



Técnicas de Programación (JavaScript)

Técnicas de Programación - JavaScript





Introducción a la Programación



¿Qué es la programación?

La programación es el acto de programar, es decir, organizar una serie de pasos ordenados a seguir para realizar un proyecto y cumplir con un objetivo.

Este término puede utilizarse en muchos contextos, por ejemplo, se puede programar una fiesta, unas vacaciones, un evento, una lista de programas televisivos con días y horarios, etc.



¿Qué es la programación en informática?

En el ámbito de la informática, la programación es la acción de crear programas o aplicaciones a través del desarrollo de un código fuente que se basa en un conjunto de instrucciones que sigue el ordenador para ejecutar un programa.

A este conjunto de reglas o instrucciones se las conoce como algoritmos.

En sí, la programación informática nos permite resolver problemas de la vida real utilizando un lenguaje para comunicarse con el ordenador.



Importancia:

- Permite automatizar tareas y resolver problemas.
- Es la base para el desarrollo de aplicaciones, sitios web, videojuegos y sistemas complejos.
- Fomenta el pensamiento lógico y la capacidad de resolver problemas de manera sistemática.

Aspectos clave:

- Requiere de planificación (análisis y diseño de algoritmos).
- Involucra tanto la parte teórica (conceptos y lógica) como la práctica (escribir y probar código).



¿Cómo se programa?

Proceso de programación:

1. Análisis del problema: Comprender el problema a resolver.
2. Diseño del algoritmo: Planificar la secuencia lógica de pasos.
3. Codificación: Escribir el código en un lenguaje de programación.
4. Pruebas y depuración: Ejecutar el código, identificar errores y corregirlos.
5. Mantenimiento: Actualizar y mejorar el código con el tiempo.



¿Qué es un algoritmo?

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un determinado problema.

En programación, un algoritmo supone el paso previo a ponerse a escribir el código. Primero debemos encontrar la forma de obtener la solución al problema (definir el algoritmo informático), para luego, a través del código, poder indicarle a la máquina qué acciones queremos que lleve a cabo. De este modo, un programa informático no sería más que un conjunto de algoritmos ordenados y codificados en un lenguaje de programación para poder ser ejecutados en un ordenador.



¿Qué es un pseudocódigo?

Un programa normalmente implementa y contiene uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un diagrama de flujo, en forma de código como en pseudocódigo o un lenguaje de programación, en forma explicativa.

Un pseudocódigo es una descripción de alto nivel de un algoritmo que emplea una mezcla de lenguaje natural con algunas convenciones estructurales propias de los lenguajes de programación. No hay un standard.

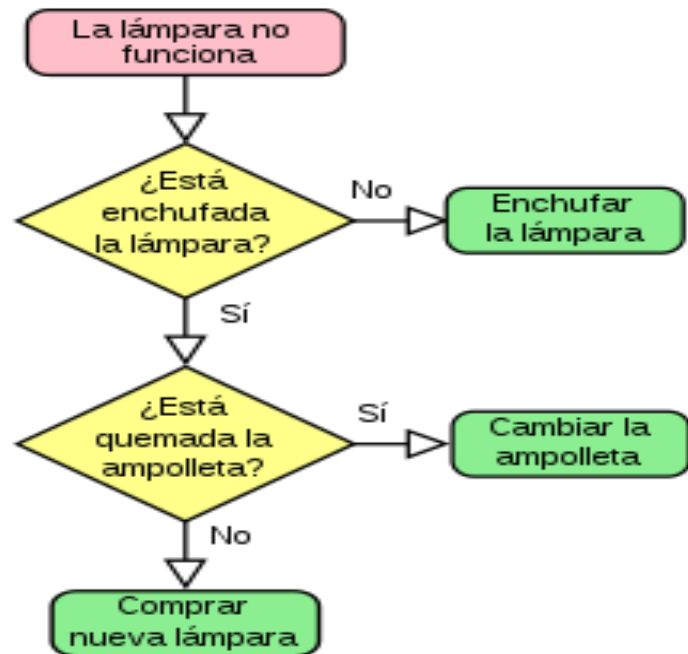


¿Qué es un diagrama de flujo?

Un diagrama de flujo es un diagrama que ilustra un flujo de trabajo, proceso o sistema.

Los diagramas de flujo describen el orden de los pasos o tareas involucradas, a menudo utilizando una línea o flecha para señalar la dirección de la información.

Son, en definitiva, la representación gráfica de un algoritmo.





¿Para qué sirve un diagrama de flujo?






Los diagramas de flujo pueden mejorar la toma de decisiones, permitir la visualización de procesos, ayudar a organizar el flujo de información, documentar y analizar procesos y sistemas.

También pueden señalar posibles problemas o cuellos de botella al desglosar procedimientos complejos y descubrir hasta el último detalle que los compone.

Estos gráficos son una manera muy buena de representar una lógica paso a paso para que cualquiera lo entienda. Si se hace un buen diagrama de flujo de una acción que quiera ser programada, programarla puede llegar a resultar mucho mas fácil.



Símbolos del diagrama de flujo

	Indica el inicio o fin de un proceso
	Indica cada actividad que necesita ser ejecutada
	Indica un punto de toma de decisión
	Indica la dirección de flujo
	Indica los documentos utilizados en el proceso



Lenguajes de programación

Un lenguaje de programación es un lenguaje formal que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina.



JavaScript





Clasificación de los lenguajes

Por nivel:

Lenguajes de bajo nivel: Muy cercanos al hardware (por ejemplo, ensamblador).

Lenguajes de alto nivel: Más abstractos y cercanos al lenguaje humano (por ejemplo, Python, JavaScript).

Por paradigma:

Imperativos: Se basan en una secuencia de instrucciones (C, Java).

Funcionales: Se centran en la evaluación de funciones (Haskell, Elixir).

Orientados a objetos: Se estructuran en torno a objetos y clases (Java, Python, JavaScript).

Lógicos: Basados en reglas y lógica formal (Prolog).



Lenguajes de bajo nivel

Son lenguajes muy cercanos al hardware. Trabajan casi directamente con los recursos físicos de la computadora (memoria, registros del procesador, etc.). Esto implica que el programador tiene mucho control sobre lo que sucede a nivel de máquina.

Ejemplos típicos

- **Código máquina (binario):** Secuencia de bits que el procesador interpreta directamente. Consiste en la combinación de 0's y 1's (ceros y unos) para formar las ordenes entendibles por el hardware de la máquina.
- **Ensamblador (Assembly):** Usa mnemónicos (por ejemplo, MOV, ADD) que representan instrucciones del procesador.



Lenguajes de bajo nivel

Ventajas

- Control total sobre el hardware (ideal para sistemas embebidos o muy optimizados).
- Posibilidad de optimizar el rendimiento al máximo.

Desventajas

- Muy complejos de leer y escribir.
- Poco portables: el código depende fuertemente de la arquitectura de la CPU.
- Toma más tiempo desarrollar y mantener programas.



Lenguajes de medio nivel

A veces se habla de una clasificación de medio nivel, son lenguajes que permiten una mayor abstracción, pero manteniendo algunas características del lenguaje de bajo nivel.

El código es enviado a un compilador que lo convierte al lenguaje máquina.

Por ejemplo: Lenguaje **C**.

C puede acceder a registros del sistema y direcciones de memoria, todas propias de lenguajes de bajo nivel.

Debido a sus características, estos lenguajes se pueden situar entre los de bajo nivel y alto nivel.



Lenguajes de alto nivel

Son lenguajes más cercanos al lenguaje humano que al hardware, y que abstraen gran parte de los detalles de bajo nivel (como la gestión de memoria). El programador se centra más en la lógica del programa y menos en cómo interactuar con el hardware.

Ejemplos típicos

- **Python, Java, JavaScript, PHP, Ruby, Kotlin, C#:** Manejan la memoria y los recursos de forma automática o semiautomática.
- **Swift:** También abstrae gran parte de la complejidad del hardware.



Lenguajes de alto nivel

Ventajas

- Más fáciles de aprender y de mantener.
- Mayor productividad: se puede desarrollar funcionalidad compleja en menos tiempo.
- Suele haber librerías y frameworks que facilitan tareas comunes (redes, interfaces gráficas, acceso a bases de datos, etc.).

Desventajas

- Menor control directo sobre el hardware.
- En general, pueden ser menos eficientes en rendimiento que un programa escrito en un lenguaje de bajo nivel (aunque esto depende mucho del caso de uso y de las optimizaciones del compilador/intérprete).



Lenguajes interpretados

En los lenguajes interpretados, el código fuente se lee y se ejecuta instrucción por instrucción en tiempo real a través de un intérprete.

El intérprete traduce cada línea de código a lenguaje máquina (o a un formato intermedio) y la ejecuta de inmediato.

JavaScript





Lenguajes interpretados

Ventajas

1. Desarrollo rápido y flexible:

- No necesitas compilar el programa completo antes de ejecutarlo.
- Facilita la experimentación y las pruebas rápidas.

2. Portabilidad:

- El mismo código fuente puede ejecutarse en distintas plataformas, siempre y cuando exista un intérprete compatible.

3. Depuración sencilla:

- Es más fácil detectar y corregir errores, ya que se identifican en tiempo de ejecución línea a línea.



Lenguajes interpretados

Desventajas

1. Rendimiento menor:

- Al traducir y ejecutar cada instrucción en tiempo real, suelen ser más lentos que los lenguajes compilados (aunque existen optimizaciones).

2. Dependencia del intérprete:

- Es necesario instalar el intérprete en cada máquina donde se quiera ejecutar el programa.



Lenguajes compilados

En los lenguajes compilados, un compilador traduce todo el código fuente a código máquina (o a un bytecode intermedio) antes de ejecutarse.

Este proceso genera un archivo ejecutable que puede ser lanzado directamente por el sistema operativo (o por una máquina virtual, en el caso de bytecode).



Swift



Lenguajes compilados

Ventajas

1. Mayor rendimiento:

- Al ejecutarse directamente como código máquina (o en un entorno cercano al hardware), suelen ser más rápidos.

2. Optimizaciones del compilador:

- El compilador puede analizar y optimizar globalmente el código antes de generar el ejecutable.

3. No requiere un intérprete en tiempo de ejecución:

- Una vez compilado, el programa puede distribuirse y ejecutarse en sistemas compatibles sin requerir el compilador o el código fuente.



Lenguajes compilados

Desventajas

1. Menor flexibilidad en el desarrollo:

- Cada vez que modificas el código, debes recompilarlo para probar los cambios.

2. Dependencia de la plataforma:

- Un ejecutable generado para una arquitectura/OS específico no funcionará en otra sin recompilar.



Consideraciones Intermedias: Bytecode y JIT

Algunos lenguajes como Java y C# se compilan primero a un bytecode intermedio y luego se interpretan o compilan en tiempo de ejecución (JIT) en la máquina virtual correspondiente (JVM, CLR).

Esto mezcla características de ambos mundos:

- **Ventajas de la compilación** (optimizaciones)
- **Flexibilidad de la interpretación** (portabilidad, ejecución multi-plataforma)





Introducción a JavaScript



JavaScript

JavaScript es un lenguaje de programación interpretado, dinámico, de alto nivel y multiparadigma.

Es esencial para desarrollar contenido dinámico y altamente interactivo en sitios web.

Se ejecuta en el navegador del usuario, permitiendo modificar el contenido, responder a eventos, realizar cálculos y comunicarse con servidores, entre otras cosas.

El código se ejecuta línea a línea sin un proceso de compilación previo, lo que permite ver resultados inmediatos y facilita la depuración.

Los tipos de datos se determinan en tiempo de ejecución y el código puede modificarse de forma flexible, lo que lo hace muy versátil.

Su uso se extiende también al desarrollo del lado del servidor mediante tecnologías como Node.js.



JavaScript

Características:

- Popularidad y Comunidad: Amplia base de usuarios y recursos disponibles.
- Interactividad inmediata: Se ejecuta directamente en el navegador, ofreciendo resultados al instante.
- Curva de aprendizaje accesible: La sintaxis es relativamente sencilla, lo que facilita entender conceptos básicos de programación.
- Versatilidad: Permite el desarrollo tanto del lado del cliente (navegador) como del lado del servidor (Node.js).
- Facilidad de integración: Se integra naturalmente con HTML y CSS, lo que facilita el desarrollo de interfaces web interactivas.

Técnicas de Programación - JavaScript



JavaScript

Características:

- Actualizaciones constantes: La evolución continua del lenguaje y su estandarización permiten incorporar nuevas funcionalidades y mejorar la eficiencia y seguridad del código.
- Ecosistema rico:
 - Herramientas de desarrollo avanzadas y entornos de ejecución modernos.
 - Amplio soporte en la mayoría de los navegadores y plataformas.
 - Numerosas librerías y frameworks que amplían sus capacidades.



JavaScript

Aplicaciones principales:

- **Front-end:** Manipulación del DOM para actualizar el contenido de la página sin recargarla. Creación de animaciones, validaciones de formularios y efectos visuales.
- **Back-end:** Con Node.js, se utiliza para crear servidores, APIs y aplicaciones web escalables.
- **Aplicaciones híbridas y móviles:** Frameworks como React Native permiten desarrollar aplicaciones móviles usando JavaScript.
- **Internet de las Cosas (IoT):** Algunos dispositivos y microcontroladores utilizan JavaScript para ejecutar funciones simples.



JavaScript

Origen:

En el año 1994 aparece Netscape, el primer navegador web comercial masivo y gratuito de la historia.

Para 1995, Netscape se da cuenta de que las páginas web eran bastante básicas, eran solamente para consumir. No había interacción. El usuario simplemente consumía.

Así que decidieron crear un lenguaje de programación que funcionara en el navegador. Un lenguaje básico de scripting. Es decir, un lenguaje de comandos específicos para hacer algunas cosas básicas.

En este contexto, en 1995 Brendan Eich crea JavaScript para Netscape.



JavaScript

Origen del nombre:

Aunque el nombre sugiere una relación con Java, se eligió estratégicamente para aprovechar la popularidad de Java en ese momento.

Sin embargo, son lenguajes distintos con filosofías y usos diferentes.

En sus orígenes se lo conoció como LiveScript, porque eran scripts vivos en el navegador. Anteriormente había sido bautizado con el nombre de Mocha.



JavaScript

Evolución y estandarización:

ECMAScript:

El lenguaje se estandarizó bajo el nombre ECMAScript. Desde ES3 hasta las versiones más modernas como ES6 (ES2015) y posteriores, se han introducido mejoras significativas:

- Funciones flecha, clases y módulos.
- Asincronía simplificada con promesas y async/await.
- Mejoras en la sintaxis y seguridad del código.

Adopción global:

JavaScript se ha convertido en el lenguaje dominante para la web, evolucionando constantemente para adaptarse a nuevas tecnologías y paradigmas de desarrollo.



JavaScript

Sintaxis básica:

El código JavaScript puede insertarse en archivos HTML mediante la etiqueta `<script>` o en archivos externos con extensión `.js`.

Un script de JavaScript está compuesto por instrucciones que se ejecutan de manera secuencial (de arriba a abajo).

Es fundamental respetar la sintaxis (uso correcto de llaves, punto y coma, paréntesis, etc.) para evitar errores de ejecución.

El punto y coma (;) se utiliza para marcar el final de una instrucción. No es obligatorio su uso, pero por convención y buenas prácticas, se pueden utilizar normalmente.



JavaScript

Sintaxis básica:

JavaScript tiene un mecanismo llamado Inserción Automática de Punto y Coma (ASI), lo que significa que, en muchos casos, el motor del lenguaje inserta los puntos y coma de forma implícita cuando no se encuentran en el código.

Cuando el intérprete encuentra un salto de línea o un final de instrucción, puede asumir que se terminó la instrucción y, si es necesario, insertar un punto y coma.

Hay que tener en cuenta que la ASI no siempre funciona como esperamos.



JavaScript

Variables:

Una variable es un contendor que almacena un valor.

Las variables son esenciales en programación porque permiten almacenar información (como números, textos o estados) y manipularla a lo largo del programa.

Esto hace posible que el código responda a diferentes condiciones y realice cálculos o cambios en función de la información que se le suministre.

En JavaScript, los nombres de variables se conocen como identificadores.



JavaScript

Definiciones de variables:

- **Caracteres Permitidos:** Pueden contener letras (A-Z, a-z), dígitos (0-9), el carácter de subrayado (_) y el signo de dólar (\$).
- **Regla de Inicio:** Un identificador no puede comenzar con un dígito. Debe empezar con una letra, un subrayado o un signo de dólar.
- **Sensibilidad a Mayúsculas y Minúsculas:** Los identificadores son case-sensitive, lo que significa que Variable, variable y VARIABLE se consideran nombres diferentes.
- **Palabras Reservadas:** No se pueden usar las palabras reservadas del lenguaje (como if, while, return, let, const, class, entre otras) como identificadores.
- **Sin Espacios:** Los identificadores no pueden contener espacios. Para separar palabras, se suelen utilizar notaciones camelCase.



JavaScript

Definiciones de variables:

JavaScript permite declarar variables de tres maneras principales:

1. **var:** Es la forma antigua de declarar variables. Tiene alcance global o de función, lo que puede llevar a comportamientos inesperados. Se recomienda evitar su uso en nuevos proyectos.
2. **let:** Es la forma moderna de declarar variables. Tiene alcance de bloque (solo es accesible dentro del bloque donde se declara). Permite reasignar nuevos valores.
3. **const:** Se utiliza para declarar variables cuyo valor no debe cambiar. También tiene alcance de bloque. Intentar reasignar un valor a una variable const generará un error.



Herramientas y recursos de desarrollo



Visual Studio Code (VSC)

Es un editor de código fuente gratuito, multiplataforma y altamente personalizable desarrollado por **Microsoft**. Es ampliamente utilizado en el desarrollo de aplicaciones web y en muchos otros lenguajes de programación.

Tiene una interfaz intuitiva, su diseño es limpio y fácil de usar. Dispone de una gran variedad de extensiones que facilitan la escritura y depuración del código.

Integración con Git: Permite manejar el control de versiones directamente desde el editor. **Soporte Multilenguaje:** No solo es útil para JavaScript, sino para muchos otros lenguajes (*Python, Java, C++*, etc.).

Se descarga desde <https://code.visualstudio.com/>



**Visual Studio
Code**



Visual Studio Code (VSC)

Un **IDE** (*Integrated Development Environment*) es un entorno de desarrollo que reúne herramientas como edición de código, depuración, compilación y ejecución en una sola aplicación.

Visual Studio Code se clasifica principalmente como un editor de código.

Pero gracias a sus extensiones, se puede transformar en un entorno muy similar a un **IDE**.