



Front-End JS

Clase 09 - "Introducción a JavaScript"

¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase

Clase 08.

Bootstrap y Git

1. ¿Qué es un framework?
2. Componentes básicos de Bootstrap
3. Cómo Grid agregar un componente a nuestro proyecto
4. Git: Descarga de Git
5. Crear un repositorio externo (GitHub)
6. Comandos básicos (Init, commit, push)

Clase 09.

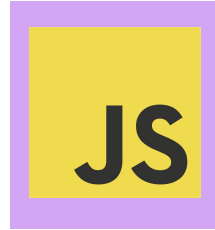
JS 1 - Introducción a JavaScript

1. ¿Qué es y para qué se usa JavaScript?
2. Conceptos generales. Sintaxis básica
3. Variable: ¿qué es y cómo declararla? Tipos
4. Asignación y cambio del valor
5. Operadores aritméticos
6. Conversión a entero y flotante

Clase 10.

JS 2 - Condicionales y ciclos

1. Diagrama de flujo
2. Condicional: ¿Qué es?
3. Operadores lógicos y de comparación: ¿Qué son y cuál es su uso en los condicionales?
4. Bucles: ¿Qué son? Tipos y diferencias entre sí
5. Cómo combinar operadores lógicos y ciclos



JavaScript

JavaScript



JavaScript (JS) es un lenguaje de programación, un mecanismo con el que podemos indicarle al navegador qué tareas debe realizar, en qué orden y cuántas veces, entre otras. Con JS agregamos comportamiento a nuestro sitio, permitiendo al usuario interactuar con él, rompiendo con la idea de una web estática. Junto con HTML y CSS es la tercera pieza fundamental del desarrollo web.

ECMAScript es el estándar que a partir del año 2015 a la actualidad se encarga de regir como debe ser interpretado y funcionar el lenguaje JavaScript. En la actualidad, JS puede ser interpretado y procesado por una multitud de plataformas, entre las que se encuentran los navegadores web.

JavaScript: Versiones



JavaScript ha sufrido modificaciones que los navegadores han debido implementar para proporcionar soporte a cada versión de ECMAScript cuanto antes. La lista de versiones de ECMAScript es la siguiente:

Ed.	Fecha	Nombre formal / informal	Cambios significativos
1	JUN/1997	ECMAScript 1997 (ES1)	Primera edición
5	DIC/2009	ECMAScript 2009 (ES5)	Strict mode, JSON, etc...
6	JUN/2015	ECMAScript 2015 (ES6)	Clases, módulos, generadores, hashmaps, sets, for of, proxies...
7	JUN/2016	ECMAScript 2016	Array includes(), Exponenciación **
8	JUN/2017	ECMAScript 2017	Async/await
9	JUN/2018	ECMAScript 2018	Rest/Spread operator, Promise.finally()...
10	JUN/2019	ECMAScript 2019	Flat functions, trimStart(), errores opcionales en catch...
11	JUN/2020	ECMAScript 2020	Dynamic imports, BigInt, Promise.allSettled

JavaScript: Características



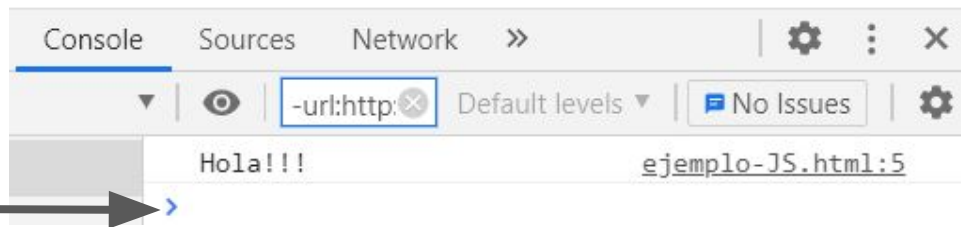
- Lenguaje del lado del cliente: el código se ejecuta en el cliente (navegador); no se necesita acceso al servidor.
- Orientado a objetos: utiliza clases y objetos como estructuras reutilizables.
- De tipado débil o no tipado: no es necesario especificar el tipo de dato al declarar una variable.
- De alto nivel: su sintaxis se encuentra alejada del nivel máquina, más cercano a un lenguaje de las personas.
- Lenguaje interpretado: el navegador convierte las líneas de código en el lenguaje de la máquina sin necesidad de realizar un proceso de compilado.
- Muy utilizado por desarrolladores: es uno de los lenguajes más demandados de los últimos años por su versatilidad y su infinita capacidad para crear plataformas cada vez más atractivas.
- Interactividad con el usuario: podemos validar el formato de los datos de un formulario (una dirección de email directamente desde el navegador del cliente), ahorrando tiempo y recursos del servidor.

JavaScript: Primeros pasos

JS

El código de nuestro script debe ser incorporado al código HTML, de forma similar a lo que ocurre con las hojas de estilo CSS. Existen tres formas de agregar código JavaScript a una página web. Una de ellas es utilizar la etiqueta `<script>` en el `<head>` de nuestro documento (referencia interna):

```
<html>
  <head>
    <title>Título de la página</title>
    <script>
      console.log("Hola!!!")
    </script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```



Desde el inspector del navegador podremos ver que se escribe el texto "Hola!!!" en la consola. Esto se logra a través de la instrucción **console.log()**

JavaScript: Configuración

JS

La ubicación de la etiqueta <script>

Según cómo esté ubicada la etiqueta <script> el navegador descarga ejecuta el archivo JavaScript en momentos diferentes:

1. En <head>: antes de empezar a dibujar la página, cuando está en blanco.
2. En <body>: cuando la página se haya dibujado hasta el <script>.
3. Antes de </body>: cuando la página se haya dibujado en su totalidad.



JavaScript: Consola

JS

Para acceder a la consola Javascript del navegador pulsamos CTRL+SHIFT+J.

Un clásico ejemplo utilizado cuando se comienza a programar es crear un programa que muestre por pantalla un texto, generalmente el texto «Hola Mundo». O mostrar el resultado de alguna operación matemática. A continuación, el código JS para realizar ambas tareas, y la salida que podemos ver en la consola del navegador:

```
console.log("Hola Mundo");  
console.log(2 + 2);
```

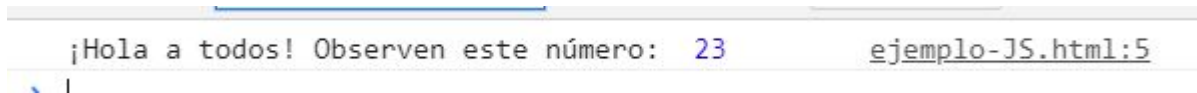


JavaScript: Consola

JS

Podemos mostrar texto, valores numéricos, etc. separados por comas:

```
console.log("¡Hola a todos! Observen este número: ", 5 + 18);
```



En esta consola podemos escribir funciones o sentencias de JavaScript que se ejecutan en la página que se encuentra en la pestaña actual del navegador. De esta forma podemos observar los resultados que nos devuelve en la consola al realizar diferentes acciones.

JavaScript: Consola

JS

JS posee, además de `console.log`, varias instrucciones similares para interactuar con el desarrollador:

Función	Descripción
<code>console.log()</code>	Muestra la información proporcionada en la consola Javascript.
<code>console.info()</code>	Equivalente al anterior. Se utiliza para mensajes de información.
<code>console.warn()</code>	Muestra información de advertencia. Aparece en amarillo.
<code>console.error()</code>	Muestra información de error. Aparece en rojo.
<code>console.clear()</code>	Limpia la consola. Equivalente a pulsar CTRL + L o escribir <code>clear()</code> .

Texto

Info



Warning



Error

JS: Incorporar archivo externo

JS

Podemos vincular al documento HTML un archivo con extensión .js usando la etiqueta `<script>`, haciendo referencia al nombre del archivo JavaScript con el atributo `src` (source):

```
<html>
  <head>
    <title>Título de la página</title>
    <script src="index.js"></script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

Los archivos .js se suelen incorporar en una carpeta llamada "js".

JS: Comentarios

JS

Los comentarios son utilizados por los programadores para anotaciones. No son tenidos en cuenta por el navegador.

Comentario de línea

```
// esto es un comentario de línea
```

Comentario de bloque

```
/*  
esto es un comentario de bloque (multilínea)  
*/
```

Son un buen recurso cuando queremos omitir la ejecución de ciertas instrucciones.

JS: Variables

JS

¿Qué son?

Es un pequeño espacio en la memoria, donde se guarda un dato. Podemos imaginarlas como “cajitas” dentro de nuestro programa. Tienen tres características:

- Nombre: debe ser representativo de la información que contiene. Se utiliza para diferenciar unas de otras y hacer referencia a ellas.
- Tipo de dato: puede ser número, texto, valores booleanos, etc.
- Contenido: el valor concreto que posee el dato almacenado.

Se llaman variables porque pueden cambiar su valor a lo largo del programa. Un programa puede tener muchas variables, y cada una de ellas tendrá un nombre que la identifique, un valor y un tipo de dato.

JS: Variables

JS

¿Como se declaran?

Una variable que se ha declarado con var pero a la que no se le asignó un valor se dice que está indefinida (no conocemos el tipo de dato):

```
var num3;
```

En este caso la variable está “vacía”, no está definido el valor que colocará en memoria. No se ha asociado ningún contenido a esa variable.

```
var num4 = 5;
```

Las sentencias en JS finalizan con “;”. La imagen anterior corresponde a la declaración de la variable “num4” con un valor numérico entero de “5”.

JS: Variables

JS

¿Como se nombran?

Los nombres de las variables (o identificadores) permiten distinguir una de otras. Para asignar los nombres de las variables debemos seguir ciertas reglas:

Un identificador de JavaScript debe comenzar con una letra, un guión bajo (_) o un signo de dólar (\$). Los siguientes caracteres también pueden ser dígitos (0 - 9). JavaScript distingue entre mayúsculas y minúsculas (es case-sensitive).

Se recomienda usar la escritura camelCase en el nombre de variables que tienen más de una palabra.

JS: Variables

JS

Podemos cambiar el valor de una variable durante el flujo del programa:

```
var IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

El = es el operador de asignación, y permite asignar un valor a una variable. Ese valor puede ser el resultado de una operación aritmética, que se evalúa y luego se asigna su resultado a la variable:

Luego de ejecutar esa línea, la variable “resultado” contiene el valor “8”.

```
var resultado= (1 + 3) * 2;
```

JS: Constantes

JS

El concepto de constante es similar al de variable, con la salvedad de que la información que contiene es siempre la misma (no puede variar durante el flujo del programa). Declaramos las constantes utilizando `const`. Su sintaxis es:

```
const PI= 3.141592;  
const IVA= 21;
```

Si intentamos modificar el valor de una constante, obtenemos un error:

```
const IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

```
✖ Uncaught TypeError: Assignment to constant variable.  
at ejemplo-JS.html:9
```

JS: Tipos de datos

JS

Las variables de JavaScript pueden contener distintos tipos de datos: numérico, cadena de caracteres, lógicos, indefinido, null, objetos y más. El tipo de dato es la naturaleza del contenido de la variable o constante. JavaScript tiene tipado dinámico, es decir que la misma variable se puede utilizar para contener diferentes tipos de datos:

```
var x;           // ahora x es indefinido (no tiene un valor definido)
x = 5;           // ahora es numérico (5)
x = "Juan";      // ahora es una cadena de caracteres o string ("Juan")
```

JavaScript deduce cuál es el tipo de dato de la variable. El tipo de dato asociado a esa variable lo determina el dato que se almacena en ella. Y si luego se le asigna un valor de otro tipo, el tipo de la variable cambia.

JS: Tipos de datos

JS

Los tipos de datos en JavaScript son los siguientes:

Tipos de datos
primitivos

Tipo de dato	Descripción	Ejemplo básico
NUMBER number	Valor numérico (enteros, decimales, etc...)	42
STRING string	Valor de texto (cadenas de texto, caracteres, etc...)	'MZ'
BOOLEAN boolean	Valor booleano (valores verdadero o falso)	true
UNDEFINED undefined	Valor sin definir (variable sin inicializar)	undefined
FUNCTION function	Función (función guardada en una variable)	function() {}
OBJECT object	Objeto (estructura más compleja)	{}

JS: Tipos de datos

JS

El último estándar ECMAScript define nueve tipos de datos:

Seis tipos de datos primitivos [+info](#)

- Undefined [+info](#)
- Boolean [+info](#)
- Number [+info](#)
- String [+info](#)
- BigInt [+info](#)
- Symbol [+info](#)
- Null (tipo primitivo especial) [+info](#)
- Object [+info](#)
- Function [+info](#)

JS: Tipos de datos

JS

Identificar el tipo de dato de una variable

Para determinar qué tipo de dato tiene una variable utilizamos **typeof()**, que devuelve el tipo de dato primitivo asociado a una variable:

```
var s = "Hola, me llamo Juan"; // s, de string
var n = 28; // n, de número
var b = true; // b, de booleano
var u; // u, de undefined
```

```
console.log(typeof s);
console.log(typeof n);
console.log(typeof b);
console.log(typeof u);
```

string

number

boolean

undefined

JS: Tipos de datos

JS

Las variables numéricas

En JavaScript, los números constituyen un tipo de datos básico (primitivo). Para crear una variable numérica basta con escribirlas. No obstante, dado que en Javascript “todo es un objeto”, también podemos declararlas como si fuesen un objeto:

Constructor	Descripción
<code>new Number(n)</code>	Crea un objeto numérico a partir del número <code>n</code> pasado por parámetro.
<code>n</code>	Simplemente, el número en cuestión. Notación preferida.

```
// Declarados como literales
const n1 = 4;
var n2 = 15.8;
```

```
// Declarados como objetos
const n1 = new Number(4);
var n2 = new Number(15.8);
```


JS: Tipos de datos

JS

Objeto Number

Number es el objeto primitivo que permite representar y manipular valores numéricos. El constructor Number contiene constantes y métodos para trabajar con números. Valores de otro tipo pueden ser convertidos a números usando la función Number(). Su sintaxis es:

```
var a = new Number('123'); // a es igual a 123
var b = Number('123'); // b es igual a 123
console.log("a: ", a);
console.log("b: ", b);
```

```
a: ▼ Number {123} ⓘ
  ► __proto__: Number
    [[PrimitiveValue]]: 123
b: 123
```

Creamos el objeto a mediante el constructor y guardamos en b el valor de la cadena '123' en forma de número. Mostramos en consola ambos elementos.

JS: Tipos de datos

JS

Comprobaciones numéricas

Varias funciones de JS permiten conocer la naturaleza de una variable numérica (número finito, número entero, número seguro o si no es representable como un número). Devuelven true o false (un valor booleano). Las podemos ver en la siguiente tabla:

Método	Descripción
BOOLEAN <code>Number.isFinite(n)</code>	Comprueba si <code>n</code> es un número finito.
BOOLEAN <code>Number.isInteger(n)</code>	Comprueba si <code>n</code> es un número entero.
BOOLEAN <code>Number.isSafeInteger(n)</code>	Comprueba si <code>n</code> es un número seguro.
BOOLEAN <code>Number.isNaN(n)</code>	Comprueba si <code>n</code> no es un número.

JS: Comprobaciones numéricas

JS

Veamos dos ejemplos para cada una de estas funciones:

```
// ¿Número finito?
Number.isFinite(42); // true
Number.isFinite(Infinity); // false, es infinito
// ¿Número entero?
Number.isInteger(5); // true
Number.isInteger(4.6); // false, es decimal
// ¿Número seguro?
Number.isSafeInteger(1e15); // true
Number.isSafeInteger(1e16); // false, es un valor
no seguro
// ¿No es un número?
Number.isNaN(NaN); // true
Number.isNaN(5); // false, es un número
```

Numero finito (42):	true
Numero finito (infinito):	false
Numero entero (5):	true
Numero entero (4.6):	false
Numero seguro (1e15):	true
Numero seguro (1e16):	false
Not a Number (NaN):	true
Not a Number (5):	false

JS: Conversión numérica

JS

Es posible convertir cadenas de texto en números, para posteriormente realizar operaciones con ellos. Las funciones de parseo numérico, `parseInt()` y `parseFloat()`, permiten realizar esto:

Método	Descripción
NUMBER <code>Number.parseInt(S)</code>	Convierte una cadena de texto S en un número entero.
NUMBER <code>Number.parseInt(S, radix)</code>	Idem al anterior, pero desde una base radix .
NUMBER <code>Number.parseFloat(S)</code>	Convierte una cadena de texto S en un número decimal.
NUMBER <code>Number.parseFloat(S, radix)</code>	Idem al anterior, pero desde una base radix .

JS: Conversión numérica

JS

Veamos un ejemplo con `parseInt()`. Recibe como parámetro un texto que queremos convertir a número:

```
Number.parseInt("42"); // 42  
Number.parseInt("42€"); // 42  
Number.parseInt("Núm. 42"); // NaN  
Number.parseInt("A"); // NaN
```

```
parseInt (42) 42
```

```
parseInt (42$) 42
```

```
parseInt (Num. 42) NaN
```

```
parseInt (A) NaN
```

`parseInt()` funciona con variables de texto que contienen números o que comienzan por números. Sin embargo, si la variable de texto comienza por un valor que no es numérico, `parseInt()` devuelve un NaN (Not a Number).

JS: Conversión numérica

JS

Si utilizamos **parseInt()** con dos parámetros, donde el primero es el texto con el número y el segundo es la base numérica del número, se realiza la conversión de tipo respetando la base elegida:

```
Number.parseInt("11101", 2); // 29 en binario  
Number.parseInt("31", 8); // 25 en octal  
Number.parseInt("FF", 16); // 255 en hexadecimal
```

```
parseInt (11101, 2 (binario)) 29  
parseInt (31, 8 (octal)) 25  
parseInt (FF, 16 (hexadecimal)) 255
```

Esta modalidad de **parseInt()** se utiliza para pasar a base decimal un número que se encuentra en otra base (binario, octal, hexadecimal, etc.) **parseFloat()** funciona exactamente igual, pero en lugar de operar con números enteros opera con números en coma flotante.

JS: Operadores aritméticos y de asignación

JS

El operador de asignación (=) le otorga un valor a una variable y se coloca entre la variable y el valor a asignar.

```
var x = 10;
```

Los operadores aritméticos que vemos a la derecha se utilizan para realizar operaciones aritméticas en números:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
**	Exponenciación
/	División
%	Módulo: resto de dividir
++	Incremento
--	Decremento

JS: Concatenación

JS

Los operadores + y += también se pueden utilizar para agregar (concatenar) cadenas. En este contexto, el operador + se denomina operador de concatenación.

```
var txt1 = "Juan";  
var txt2 = "Pablo";  
var txt3 = txt1 + " " + txt2;  
console.log(txt3);
```

Juan Pablo

```
var txt4 = "Bienvenidos ";  
txt4 += "a Javascript";  
console.log(txt4);
```

Bienvenidos a Javascript

Agregar dos números devolverá la suma, pero agregar un número y una cadena devolverá una cadena.

JS: Prompt()

JS

La función prompt es un método del objeto Window. Se utiliza para solicitarle al usuario que ingrese datos por medio del teclado. Recibe dos parámetros: el mensaje que se muestra en la ventana y el valor inicial del área de texto. Su sintaxis es:
variable = prompt(mensaje, valor inicial)

```
<script>
  var nombre = prompt ("Ingrese su nombre", "")
  document.write( "Hola " + nombre)
</script>
```

127.0.0.1:5500 dice

Ingrese su nombre

Cancelar


Aceptar

JS: Document.write()

JS

`document.write()` nos permite escribir directamente dentro del propio documento HTML.

```
<html>
  <head>
    <title>Título de la página</title>
    <script>
      document.write("Hola mundo (HTML)");
    </script>
  </head>
  <body>
  </body>
</html>
```



Hola mundo (HTML)

JS: Artículos de interés

JS

Documentación extra:

- [¿Qué es JavaScript?](#)
- [¿Qué es EcmaScript?](#)
- [¿Debo usar “;” en Javascript?](#)
- [Tipos de datos en JavaScript](#)
- [Variables en JavaScript](#)
- [El objeto Number en JavaScript](#)
- Métodos del objeto Math en [Developer Mozilla](#), [W3Schools](#) y en [LenguajeJS](#)

¡Vamos a la práctica!





Ejercicios Prácticos



Opativos | No entregables

Operaciones con Variables y Tipos de Datos

Crear un programa que reciba dos números como entrada, realice varias operaciones aritméticas con ellos y muestre los resultados en la consola del navegador. Además, se deberá verificar si el resultado de la suma de ambos números es mayor o menor que un valor dado.

Tips:

1. **Validación de entradas:** asegurate de que los usuarios ingresen números válidos. Utilizá `isNaN()` para verificar que las entradas no sean texto u otros valores no numéricos.
2. **Uso de `parseFloat()` y `parseInt()`:** dependiendo del ejercicio, recomendá que utilicen `parseFloat()` si los números pueden tener decimales, o `parseInt()` si solo aceptan números enteros.
3. **Descomposición del problema:** recordá que podés dividir el problema en partes más pequeñas. Primero capturá los números, luego realizá las operaciones, y por último verificá los resultados.
4. **Consola del navegador:** mostrá los resultados usando `console.log()` para que los estudiantes puedan ver los cálculos en la consola del navegador. Es una excelente herramienta para debuggear.



Ejercicios Prácticos



Optativos | No entregables

Concatenación y Conversión de Tipos de Datos

Crear un programa que reciba el nombre y la edad de una persona, los concatene en una frase y luego convierta la edad de string a número para verificar si la persona es mayor de edad.

Tips:

1. **Validación de edad:** asegurate de que la edad ingresada sea un número válido antes de realizar cualquier operación. Usá `isNaN()` para evitar errores cuando el usuario ingresa texto o un valor vacío en lugar de un número.
2. **Concatenación de cadenas:** recordá que podés concatenar textos fácilmente con el operador `+`. Experimentá con diferentes formas de concatenar los valores para personalizar el mensaje de salida.
3. **Conversión de tipos:** usá `parseInt()` o `Number()` para convertir la edad de un string a un número, lo que es clave para realizar comparaciones o cálculos matemáticos.
4. **Mensajes claros:** asegurate de mostrar mensajes claros en la consola. Esto ayuda tanto al usuario como a vos mismo a entender si el programa está funcionando como esperabas.
5. **Pruebas con datos diferentes:** probá el programa con diferentes nombres y edades (incluyendo casos límite como menores de edad o números cercanos a 18) para verificar que la lógica del programa sea robusta.