

Diplomatura en



Programación FullStack Developer

JAVASCRIPT

Fundamentos de JavaScript: Tipos de
Datos y Variables

Fundamentos

JavaScript (JS) es un lenguaje de programación interpretado. Se utiliza principalmente para agregar interactividad y dinamismo a las páginas web.

Lenguaje interpretado: Es un tipo de lenguaje de programación para el cual la mayoría de sus implementaciones se ejecutan instrucciones directamente, sin requerir una compilación previa a un conjunto de instrucciones en lenguaje de máquina. Esto se logra a través de un programa llamado intérprete.

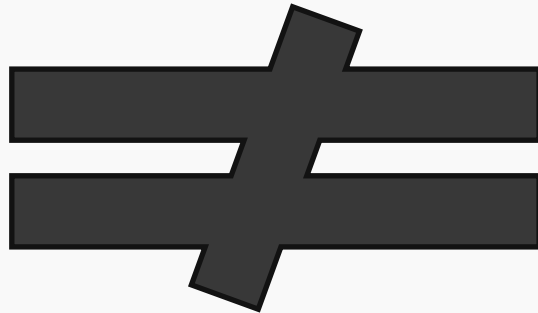
En este caso, los navegadores se encargarán de interpretar y ejecutar el código JavaScript. Se recomiendan los siguientes:



JavaScript vs Java

Son dos **lenguajes diferentes**, y no se deben entender como sinónimos.

Sin ahondar mucho, diremos que Java es un lenguaje que se ejecuta en el servidor (Back-End), su sintaxis y propósitos son diferentes.



JavaScript y sus usos

Es un lenguaje de programación versátil que se puede usar en una variedad de contextos y plataformas. Se puede utilizar para realizar aplicaciones móviles, servidores, aplicaciones web, aplicaciones de escritorio, realidad aumentada, realidad virtual, etc.

Ventajas:

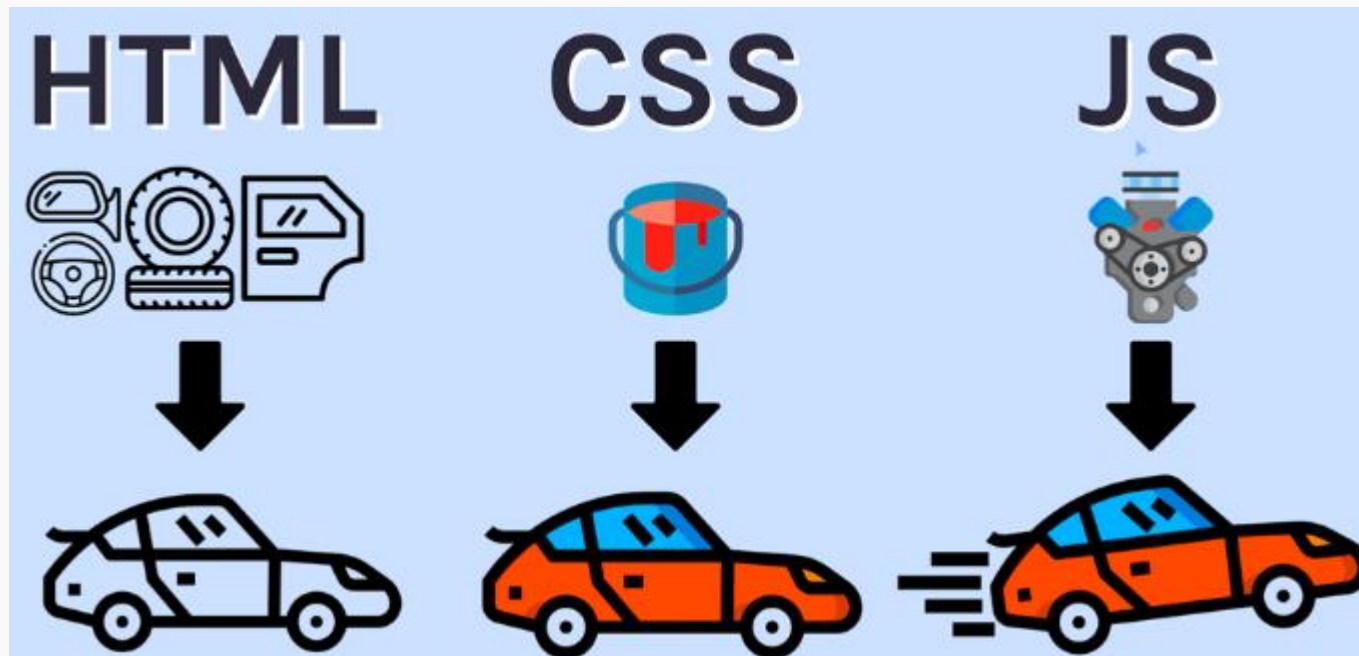
JavaScript es el único lenguaje que se ejecuta en prácticamente todos los navegadores modernos sin necesidad de plugins ni herramientas adicionales, tiene una gran comunidad activa, se integra fácilmente a otras tecnologías, las herramientas de desarrollo (DevTools) en navegadores como Chrome o Firefox son extremadamente poderosas para depurar y analizar código JavaScript, el mercado laboral por su popularidad, existe una demanda constante de desarrolladores de JavaScript, lo que ofrece amplias oportunidades de empleo y crecimiento profesional, un ecosistema rico ya que es uno de los repositorios de software más grandes del mundo, proporcionando acceso a una vasta cantidad de bibliotecas y módulos.

Desventajas:

A partir de su versatilidad vinieron algunas desventajas que iremos abordando en las clases y como se pueden solucionar o bien atenuar.

JavaScript y HTML

Hablemos un poco de como se trabaja con JavaScript y HTML, así como integramos CSS por medio de hojas de estilo, lo mismo haremos con JavaScript, es decir, tenemos 3 maneras de integrar código JavaScript en el HTML.



Código JavaScript en línea:

Usando una etiqueta button

Por medio del atributo "onclick" de la etiqueta button podemos pasar cierto código JavaScript para que se ejecute.

```
<button onclick="alert('Hola Mundo')">Hacé click acá</button>
```

Código JavaScript en el documento HTML:


```
<button onclick="mostrarMensaje()">Haz clic aquí</button>  
<script>  
    function mostrarMensaje() { alert('Hola Mundo!'); }  
</script>
```


Y de la manera que vamos a trabajar es **"Vinculando un archivo JavaScript externo"**:


Es una buena práctica mantener el código JavaScript en un archivo separado. Esto hace que el código sea más legible y permite reutilizar el mismo archivo JS en diferentes páginas.

Estructura base

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <script src="main.js"></script>
</body>
</html>
```

 index.html

 main.js

 style.css

Estructura de archivos que utilizaremos para empezar nuestro proyecto.

Acá empezaremos a trabajar con un poco de HTML, CSS pero nuestro fuerte será JavaScript

Sintaxis básica

La fundamentación de JavaScript se sustenta en su sintaxis básica, que nos guía en la escritura de código de forma precisa y eficaz. Antes de adentrarnos en la programación con este lenguaje, es crucial familiarizarnos con ciertas reglas y normativas que aseguran la calidad y legibilidad del código. A medida que desglosamos estos principios, nos embarcaremos en un viaje de aprendizaje profundo, explorando cada aspecto en detalle para fortalecer nuestra comprensión y habilidad en la escritura de scripts efectivos y eficientes en JavaScript.

Comentarios:

El uso de comentarios en el código es fundamental para garantizar la legibilidad y el mantenimiento del mismo. Ayudan no solo a nuestro "yo" futuro a comprender rápidamente lo que estábamos tratando de lograr, sino también a cualquier otro desarrollador que pueda trabajar en el proyecto más adelante.

En JavaScript, disponemos de dos principales maneras de incorporar comentarios:

Comentarios de línea única:

Son útiles para hacer anotaciones breves o para desactivar temporalmente una línea de código. Se crean utilizando dos barras diagonales (//). Todo lo que sigue a estas barras en la misma línea se considera un comentario y no se ejecutará.

Ejemplo:

```
// Este es un comentario de una sola línea
```

Comentarios en bloque:

Están diseñados para descripciones más extensas o para desactivar múltiples líneas de código a la vez. Se inician con una barra diagonal seguida de un asterisco (/*) y terminan con un asterisco seguido de una barra diagonal (*/).

Ejemplo:

```
/*  
    Este es un comentario en bloque. Puede extenderse  
    a lo largo de varias líneas y es especialmente útil  
    para describir la funcionalidad general de un segmento de código  
*/
```

Es vital desarrollar el hábito de comentar el código de manera adecuada. No solo para aclarar la funcionalidad o la intención detrás de segmentos específicos de código, sino también para proporcionar contexto, advertir sobre posibles problemas o indicar áreas que podrían necesitar revisión o mejora en el futuro.

Secuencialidad y Reglas de Sintaxis en JavaScript

En JavaScript, el código se ejecuta secuencialmente de arriba hacia abajo. Esto significa que cada línea de código se interpreta y ejecuta en el orden en que aparece. Esta secuencialidad es esencial para comprender el flujo del programa, especialmente cuando se trata de declaraciones de variables, inicializaciones y llamadas a funciones. Con base en las reglas de sintaxis, es crucial asegurarse de que las variables estén correctamente declaradas e inicializadas antes de su uso y que los bloques de código estén delimitados de manera adecuada. Además, siendo JavaScript un lenguaje sensible a mayúsculas y minúsculas, el orden en el que se escriben las declaraciones y se hacen las referencias es vital para el correcto funcionamiento del programa.

Palabras reservadas

Vamos a seguir hablando desde ahora en más sobre la sintaxis de JavaScript, mientras vamos desarrollando cada tema en cuestión, ahora hablemos de las palabras reservadas en el lenguaje.

Dentro de cualquier lenguaje de programación, y JavaScript no es la excepción, hay ciertas palabras que tienen un significado especial y que no se pueden usar como nombres de variables, funciones, clases, o cualquier otro identificador.

Si trabajamos con Visual Studio Code, este nos dará una gran ayuda ya que nos lo va a mostrar de un color distinto al resto del código que estemos escribiendo.

Variables y constantes

Son elementos fundamentales para almacenar y gestionar datos dentro de un programa. Aunque históricamente solo había una manera de declarar variables usando la palabra reservada "var", con la introducción de ES6 (ECMAScript 2015), surgieron dos nuevas formas: "let" para variables y "const" para constantes (En este curso vamos a utilizar let y const según la necesidad).

Ejemplo:

```
let variable;  
const CONSTANTE;
```

Reglas de sintaxis para nombrar variables y constantes:

- Los nombres pueden contener letras, números, guiones bajos (_) y el signo de dólar (\$), pero no pueden empezar con un número.
- No se pueden usar palabras reservadas de JavaScript como nombres de variables o constantes.
- Los nombres son sensibles a mayúsculas y minúsculas, lo que significa que "nombreVariable" y "nombrevariable" serían consideradas diferentes.
- Es recomendable utilizar nombres descriptivos y seguir convenciones de nomenclatura, como camelCase, para mejorar la legibilidad del código.

Tipo de datos primitivos

Estos tipos de datos son fundamentales en la programación y sirven como bloques básicos para construir estructuras más complejas.

- **Number:** Representa tanto números enteros como flotantes.
- **String:** Representa una secuencia de caracteres. Se delimita usando comillas simples ("), dobles ("") o invertidas (~).
- **Boolean:** Representa un valor verdadero o falso. Tiene solo dos posibles valores: true y false.
- **Undefined:** Es un tipo de dato que se asigna a una variable que ha sido declarada pero aún no tiene un valor asignado.
- **Null:** Representa la ausencia intencional de cualquier valor o referencia. Es importante destacar que null es distinto de undefined.

Inicializar variables y asignación un valor

La inicialización y asignación de valores a variables y constantes es una operación fundamental. Para hacerlo, se utiliza el símbolo “=”. Este símbolo, conocido como operador de asignación, permite establecer un valor específico a una variable o constante.

Inicialización:

Es el proceso de declarar una variable o constante por primera vez, preparándola para su uso posterior. Por ejemplo:

```
let variable;  
let anioDeNacimiento;
```

Asignación de valor:

Una vez inicializada, podemos asignar un valor a nuestra variable usando el símbolo =

```
anioDeNacimiento = 1988;
```

En muchos casos, la inicialización y la asignación de valores se realizan en un solo paso:

```
let nombre = "Juan";  
const PI = 3.14159;
```

Consejos para nombrar variables y constantes:

Convención camelCase: En JavaScript, es común utilizar la convención de nomenclatura camelCase para nombrar variables y constantes. Esto significa que si una variable o constante está compuesta por más de una palabra, la primera palabra comienza con minúsculas y todas las palabras subsiguientes inician con mayúsculas. Ejemplos: fechaDeNacimiento, direccionDeCorreo.

Descriptividad: Es recomendable que el nombre de la variable o constante describa claramente su propósito o uso. Por ejemplo, en lugar de usar "d" para una fecha, es preferible fechaDeCreacion.

Evitar nombres genéricos: Nombres como data, info o temp pueden ser confusos. Es más claro usar nombres específicos que describan el contenido de la variable.

Consistencia: Es vital ser consistente en la nomenclatura a lo largo de todo el código. Esto hace que el código sea más legible para otros desarrolladores y para uno mismo en futuras revisiones.

No usar palabras reservadas: JavaScript tiene un conjunto de palabras reservadas que no deben ser usadas como nombres de variables o constantes, como `function`, `return` y `class`, entre otras.

Constantes: Si a una constante vamos a asignarle un tipo de dato primitivo, debemos definir su nombre en mayúsculas y si lo componen mas de una palabra separar con un guión bajo entre ellas. Ejemplo: -> `const PRIMER_NOMBRE = 'Pepe'`

Recordar estas buenas prácticas al nombrar variables y constantes puede marcar la diferencia entre un código difícil de leer y uno que sea claro y fácilmente mantenible. La claridad y legibilidad son esenciales para el desarrollo colaborativo y la longevidad del código.

Consistencia: Es vital ser consistente en la nomenclatura a lo largo de todo el código. Esto hace que el código sea más legible para otros desarrolladores y para uno mismo en futuras revisiones.

No usar palabras reservadas: JavaScript tiene un conjunto de palabras reservadas que no deben ser usadas como nombres de variables o constantes, como function, return y class, entre otras.

Constantes: Si a una constante vamos a asignarle un tipo de dato primitivo, debemos definir su nombre en mayúsculas y si lo componen mas de una palabra separar con un guión bajo entre ellas. Ejemplo: -> `const PRIMER_NOMBRE = 'Pepe'`

Recordar estas buenas prácticas al nombrar variables y constantes puede marcar la diferencia entre un código difícil de leer y uno que sea claro y fácilmente mantenible. La claridad y legibilidad son esenciales para el desarrollo colaborativo y la longevidad del código.

Operaciones básicas

Operaciones Aritméticas:

Podemos trabajar con numbers ya sea guardándolos en una variable previamente o realizando una operación y guardarlos también en una variable.

Ejemplos:

```
let resultado = 1 + 10 + 20

let numero1 = 10

let numero2 = 30

let suma = numero1 + numero2

let resta = numero1 - numero2

let multiplicacion = numero1 * numero2

let division = numero1 / numero2

let resto = numero1 % numero2
```

Operaciones con strings:

El manejo de cadenas de texto, o "strings", es esencial en la programación. JavaScript proporciona una serie de operaciones y métodos para trabajar con strings. Veamos algunas de las operaciones más comunes:

```
// Concatenación

let saludo = 'Hola, ' + 'Gente'

// Longitud

let palabra = 'JavaScript'
let longitud = palabra.length // Nos devuelve el número 10

// Acceder a caracteres

let letra = palabra[1] // la letra devuelta es "a"
```

Los strings tienen 2 propiedades muy útiles: la longitud y cada carácter tiene un índice, este empieza desde la posición 0, por ese motivo el valor de letra es a

Operaciones de Comparación:

Estas operaciones devuelven un valor booleano (true o false)

```
// Igual -> compara si 2 valores son iguales
5 == 5; // true

// Estrictamente Igual (===): Compara valor y tipo.
5 === '5'; // false

// Diferente (!=):
5 != 6; // true

// Estrictamente Diferente (!==):
5 !== '5'; // true

// Mayor que (>), Menor que (<), Mayor o igual que (>=), Menor o igual que (<=):
10 > 5; // true
7 <= 7; // true
```

Nosotros vamos a trabajar con las comparaciones estrictas, es decir, usando triple igual o un signo de exclamación y 2 iguales

alert, prompt y console

Interacción y Depuración en JavaScript:

Estas tres herramientas son esenciales para la interacción con el usuario y el diagnóstico en JavaScript. Veamos una breve descripción y ejemplos de cada uno:

alert:

Muestra un cuadro de diálogo con un mensaje para el usuario.

```
alert("¡Hola, mundo!");
```

prompt:

Muestra un cuadro de diálogo con un campo de texto para que el usuario ingrese información. Retorna la entrada del usuario como un string o null si el usuario presiona cancelar.

```
let nombre = prompt("¿Cuál es tu nombre?");
```

console:

Proporciona acceso al área de consola del navegador, permitiendo imprimir mensajes, errores, advertencias y más. Es una herramienta fundamental para la depuración.

```
console.log("Este es un mensaje en la consola.");  
console.warn("¡Advertencia! Algo puede estar mal.");  
console.error("Error: algo salió mal.");
```

Mas formas de concatenar strings:

Notemos que prompt se asignó a una variable, esto es porque nos devuelve un resultado, y es porque prompt es un "método".

De acá nace la necesidad de concatenar strings. En JavaScript hay una forma mucho más optima de concatenar strings y es usando `` las comillas invertidas (templates strings).

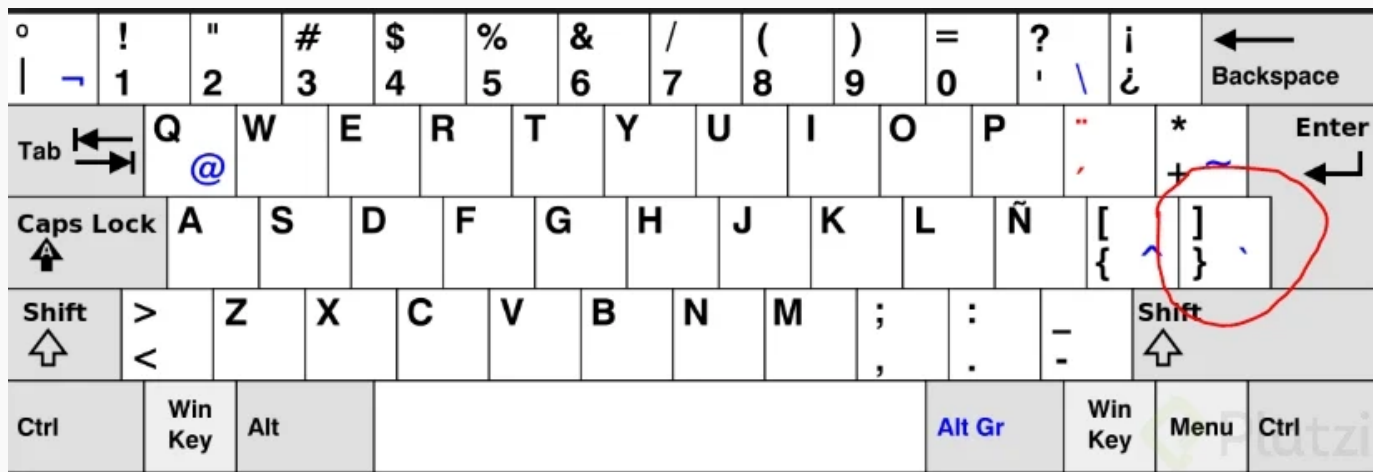
Ejemplo

```
let nombre = prompt("¿Cuál es tu nombre?");  
  
let oracion = `Hola que tal ${nombre}, un gusto!`;
```

De esta forma podremos concatenar variables o constantes a nuestros strings de manera mucho más fácil.

```
let nombre = "Juan";  
let saludo = `Hola, ${nombre}!`;   
console.log(saludo); // Hola, Juan!
```

Colocar las comillas invertidas:



Presionar Alt Gr + back quote (presionar esta dos veces)

O bien:

Alt + 96

Ejercicios

Ejercicio 1: Crea un programa que solicite al usuario su nombre y apellido. Luego, muestra una alerta con su nombre completo (nombre y apellido concatenados) utilizando template strings.

Ejercicio 2: Pide al usuario que ingrese una frase. Luego, muestra en la consola la longitud de la frase utilizando template strings.

Ejercicio 3: Solicita al usuario que ingrese dos palabras. Concatena las dos palabras y muestra el resultado en la consola. Además, calcula la longitud total de las dos palabras concatenadas y muestra el resultado en la consola.

Recordar que en estos ejercicios debemos utilizar template strings para la concatenación de cadenas de texto, también usar prompt, console.log y alert para interactuar con el usuario y mostrar los resultados.