¿Qué es JavaScript?

JavaScript es un lenguaje de programación o secuencias de comandos que se utiliza en páginas web y más allá. Te permite crear contenido dinámico, controlar multimedia, animar imágenes y mucho más. Imagina que HTML es la estructura y CSS es el estilo, y JavaScript es la capa que agrega interactividad y funcionalidad a tu sitio web .

Historia y evolución de JavaScript

1. \*\*Origen y desarrollo\*\*: JavaScript fue creado por Brendan Eich en 1995 mientras trabajaba en Netscape Communications Corporation. Originalmente, se llamaba LiveScript, pero luego fue renombrado a JavaScript para aprovechar el auge de Java en ese momento. Desde entonces, ha habido varias versiones y actualizaciones del lenguaje para mejorar su rendimiento y funcionalidades.

2. \*\*Características principales\*\*:

- \*\*Lado del cliente\*\*: JavaScript se ejecuta en el navegador del usuario, lo que permite la interacción directa con la página web sin necesidad de comunicación con el servidor.

- \*\*Sintaxis sencilla\*\*: Utiliza una sintaxis similar a otros lenguajes como C y Java, lo que facilita su aprendizaje y uso.

- \*\*Integración con HTML y CSS\*\*: Se puede integrar fácilmente con HTML y CSS, permitiendo la creación de páginas web completas y dinámicas¹.

3. \*\*Frameworks y bibliotecas\*\*: A lo largo de los años, han surgido numerosos frameworks y bibliotecas de JavaScript, como React, Angular y Vue.js, que simplifican el desarrollo de aplicaciones web y móviles.

4. \*\*Aplicaciones y beneficios\*\*: JavaScript se utiliza en todo, desde aplicaciones web interactivas hasta aplicaciones móviles de vanguardia. Permite crear contenido dinámico, controlar multimedia y animar imágenes, brindando una experiencia más rica al usuario.

Principales usos de JavaScript

Desarrollo de sitios web del lado del cliente (front-end): JavaScript permite crear páginas web interactivas. Desde formularios hasta animaciones, todo lo que ves en una página web dinámica se debe a JavaScript1.

Desarrollo de aplicaciones móviles: JavaScript se utiliza en frameworks como React Native y Ionic para crear aplicaciones móviles híbridas o que compilan a código nativo.

Construcción de servidores web y aplicaciones de servidor: Node.js, basado en JavaScript, permite desarrollar servidores y aplicaciones del lado del servidor.

Juegos: Muchos juegos en línea utilizan JavaScript para la lógica del juego, gráficos y animaciones.

Similitudes y diferencias entre Java y JavaScript

Propósito:

Java: Es un lenguaje de propósito general, orientado a objetos. Se utiliza para construir aplicaciones complejas a escala empresarial.

JavaScript: Es un lenguaje interpretado utilizado principalmente para el desarrollo web y la ejecución de scripts en páginas

Contexto de ejecución:

Java: Se ejecuta en servidores y dispositivos móviles.

JavaScript: Se ejecuta dentro del navegador web y se utiliza para interactuar con el contenido HTML y CSS de una página

Aplicaciones:

Java: Maneja grandes cantidades de datos y realiza operaciones complejas en aplicaciones empresariales.

JavaScript: Proporciona una experiencia dinámica y rica al usuario final en sitios web1.

Origen de la similitud en los nombres

Java: Fue creado en 1995 por Sun Microsystems como un lenguaje de programación orientado a objetos para aplicaciones empresariales. Java se estaba volviendo muy popular en esa época12.

JavaScript: En el mismo año, Brendan Eich desarrolló JavaScript en Netscape Communications como un lenguaje de programación de scripts para páginas web interactivas. Originalmente, se llamaba LiveScript, pero luego se renombró a JavaScript para aprovechar la popularidad creciente de Java12.

Marketing inteligente: Netscape vio una oportunidad al darle un nombre similar a JavaScript. Aprovecharon la fama de Java para ganar visibilidad y atraer a los desarrolladores web32.

Lenguajes de programación de propósito general

Python: Ampliamente utilizado en desarrollo web, aprendizaje automático y automatización de tareas2.

Java: Ideal para aplicaciones empresariales y desarrollo móvil3.

C++: Conocido por su rendimiento y uso en sistemas embebidos y videojuegos.

C#: Principalmente utilizado en el desarrollo de aplicaciones Windows y juegos con Unity.

JavaScript: Esencial para la interacción en línea y el desarrollo web3

Lenguajes de programación específicos

Logo: Utilizado para enseñar programación a niños.

Verilog y VHSIC: Para diseño de circuitos digitales.

R y S: En estadísticas.

Mata: Para programación matricial.

Mathematica y Maxima: En matemáticas y fórmulas de hojas de cálculo.

SQL: Para consultas en bases de datos relacionales.

Yacc: Para crear parseadores.

Expresiones regulares: Para análisis léxico.

Csound: Lenguaje de síntesis digital.

GraphViz y GrGen: Utilizados para graficar y reescribir gráficas1.

Los paradigmas de programación

En el ámbito de la programación, un **paradigma** es un conjunto de principios y directrices que define un enfoque particular para diseñar, estructurar y escribir código. Cada paradigma impone una forma única de pensar sobre cómo debería desarrollarse el software y cómo interactúan sus componentes. Existen varios tipos de paradigmas de programación:

1. **Programación imperativa**: En este paradigma, se describen detalladamente los pasos que debe seguir el programa para alcanzar un estado deseado. Lenguajes como C y Pascal son ejemplos clásicos de este enfoque.
2. **Programación declarativa**: A diferencia de la programación imperativa, aquí se centra en describir el resultado deseado sin especificar los pasos detallados para llegar allí. Dos subcategorías comunes son la programación funcional y la lógica. Lenguajes como Haskell y Prolog son representativos de estas subcategorías, respectivamente.
3. **Programación orientada a objetos (OOP)**: En este paradigma, los programas se estructuran alrededor de “objetos”, que son instancias de clases que encapsulan datos y métodos. Java, Python y C++ siguen este enfoque.
4. **Programación orientada a aspectos (AOP)**: Permite modularizar aspectos transversales a través de la aplicación, como el registro o la seguridad, que no encajan fácilmente en un paradigma OOP tradicional. AspectJ es un ejemplo de lenguaje que implementa la programación orientada a aspectos.
5. **Programación funcional**: Se centra en tratar las computaciones como evaluaciones de funciones matemáticas y evita el cambio de estado y datos mutables. Lenguajes como Lisp, Haskell y Erlang siguen este paradigma.
6. **Programación lógica:** Utiliza la lógica formal para expresar reglas y relaciones que gobiernan el problema en cuestión

Programación orientada a objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en el concepto de “objetos”. En este enfoque, los objetos contienen información (como campos o atributos) y código (como métodos). Aquí tienes algunos puntos clave:

Clases y Objetos:

Clase: Es una plantilla o modelo que define las propiedades y comportamientos de un tipo de objeto.

Objeto: Es una instancia específica de una clase. Los objetos se crean a partir de las clases y pueden interactuar entre sí.

Principios Fundamentales:

Abstracción: Representa solo los detalles relevantes de un objeto y oculta la complejidad interna.

Encapsulamiento: Agrupa datos (atributos) y métodos relacionados en un solo objeto, protegiendo su acceso desde fuera.

Herencia: Permite que una clase herede propiedades y métodos de otra clase.

Polimorfismo: Permite que objetos de diferentes clases respondan de manera similar a un mismo método.

Motivación:

La POO busca reutilizar código y modelar problemas de manera más natural, similar a cómo pensamos en la vida real.

Permite diseñar aplicaciones complejas sin que el código se vuelva inmanejable123

**Programación funcional**:

Se centra en tratar las computaciones como evaluaciones de funciones matemáticas y evita el cambio de estado y datos mutables. Lenguajes como Lisp, Haskell y Erlang siguen este paradigma.

Lenguaje interpretado vs. lenguaje compilado

1. **Lenguajes Compilados**:
   * En un lenguaje compilado, el código fuente se traduce directamente a **código máquina** que el procesador puede ejecutar.
   * Estos lenguajes suelen ser más **rápidos** y **eficientes** en tiempo de ejecución que los lenguajes interpretados.
   * Ejemplos de lenguajes compilados puros incluyen **C**, **C++**, **Erlang**, **Haskell**, **Rust** y **Go**[1](https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/).
2. **Lenguajes Interpretados**:
   * En un lenguaje interpretado, el código fuente se ejecuta **línea por línea** en tiempo real.
   * No se compila previamente a código máquina; en cambio, un **intérprete** lee y ejecuta el código directamente.
   * Aunque solían ser más lentos, con el desarrollo de la **compilación justo a tiempo**, la diferencia se ha reducido.
   * Ejemplos comunes de lenguajes interpretados son **PHP**, **Ruby**, **Python** y, sí, **JavaScript**[1](https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/)[2](https://trspos.com/javascript-es-compilado-o-un-lenguaje-interpretado/).

Lenguaje interpretado vs. lenguaje compilado en javascript

1. **Lenguajes Compilados**:
   * En un lenguaje compilado, el código fuente se traduce directamente a **código máquina** que el procesador puede ejecutar.
   * Estos lenguajes suelen ser más **rápidos** y **eficientes** en tiempo de ejecución que los lenguajes interpretados.
   * Ejemplos de lenguajes compilados puros incluyen **C**, **C++**, **Erlang**, **Haskell**, **Rust** y **Go**[1](https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/).
2. **Lenguajes Interpretados**:
   * En un lenguaje interpretado, el código fuente se ejecuta **línea por línea** en tiempo real.
   * No se compila previamente a código máquina; en cambio, un **intérprete** lee y ejecuta el código directamente.
   * Aunque solían ser más lentos, con el desarrollo de la **compilación justo a tiempo**, la diferencia se ha reducido.
   * Ejemplos comunes de lenguajes interpretados son **PHP**, **Ruby**, **Python** y, sí, **JavaScript**[1](https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/)[2](https://trspos.com/javascript-es-compilado-o-un-lenguaje-interpretado/).

**case sensitivity**

1. **¿Qué es la Sensibilidad a Mayúsculas y Minúsculas?**:
   * La sensibilidad a mayúsculas y minúsculas, conocida técnicamente como **case sensitivity**, se refiere a la distinción que hacen los sistemas entre las letras en mayúscula y minúscula al momento de ingresar una contraseña.
   * Esto significa que, por ejemplo, las letras “A” y “a” serían consideradas como caracteres diferentes, aumentando así las posibles combinaciones al crear una contraseña[1](https://skiller.education/case-sensitive/).
2. **Incremento en la Entropía de las Contraseñas**:
   * La **entropía** mide la imprevisibilidad o aleatoriedad de una contraseña.
   * Al ser sensibles a mayúsculas y minúsculas, las contraseñas adquieren un nivel de entropía más alto, ya que el número de posibles combinaciones se multiplica.
   * Esto dificulta que los atacantes adivinen o descifren las contraseñas mediante técnicas de fuerza bruta o ataques de diccionario[1](https://skiller.education/case-sensitive/).
3. **Mejores Prácticas en la Creación de Contraseñas**:
   * Para aprovechar al máximo la sensibilidad a mayúsculas y minúsculas, es recomendable seguir ciertas mejores prácticas al crear contraseñas:
     + Utiliza una mezcla de letras mayúsculas y minúsculas, números y símbolos.
     + Evita palabras comunes o secuencias predecibles.
     + Opta por contraseñas largas y únicas para cada servicio o plataforma[1](https://skiller.education/case-sensitive/).
4. **Desafíos de la Sensibilidad a Mayúsculas y Minúsculas**:
   * A pesar de sus ventajas, recordar contraseñas complejas puede ser difícil para los usuarios.
   * Esto puede llevar a soluciones menos seguras, como la reutilización de contraseñas

Sintaxis básica de JavaScript

1. **Variables**: En JavaScript, puedes definir variables usando las palabras clave const, var o let.
2. **Tipos de datos**:
   * **Números**: Pueden ser enteros o decimales.
   * **Cadenas de texto**: Se representan con comillas simples o dobles.
   * **Booleanos**: Representan valores verdaderos o falsos.
   * **Arreglos**: Colecciones ordenadas de elementos.
   * **Objetos**: Estructuras que almacenan propiedades y valores.
3. **Bucles**:
   * while: Repite según una condición.
   * for: Repite hasta que se alcance un límite.
4. **Evaluaciones condicionales**:
   * if/else: Permite ejecutar código según una condición.
   * switch: Similar al if, pero con múltiples casos.
5. **Asignación**:
   * El símbolo = se utiliza para asignar valores a variables.