



FORMACIÓN PROFESIONAL  
MONTECASTELO

# Sintaxis básica de JavaScript

Desarrollo Web en Entorno Cliente

Ciclo Superior de Desarrollo de Aplicaciones Web

2023/2024

# JavaScript

- JavaScript (JS) es un lenguaje ejecutado por el navegador web como parte del renderizado de una página web.
- El navegador interpreta el marcado HTML línea a línea y procesa el código JS tan pronto lo encuentre.
- Existen dos modos fundamentales de insertar código JS en una página HTML: mediante código en línea o código externo.



# Código JavaScript en línea

- Se puede insertar JavaScript en HTML escribiendo el código directamente dentro de una etiqueta `<script> </script>`.
- Este tipo de código JS se considera en general una mala práctica y no se recomienda.
- Es empleado a menudo con fines de demostración para evitar manejar archivos separados.



# Código JavaScript en línea

- Se puede agregar el código JS tanto dentro de la etiqueta `<head>` como dentro de la etiqueta `<body>`.
- Por lo general se recomienda agregarlo en la etiqueta `<head>` para que permanezca separado del contenido del archivo HTML.



# Código JS externo

- Se puede hacer referencia a archivos externos de JS desde documentos HTML, de modo análogo a como se hace en CSS.
- De este modo no se incluye código JS dentro de la etiqueta `<script>`, sino que se indica en el atributo `src` la ruta al fichero que lo contiene.

```
<script src="scripts/main.js"></script>
```

- En este caso la etiqueta `<script>` se sitúa dentro de la etiqueta `<head>`.



# Ventajas de código JS externo

- Cuando los códigos HTML y JavaScript están separados, se cumple el principio de diseño de separación, favoreciendo la sostenibilidad y reutilización del código.
- Se facilita la legibilidad y el mantenimiento del código.
- Los archivos JS en caché mejoran el rendimiento general del sitio web al disminuir el tiempo que tardan las páginas en cargarse.



# Variables

- La palabra `let` permite definir una variable en JS.
- La palabra `const` permite definir un dato cuyo valor no cambia durante la ejecución de un programa.
- La palabra `var` establece el modo en el que se definen las variables en versiones de JS anteriores a la 2015 (JS ES6).



# Variables

- En JavaScript no se indica el tipo de la variable cuando la declaramos.
- Para obtener el tipo de una variable podemos utilizar el operador `typeof`.

```
let m = 1, n = 2;
```





# Variables

- No es obligatorio inicializar las variables (el valor que almacena es undefined).
- Las variables no inicializadas no se podrán utilizar hasta que reciban un valor, ya que el lenguaje no les asigna un valor por defecto.
- Es recomendable terminar cada sentencia con el carácter de punto y coma.



# Tipos de datos primitivos

- Los siguientes son los principales tipos de datos primitivos de JS:
  - `boolean`: valor binario que puede tomar el valor `true` o `false`.
  - `number`: representa valores numéricos de cualquier tipo.
  - `string`: representa texto como una secuencia de caracteres.
  - `object`: representa una estructura de datos.



# Valores vacíos

- Existen dos valores especiales que a efectos prácticos pueden considerarse intercambiables:
  - `undefined`: representa un valor no asignado.
  - `null`: representa un valor asignado como vacío o desconocido.



# Valores numéricos especiales

- Existen también dos valores numéricos que son considerados especiales.
- `Infinity` y `-Infinity` representan el infinito matemático.
- `NaN` (*Not a Number*) utilizado para representar resultados que no tienen sentido numérico, como una división entre 0.



# Nombres de variables

- Pueden contener cifras, pero no al comienzo del nombre.
- Pueden incluir el signo del dólar (\$) o guiones bajos ( \_ ), pero ningún otro signo de puntuación o signos especiales (como espacios).
- No pueden coincidir con palabras clave o palabras reservadas por el lenguaje.
- Es recomendable emplear el siguiente formato para el nombrado de variables:

`unaVariableUtil`



# Operadores aritméticos

- Los operadores aritméticos toman los valores numéricos (tanto literales como variables) de sus operandos y devuelven un único resultado numérico.



# Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto de división)
++	Incremento
--	Decremento
-	Cambio de signo
**	Exponenciación



# Operadores de comparación

- Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (`true`) o falsa (`false`)





# Operadores de comparación

Operador	Operación
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
==	Igualdad
===	Igualdad estricta
!=	Desigualdad
!==	Desigualdad estricta



# Comparación de igualdad

- Una comparación de igualdad estricta (`===`) sólo es verdadera si los operandos son del mismo tipo y los contenidos coinciden.
- La comparación de igualdad (`==`) convierte los operandos al mismo tipo antes de hacer la comparación.



# Comparación de igualdad

- Ejemplo de comparación de igualdad

```
12 == "12"; // devuelve true
```

```
12 === "12"; // devuelve false
```



# Operadores lógicos

- Permiten generar expresiones lógicas compuestas conjuntamente con valores booleanos.

Operador	Operación
&&	AND
	OR
!	NOT



# Operadores bit a bit

- Los operadores bit a bit tratan a sus operandos como un conjunto de 32 bits (ceros y unos), en vez de como números decimales, hexadecimales u octales.
- Realizan sus operaciones en dicha representación binaria, pero devuelven un valor numérico estándar.

Operador	Operación
&	AND
	OR
~	NOT
^	XOR



# Operadores de asignación

- Un operador de asignación asigna un valor al operando de la izquierda en función del valor del operando de la derecha.

Operador	Operación
=	Asignación estándar de derecha a izquierda
+=	Suma y asignación
-=	Resta y asignación
*=	Multiplicación y asignación
/=	División y asignación
%=	Módulo y división



# Operadores de cadenas

- Los operadores de comparación pueden ser usados en cadenas de caracteres. La comparación se basa en el orden alfabético.
- El operador de concatenación (+) une dos cadenas, devolviendo otra cadena resultante de la unión de los dos operandos.



# Operador coma

- El operador coma (,) evalúa ambos operandos y retorna el valor del último.

```
var m, n;  
m=1, n=10; // devuelve 10
```





# Operadores typeof e instanceof

- El operador `typeof` devuelve una cadena de caracteres indicando el tipo del operando evaluado.
- El operador `instanceof` devuelve `true` si el objeto especificado como primer operando es del tipo de objeto especificado como segundo parámetro.



# Comentarios

- Un comentario es una parte del código que debe ser ignorada por el intérprete del lenguaje.
- En JS existen dos tipos de comentarios.
- Los comentarios de una línea comienzan por `//`.
- Los comentarios de varias líneas se escriben entre `/*` y `*/`.



# Condicionales

- Las sentencias condicionales permiten que un fragmento de código se ejecute una vez si una condición se evalúa al valor true.
- JS soporta las siguientes condicionales:
  - Sentencias if-else
  - Sentencias switch
  - Operador condicional
- Las condicionales en JS comparten sintaxis con Java.



# Bucles

- Las sentencias condicionales permiten que un fragmento de código se ejecute una vez si una condición se evalúa al valor true.
- En cambio, los bucles permiten que un fragmento de código se ejecute repetidas veces mientras una determinada condición se mantenga al valor true.
- JS soporta los siguientes bucles:
  - Bucles for
  - Bucles while
  - Bucles do-while
- Los bucles en JS comparten sintaxis con Java.



# Funciones

- Una función permite agrupar un bloque de código para que pueda ser reutilizado.
- Las funciones permiten estructurar programas largos, reducir la repetición de código, asociar un nombre a cada bloque de código y aislar los distintos bloques entre sí.



# Funciones

- Las funciones se definen con la siguiente sintaxis:  

```
function saludo (persona) {  
  let cad = "Hola " + persona;  
  return cad;  
}
```
- El código situado entre las llaves { } se denomina cuerpo de la función.
- Si la función debe devolver un valor, se emplea la palabra clave return.
- Entre paréntesis después del nombre de la función se sitúan los nombres de los parámetros de la función separados por comas.



# Parámetros y argumentos

- Parámetros son los nombres que aparecen en la definición de una función.
- Argumentos son los valores que pasamos a (y que recibe) una función.
- Una función puede tener cero o más parámetros.



# Llamada a funciones

- Para llamar una función se escribe su nombre seguido de paréntesis.
- De ser necesarios, entre paréntesis se pasan los argumentos.

```
var cadena = saludo("Pedro");
```





# Llamada a funciones

- Si el nombre de la función se escribe sin paréntesis, se puede asignar la función a una variable, en lugar del valor devuelto por la función.

```
var cadena = saludo("Pedro");  
// cadena contiene el valor devuelto por saludo  
var f = saludo;  
// f contiene la función saludo
```



# Funciones anónimas

- Las funciones anónimas son aquéllas que no tienen nombre.
- Se invocan a través del nombre de una variable:

```
var saludo = function (persona) {  
  var cad = "Hola" + persona;  
  return cad;  
}  
var cadena = saludo("Pedro");
```



# Funciones flecha

- Un modo alternativo de definir funciones emplea una flecha ( $=>$ ) en lugar de la palabra reservada 'function'
- La flecha ( $=>$ ) se sitúa entre los parámetros y el cuerpo de la función
- Simboliza que a partir de una entrada (los parámetros) se genera una salida (el cuerpo de la función)



# Funciones flecha

- Ejemplo

```
const suma = (a, b) => {  
  let resultado = a + b;  
  return resultado;  
};
```



# El constructor Function

- Cada función de JavaScript es en realidad un objeto de la clase Function.
- Por tanto una función puede ser creada mediante el constructor de la clase Function, que recibe una lista de cadenas con los nombres de los parámetros, seguidos del código de la función:

```
var miFuncion = new Function ("a", "b", "return a*b;");  
var resultado = miFuncion(3,6); // devuelve 18
```



# Ámbito de las variables

- En relación con el ámbito (visibilidad o *scope*) JS es un lenguaje de programación con las siguientes características:
  - Ámbito global como predeterminado.
  - Paso por referencia también de forma predeterminada.



# Ámbito de las variables

- Las declaraciones mediante `let` y `const` tienen ámbito de bloque, mientras que las declaraciones mediante `var` tienen ámbito de función.
- Un bloque en JavaScript se puede entender como el contenido encerrado por dos llaves `{ }`, ya sean definiciones de funciones o bloques `if`, `while`, `for` y bucles similares.



# Ámbito de var

- La declaración con `var` define una variable en el ámbito local (función) actual.
- Por ese motivo se dice que las declaraciones mediante `var` tienen ámbito de función.
- Si la variable es declarada fuera de una función, la variable será una variable global.





# Ámbito de `let` y `const`

- Si una variable es declarada con `let` en el ámbito global o en el de una función, la variable pertenecerá al ámbito global o al ámbito de la función respectivamente, de forma similar a como ocurría con `var`.
- Pero si declaramos una variable con `let` dentro de un bloque que a su vez está dentro de una función, la variable pertenece solo a ese bloque. Fuera del bloque donde se declara con `let`, la variable no está definida.
- Por ese motivo se dice que las declaraciones mediante `let` y `const` tienen ámbito de bloque.



# Ámbito de las variables

- Debido a los comportamientos anteriormente descritos, la tendencia en JS es tomar `let` como la forma predeterminada de declarar variables, dado que el ámbito más específico evita la sobreescritura de variables de forma accidental (`let` declara variables sin interferir con el ámbito superior).



# Referencias

- [JavaScript en w3schools](#)
- [JavaScript en Mozilla Developer Network](#)





FORMACIÓN PROFESIONAL  
MONTECASTELO