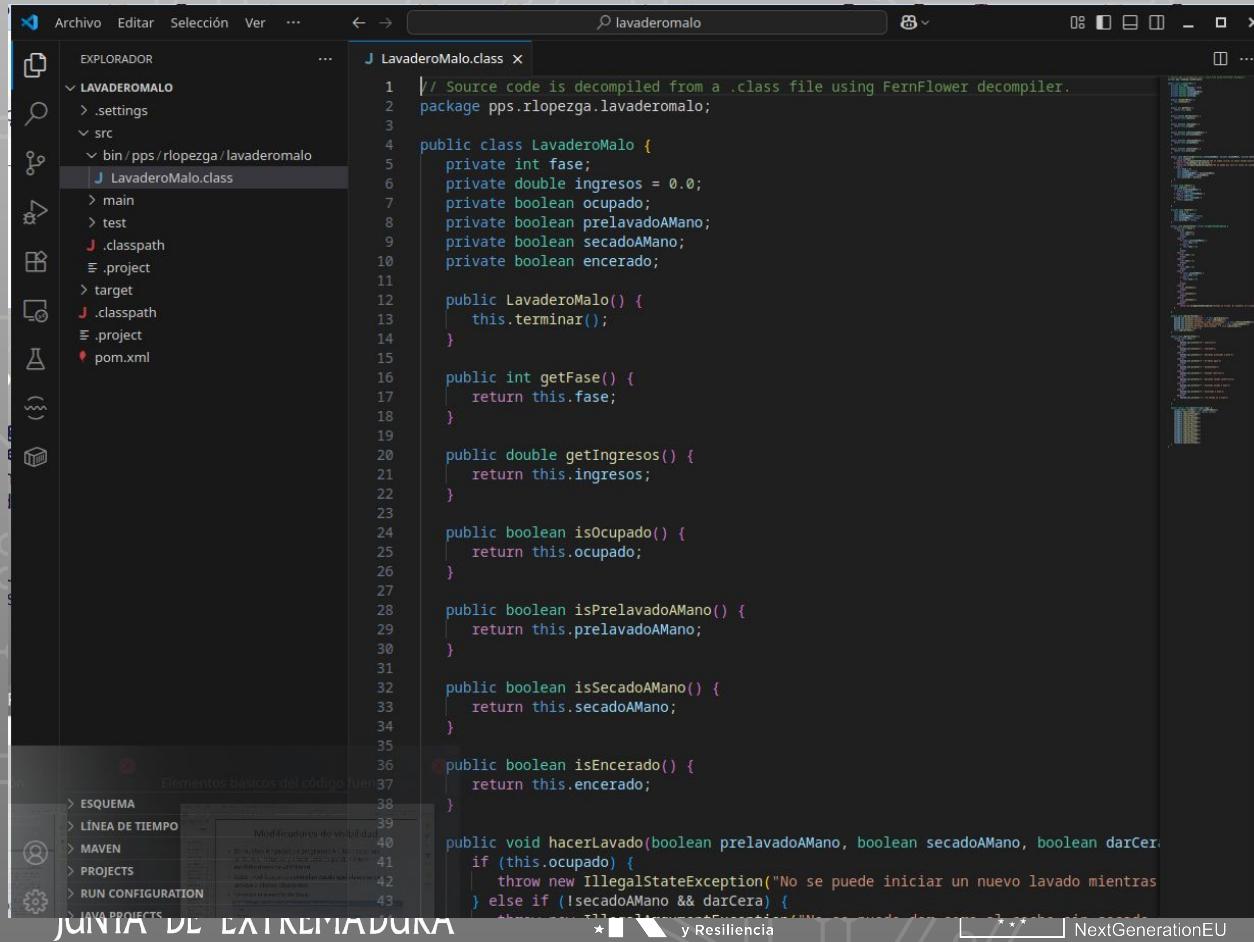
# Ventajas de los IDE

- **Agilización del flujo de trabajo:** ayuda a desarrolladores a configurar las herramientas de desarrollo web.
- **Minimización de los posibles errores:** permiten probar y depurar el código.
- **Mejora de la productividad:** facilita el desarrollo mediante las herramientas que incorporan.
- **Estandarización del proceso de desarrollo:** ayuda a la colaboración de desarrolladores.



# Componentes o características de los IDE

- **Editor de código:** escribir y editar código, resultado de sintaxis.
- **Finalización de Código:** identificar e insertar código.
- **Compilador:** traduce lenguaje alto nivel a interpretable.
- **Depurador:** ayuda a encontrar y corregir problemas.
- **Pruebas:** permite ejecutar los archivos de pruebas.
- **Herramientas de automatización de creación:** automatización de procesos de compilación, etc.
- **Compatibilidad con el lenguaje de programación:** Permite programar en diferentes lenguajes de programación.
- **Control de versiones:** permite hacer seguimiento de cambios.
- **Extensiones o plugins:** Permiten añadir funcionalidades.



The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** Archivo, Editar, Selección, Ver, ... (File, Edit, Selection, View, ...)
- Search Bar:** lavaderomalo
- Left Sidebar (Explorador):** Shows the project structure:
  - LAVADEROMALO (selected)
  - .settings
  - src
    - bin/pps/rlopezga/lavaderomalo (selected)
    - main
    - test
    - .classpath
  - .project
  - target
  - .classpath
  - .project
  - pom.xml
- Central Area:** Code editor for `LavaderoMalo.class`. The code is decompiled from a .class file using FernFlower decompiler. It contains methods for getting phase, income, and status, and for performing a wash cycle.
- Bottom Left:** Elements basicos del código fuente (Source code basic elements) and a list of Java projects.
- Bottom Right:** NextGenerationEU logo.

## Ejemplo: IDE Visual Studio Code.



# 06

## Ejecución de software



# Lenguajes compilados

- **Proceso:** Un compilador traduce el código fuente completo a código máquina antes de la ejecución. Este código máquina es específico para la arquitectura del sistema y permite una ejecución rápida y eficiente.
- **Ejemplos:** C, C++, Go (en algunos casos), Rust.
- **Ventajas:** Mayor rendimiento, código máquina optimizado.
- **Desventajas:** Proceso de compilación puede ser más lento, y la ejecución no es multiplataforma a menos que se compile para cada plataforma específica.



# Lenguajes interpretados

- **Proceso:** Un intérprete ejecuta el código fuente línea por línea, sin una traducción completa a código máquina previa. El intérprete traduce cada línea a medida que se ejecuta.
- **Ejemplos:** Python, JavaScript, Ruby.
- **Ventajas:** Facilidad para la depuración, mayor portabilidad (si el intérprete está disponible en la plataforma).
- **Desventajas:** Generalmente menor rendimiento que los lenguajes compilados, ya que la traducción ocurre en tiempo de ejecución.



# Lenguajes híbridos

- **Proceso:** Un compilador traduce el código fuente a un código intermedio (bytecode) que no es específico de una plataforma. Luego, una máquina virtual (VM) interpreta o compila este bytecode en tiempo de ejecución para la plataforma específica.
- **Ejemplos:** Java (utiliza la Máquina Virtual Java - JVM), C# (utiliza el Common Language Runtime - CLR).
- **Ventajas:** Portabilidad (la misma bytecode puede ejecutarse en diferentes plataformas con diferentes VMs), rendimiento relativamente bueno.
- **Desventajas:** Requiere la instalación de la máquina virtual, un nivel de abstracción adicional que puede afectar el rendimiento en comparación con lenguajes compilados directamente a código máquina.

# Entornos de ejecución de software

Independientemente del lenguaje, la ejecución requiere un entorno que proporcione las herramientas y recursos necesarios. Estos entornos pueden incluir:

- **Compiladores e Intérpretes**: Herramientas esenciales para la traducción y ejecución del código.
- **Máquinas virtuales** (JVM, CLR): Para lenguajes híbridos, que actúan como intermediarios entre la aplicación y el sistema operativo.
- **Navegadores web**: Para JavaScript, que se incrusta en documentos HTML.
- **Sistemas operativos** (como Node.js): Para JavaScript en el lado del servidor, o para otros lenguajes que se ejecutan en un entorno de sistema.
- **Librerías**: Conjuntos de código pre-escrito que facilitan la implementación de funcionalidades comunes.

# Ejecución de software

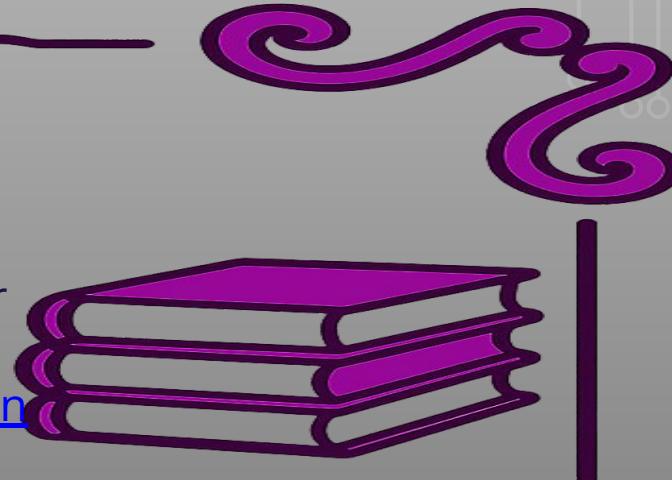
La ejecución de software en diferentes lenguajes de programación implica procesos diversos, principalmente diferenciados por la forma en que el código fuente se traduce y se ejecuta.

- Lenguajes compilados.
- Lenguajes interpretados.
- Lenguajes híbridos o de Máquina Virtual.



# Bibliografía y Webgrafía

- **Introducción a los lenguajes de programación.** Rafael López García.
- **Iniciación a la programación.** Victor Ponz.
- ¿Qué es un entorno de desarrollo y en qué se diferencia de un entorno de desarrollo integrado (IDE)?



# Gracias!

¿Alguna pregunta?



[informatica.iesvalledeljerteplasencia.es](http://informatica.iesvalledeljerteplasencia.es)



[coordinacion.cenfp@iesvp.es](mailto:coordinacion.cenfp@iesvp.es)



C/ Pedro y Francisco González, s/n  
10600, Plasencia (Cáceres)



927 01 77 74

