

Lenguajes de programación y paradigmas

Entornos de desarrollo

Diapositivas realizadas por Aitor Ventura Edo

Cr terios de clasificaci n de lenguajes

Idea








Idea clave

Los lenguajes de programación no son todos iguales. Podemos **clasificarlos** según distintos criterios, lo que nos ayuda a entender **cómo se usan, qué tan fáciles son y para qué sirven**.






Según el nivel de abstracción

⚙️ Lenguajes de bajo nivel



-  Cercanos al lenguaje máquina.
-  Ejemplo: **Ensamblador**.
-  Pros: control total del hardware.
-  Contras: muy difíciles de aprender y mantener.
-  Uso típico: sistemas embebidos, drivers, firmware.

Según el nivel de abstracción

Lenguajes de alto nivel

-  Cercanos al lenguaje humano.
-  Ejemplos: **Python, Java, C#**.
-  Pros: fáciles de aprender, portables.
-  Contras: menos control directo sobre el hardware.
-  Uso típico: apps web, móviles, software de usuario.

Según el propósito

Tipo	Características	Ejemplos	Uso típico
 Generales	Sirven para casi cualquier aplicación	Java, Python, C++	Desarrollo de software en general
 De dominio específico (DSL)	Creado para tareas concretas	SQL, HTML/CSS, MATLAB	Bases de datos, web, cálculo científico

Según el dominio de aplicación

Sistemas

- Lenguajes pensados para interactuar con hardware y SO.
- Ejemplos: **C, C++**.

Aplicaciones web

- Enfocados a Internet y servidores.
- Ejemplos: **JavaScript, PHP**.

Según el dominio de aplicación

Datos / IA

- Análisis, big data, machine learning.
- Ejemplos: **Python, R, Julia.**

Móviles

- Para apps en Android/iOS.
- Ejemplos: **Kotlin, Swift.**

Según el dominio de aplicación

Embebidos

- Para dispositivos con pocos recursos.
- Ejemplos: **C, Rust.**

Típado de lenguajes



¿Qué es el tipado?

El **tipo de dato** (número, texto, booleano, etc.) define qué operaciones se pueden hacer.

Según el lenguaje, los **tipos se controlan de formas distintas**, lo que afecta a la seguridad, flexibilidad y facilidad de programación.



Según el momento en que se comprueban

✓ Estático

- El compilador revisa los tipos **antes de ejecutar**.
- Ejemplos: **Java, C, Rust**.
-  Ventaja: detecta muchos errores en la fase de compilación → más seguro.
-  Inconveniente: más estricto, hay que declarar todo con detalle.

Según el momento en que se comprueban

► Dinámico

- Los tipos se verifican **mientras se ejecuta**.
- Ejemplos: **Python, JavaScript, Ruby**.
-  Ventaja: más flexible y rápido de escribir (menos código).
-  Inconveniente: errores de tipos pueden aparecer en **tiempo de ejecución**.

Según la rigidez en el uso de tipos

Tipo	Descripción	Ejemplo	Resultado
 Fuerte	No permite mezclar tipos sin conversión explícita.	Python: <code>"3"</code> <code>+ 2</code>	 Error
 Débil	Hace conversiones automáticas (a veces inesperadas).	JavaScript: <code>"3"</code> + 2	<code>"32"</code>

Según la rigidez en el uso de tipos



Atención



Un lenguaje de tipado **débil** puede producir resultados extraños sin avisar, lo que genera **bugs difíciles de detectar**.

Inferencia de tipos

Algunos lenguajes **deducen automáticamente** el tipo de una variable según el valor que se le asigna.

Kotlin

```
1 // Kotlin
2 val x = 5 // El compilador entiende que x es un Int
3 val nombre = "Ana" // El compilador entiende que es un String
```



-  Ventaja: ahorra escritura y mantiene seguridad de tipos.
-  Inconveniente: a veces el tipo deducido no es el esperado si no se tiene cuidado.

Inferencia de tipos

Algunos lenguajes **deducen automáticamente** el tipo de una variable según el valor que se le asigna.

Kotlin

```
1 // Kotlin
2 val x = 5 // El compilador entiende que x es un Int
3 val nombre = "Ana" // El compilador entiende que es un String
```

-  Ventaja: ahorra escritura y mantiene seguridad de tipos.
-  Inconveniente: a veces el tipo deducido no es el esperado si no se tiene cuidado.

Paradigmas de programación

Idea

Idea clave

Un **paradigma** es un **estilo de programación**, es decir, la manera en que se organiza y estructura el código.

 Un mismo lenguaje puede soportar **varios paradigmas**.

Típos de paradigmas

{ } Imperativo / Procedimental




- 📄 El programa es una **secuencia de instrucciones** paso a paso.
- 🛠️ Uso típico: algoritmos simples, sistemas básicos.
- 💻 Ejemplo: **C**.

📦 Orientado a Objetos (POO)




- 📦 Código organizado en **clases y objetos**.
- 🔄 Cada objeto tiene **atributos (datos)** y **métodos (acciones)**.
- 💻 Ejemplo: **Java, C++**.

Típos de paradigmas

f_x Funcional




-  Se centra en **funciones puras** (sin efectos secundarios).
-  Evita modificar datos directamente.
-  Ejemplo: **Haskell, Scala, Elixir.**

Lógico




-  Basado en **reglas y hechos.**
-  El programa **deduce soluciones** automáticamente.
-  Ejemplo: **Prolog.**

Típos de paradigmas

Orientado a eventos

-  El flujo depende de **eventos externos** (clics, señales...).
-  Muy común en interfaces gráficas y la web.
-  Ejemplo: **JavaScript en navegadores**.

Reactivo

-  Pensado para **datos en tiempo real y asincronía**.
-  Se adapta a flujos de información continuos.
-  Ejemplo: **RxJava, frameworks modernos (Angular, React)**.

Ecosistemas y estándares

Los lenguajes suelen estar regulados por **organismos y estándares**:

- **ISO C/C++** → definen cómo deben funcionar los compiladores.
- **ECMA** → regula JavaScript (ECMAScript).
- **PEPs (Python Enhancement Proposals)** → propuestas de mejora en Python.
- **JSRs (Java Specification Requests)** → definen cómo evoluciona Java.

Elección del lenguaje según escenario

Escenario	Lenguajes típicos	Razón
Sistemas operativos / drivers	C, C++	Control cercano al hardware
Web (frontend)	JavaScript, TypeScript	Se ejecuta en navegadores
Web (backend)	Python, Java, Node.js	Gran ecosistema y frameworks

Escenario	Lenguajes típicos	Razón
Móviles	Kotlin (Android), Swift (iOS)	Integración nativa
Datos / IA	Python, R	Librerías para análisis y ML
Embebidos / IoT	C, Rust	Bajo consumo, eficiencia

Casos comparativos breves



Lenguaje C

- 🏷️ **Nivel:** medio-bajo.
- 🛡️ **Tipado:** estático y fuerte.
- 🧩 **Paradigma:** procedimental.
- ⚙️ **Usos típicos:** sistemas operativos, drivers, software embebido.



Java

- 🏷️ **Nivel:** alto.
- 🛡️ **Tipado:** estático y fuerte.
- 🧩 **Paradigma:** orientado a objetos.
- 🏢 **Usos típicos:** aplicaciones empresariales, Android, sistemas distribuidos.



Python

- 🏷️ **Nivel:** alto.
- 🛡️ **Tipado:** dinámico y fuerte.
- 🧩 **Paradigma:** multiparadigma (imperativo, POO, funcional).
- 🤖 **Usos típicos:** ciencia de datos, IA, scripting, desarrollo web.



JavaScript

- 🏷️ **Nivel:** alto.
- 🛡️ **Tipado:** dinámico y débil.
- 🧩 **Paradigma:** orientado a eventos, funcional.
- 🌐 **Usos típicos:** frontend web, backend (Node.js), apps híbridas.



Rust

- 🏷️ **Nivel:** medio-alto.
- 🛡️ **Tipado:** estático y fuerte.
- 🧩 **Paradigma:** multiparadigma.
- ⚡ **Usos típicos:** sistemas de alto rendimiento, programación segura, embebidos.



Kotlin

- 🏷️ **Nivel:** alto.
- 🛡️ **Tipado:** estático, con inferencia.
- 🧩 **Paradigma:** orientado a objetos + funcional.
- 📱 **Usos típicos:** Android, aplicaciones multiplataforma, backend.