

Tema 2 - JavaScript



Fundamentos de JavaScript

Desarrollo web en entorno cliente
IES Pere Maria Orts I Bosch



Index

Introducción.....	3
Editores y herramientas para programar.....	4
La consola del navegador.....	4
Editores de escritorio.....	4
Editores web.....	5
Javascript, primeros pasos.....	6
Integrando JavaScript con HTML.....	6
Variables.....	7
Constantes.....	8
Funciones.....	8
Funciones lambda (o arrow functions).....	9
Parámetros por defecto.....	10
Estructuras condicionales.....	10
Bucles.....	11
Tipos de datos básicos.....	12
Ámbito de las variables.....	16
Operadores.....	17

Introducción

En esta primera semana, vamos a aprender los fundamentos de JavaScript en cuanto a sintaxis se refiere. Este documento será especialmente importante para quién no tenga base previa del lenguaje (pero sí de programación con otros lenguajes, ya que este no es un curso sobre fundamentos de programación).

JavaScript es un lenguaje interpretado, ejecutado por un intérprete normalmente integrado en un navegador web (pero no sólo en ese contexto). Lo que se conoce como JavaScript es en realidad una implementación de ECMAScript, el estándar que define las características de dicho lenguaje. La versión más reciente de ECMAScript specification es ES2018 (Junio del 2018). En la versión ES2015 (Junio del 2015, también conocida como ES6) se introdujeron numerosos cambios en el lenguaje y una modernización necesaria después de pasar muchos años sin apenas cambios desde la primera versión del lenguaje en 1997. La versión anterior fue ES5 (Diciembre. 2009), y ES5.1 (Junio 2011). [Aquí](#) puedes ver la compatibilidad de los distintos navegadores con las diferentes versiones.

JavaScript se verá durante las 5 primeras semanas del curso. Posteriormente veremos herramientas muy útiles para gestionar proyectos JavaScript como NPM (Node Package Manager) y WebPack. Finalmente estaremos preparados para conocer TypeScript, que se puede definir como un superconjunto de JavaScript que añade tipado a las variables y métodos, junto con otras características extra.

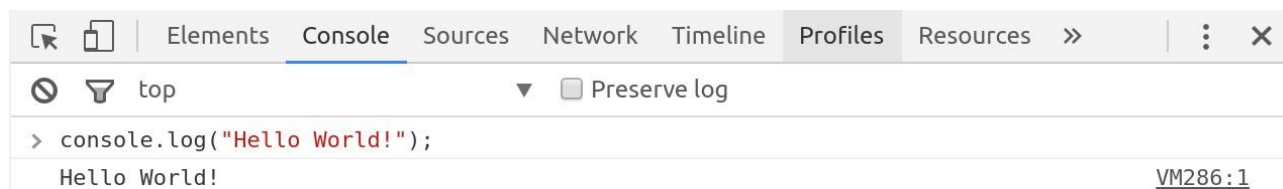
[TypeScript](#) está siendo cada vez más popular, y cada vez lo será más, ya que se utiliza por defecto en frameworks tan populares como Angular. Cada vez más gente está optando por trabajar con esta variante del lenguaje por las ventajas que supone en cuanto a desarrollo gracias a características como: programación orientada a objetos más completa, autocompletado y detección temprana de errores, tal y como veremos a final del curso.

Editores y herramientas para programar

Para realizar este curso, puedes elegir cualquiera de los editores o IDE que prefieras. Sin embargo, lo idóneo sería que usaras el editor usado para los ejemplos (Visual Studio Code), porque todas las instrucciones serán dadas teniendo en cuenta este editor. Hay muchas opciones posibles, y al principio no es demasiado importante qué editor elijas.

La consola del navegador

Abre tu navegador (Te recomiendo Chrome or Firefox) y pulsa F12 para mostrar las herramientas de desarrollador que vienen integradas. Ve a la pestaña de Consola, y desde ahí puedes ejecutar instrucciones en JavaScript y visualizar el resultado de forma inmediata. Esta opción es una buena opción para *testear* pequeñas partes de código.



Por curiosidad, prueba este trozo de código donde podemos ver uno de los famosos bugs de JavaScript (la precisión decimal en algunas operaciones matemáticas)

```
> console.log(5.1 + 3.3);  
8.399999999999999
```

Editores de escritorio

Muchos editores de código e IDEs soportan las últimas versiones de la sintaxis de JavaScript, mientras que algunos de ellos soportan el código completo, integrándolo con jQuery, Angular, etc. Pudes usar aque con el que más cómodo te sientas (Visual Studio, Netbeans, Webstorm, Atom, Sublime, Kate, Notepad++, ...), pero todos los ejemplos que ponga serán en Visual Studio Code, porque se integra muy bien con JavaScript, Node, Typescript, y Angular.

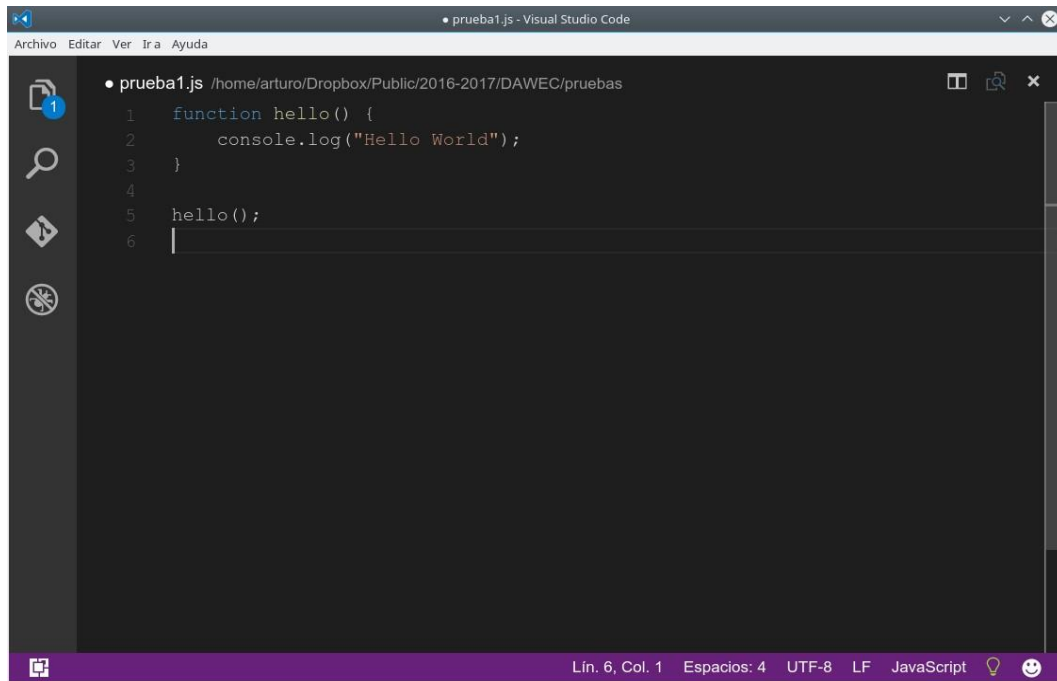
Visual Studio Code

Este editor (con algunas características de IDE) es desarrollado por Microsoft y de código abierto. Se encuentra disponible en Windows, Mac y GNU/Linux. Ha sido desarrollado utilizando [Electron](#), que es un framework que usa Chromium y Node.js para desarrollar aplicaciones de escritorio utilizando HTML, CSS y JavaScript. Es decir, es una aplicación web de escritorio.

Visual Studio Code es un editor ligero, soporta resaltado de sintaxis, y autocompletado de código para JavaScript y TypeScript entre otros. También se

integra muy bien con Angular y otros frameworks. Otros editores que integran [TypeScript](#) son: Visual Studio, Atom, Sublime Text, Webstorm, Emacs o Vim (muchos de ellos mediante plugins). Visual Studio Code, es el editor que os recomiendo para este curso.

Puedes descargarlo de: <https://code.visualstudio.com/Download>



Editores web

Podemos usar editores web, al menos para probar código que no sea muy complejo o muy grande. Esta opción es bastante buena para probar código fácilmente sin necesidad de tener instalado un editor, y también para compartir código de forma rápida con otras personas. La parte mala es que no tienen muchas de las ventajas que presentan los editores de escritorio como autocompletado, o plugins, por ejemplo.

Dos famosos editores web son Fiddle (<https://jsfiddle.net/>) y Plunker (<https://plnkr.co/>). Puedes guardar tus proyectos y continuar el desarrollo más tarde en cualquier dispositivo.

Javascript, primeros pasos

Es probable que tengas ciertos conocimientos de Javascript, o incluso experiencia en el desarrollo de páginas web. Sin embargo, no es una mala opción hacer una revisión sobre algunos aspectos del lenguaje, esto ayudará a consolidar los conocimientos que tengas e incluso puedes aprender algo que no conocías, como las mejoras introducidas en la versión ES2015.

Integrando JavaScript con HTML

Para integrar el código JavaScript en nuestro HTML, necesitamos usar la etiqueta `<script>`. El sitio recomendado para poner la etiqueta es justo antes de cerrar la etiqueta `</html>`, para que algunos navegadores puedan cargar y construir el DOM antes de procesar el código JavaScript, de otro modo, el navegador se bloqueará el renderizado de la página hasta que se haya procesado todo el código JS.

¿Dónde podemos poner el código JS?:

Dentro de la etiqueta `<script>` (no recomendado)

```
<!DOCTYPE>
<html>
  <head>
    <title>Ejemplo JS</title>
  </head>
  <body>
    <p>Hola Mundo!</p>
    <script>
      console.log("Hola Mundo!");
    </script>
  </body>
</html>
```

En un archivo separado (recomendado)

Archivo: ejemplo1.html

```
<!DOCTYPE>
<html>
  <head>
    <title>Ejemplo JS</title>
  </head>
  <body>
    <p>Hola Mundo!</p>
    <script src="ejemplo1.js"></script>
  </body>
</html>
```

Archivo: ejemplo1.js

```
console.log("Hola Mundo!");
```

`console.log()` se utiliza a escribir por la consola del navegador aquello que

nosotros le pasamos (F12 para abrir las herramientas de desarrollador y ver el resultado). Puedes usar el método `console.error()` para mostrar los errores.

La etiqueta `<noscript>` se utiliza para poner código HTML que será renderizado sólo cuando el navegador no soporte JavaScript o cuando haya sido desactivado. Esta etiqueta es muy útil para decirle al usuario que la web necesita tener JavaScript activado para funcionar correctamente, por ejemplo.

```
<!DOCTYPE>
<html>
  <head>
    <title>Ejemplo JS</title>
  </head>
  <body>
    <p>Hola Mundo!</p>
    <noscript>
      <h1>JavaScript no está activado. Por favor, actívalo o la aplicación
web no funcionará correctamente.</h1>
    </script>
  </body>
</html>
```

Variables

Puedes declarar una variable usando la palabra reservada `let` (también puedes usar `var`, pero no se recomienda desde la versión ES2015). El nombre de la variable deberá ser con el formato CamelCase, es decir, la primera letra en minúsculas, también puede empezar por subrayado (`_nombre`) o por dólar (`$nombre`). En JS las variables no tienen un tipo de dato explícito. El tipo puede cambiar internamente dependiendo de cual sea el valor que se le asigne. Esto significa que puedes asignar un string, y posteriormente un número.

```
let v1 = "Hola Mundo!";
console.log(typeof v1); // Imprime -> string

v1 = 123;
console.log(typeof v1); // Imprime -> number
```

¿Qué pasa si declaramos una variable pero no le asignamos un valor?. Hasta que no se le asigne un valor, tendrá un tipo especial conocido como `undefined`. Nota: Este valor es diferente de `null` (el cual se considera un valor).

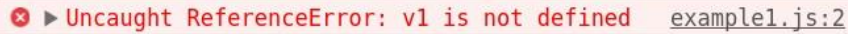
```
let v1;
console.log(typeof v1); // Imprime -> undefined
if (v1 === undefined) { // (!v1) or (typeof v1 === "undefined") también funciona
  console.log("Has olvidado darle valor a v1");
}
```

¿Qué ocurre si se nos olvida poner `let` o `var`?. JavaScript declarará esa variable como **global**. Esto NO es lo recomendado porque las variables globales son peligrosas. Por tanto, es recomendable que usemos siempre `let` la primera vez que vayamos a necesitar una variable local.

Para evitar que se nos olvide declarar la variable, podemos usar una declaración

especial: 'use strict' al comienzo de nuestro archivo JS. De este modo, no vamos a poder declarar variables globales omitiendo la palabra reservada let.

```
'use strict';  
v1 = "Hola Mundo";
```



Uncaught ReferenceError: v1 is not defined example1.js:2

Constantes

Cuando a lo largo de una función o bloque, una variable no va a cambiar de valor, o cuando queremos definir un valor global inmutable (por ejemplo el número PI), se recomienda declararla como constante con la palabra reservada `const` en lugar de usar `let`. En el caso de las constantes globales se recomienda usar mayúsculas.

```
'use strict';  
const MY_CONST=10;  
MY_CONST=200; → Uncaught TypeError: Assignment to constant variable.
```

Funciones

En JavaScript, declaramos funciones usando la palabra reservada `function` antes del nombre de la función. Los argumentos que le pasaremos a la función van dentro del paréntesis tras el nombre de la función (recuerda que no hay tipos de variable en JS). Una vez definida la función, entre llaves declaramos el cuerpo de la misma. El nombre de las funciones (como el de las variables) debe escribirse en formato CamelCase con la primera letra en minúsculas.

```
function sayHello(name) {  
  console.log("Hello " + name);  
}  
  
sayHello("Tom"); // Imprime "Hello Tom"
```

No necesitas tener la función declarada antes de llamarla, esto es debido a que el intérprete de JavaScript primero procesa las declaraciones de variables y funciones y después ejecuta el resto del código.

Podemos llamar a una función enviándole más o menos parámetros de los establecidos en la declaración. Si le enviamos más parámetros, los sobrantes serán ignorados y si le enviamos menos, a los no recibidos se les asignará `undefined`.

```
sayHello(); // Imprime "Hello undefined"
```

Retorno de valores

Podemos usar la palabra reservada `return` para devolver un valor en una función. Si intentamos obtener un valor de una función que no devuelve nada, obtendremos `undefined`.

```
function totalPrice(priceUnit, units) {  
  return priceUnit * units;  
}
```



```
let total = totalPrice(5.95, 6);  
console.log(total); // Imprime 35.7
```

Funciones anónimas

La forma de declarar una función anónima es no asignarle ningún nombre. Podemos asignar dicha función como valor a una variable, ya que es un tipo de valor (como puede ser un *string* o número), por tanto, puede ser asignada a (o referenciada desde) múltiples variables. Se utiliza igual que una función clásica.

```
let totalPrice = function(priceUnit, units) {  
  return priceUnit * units;  
}  
  
console.log(typeof totalPrice); // Imprime "function" (tipo de la variable totalPrice)  
  
console.log(totalPrice(5.95, 6)); // Imprime 35.7  
let getTotal = totalPrice; // Referenciamos a la misma función desde la variable getTotal  
console.log(getTotal(5.95, 6)); // Imprime 35.7. También funciona
```

Funciones lambda (o arrow functions)

Una de las funcionalidades más importantes que se añadió en ES2015 fue la posibilidad de usar las funciones lambda (o flecha). Otros lenguajes como C#, Java, etc. también las soportan. Estas expresiones ofrecen la posibilidad de crear funciones anónimas pero con algunas ventajas.

Vamos a ver las diferencias que tiene por creando dos funciones equivalentes (una anónima y otra lambda que hacen lo mismo):

```
let sum = function(num1, num2) {  
  return num1 + num2;  
}  
console.log(sum(12,5)); // Imprime 17  
  
let sum = (num1, num2) => num1 + num2;  
console.log(sum(12,5)); // Imprime 17
```

Cuando declaramos una función lambda, la palabra reservada *function* no se usa. Si sólo se recibe un parámetro, los paréntesis pueden ser omitidos. Después de los parámetros debe ir una flecha (*=>*), y el contenido de la función.

```
let square = num => num * num;  
console.log(square(3)); // Imprime 9
```

Si sólo hay una instrucción dentro de la función lambda, podemos omitir las llaves '{}', y debemos omitir la palabra reservada *return* ya que lo hace de forma implícita (devuelve el resultado de esa instrucción). Si hay más de una instrucción, usamos las llaves y se comporta como una función normal y por tanto, si devuelve algo, debemos usar la palabra reservada *return*.

```
let sumInterest = (price, percentage) => {  
  let interest = price * percentage / 100;  
  return price + interest;  
}  
console.log(sumInterest(200,15)); // Imprime 230
```

La diferencia más importante entre ambos tipos de funciones es el comportamiento de la palabra reservada `this`. Pero eso lo veremos en el tercer bloque del curso (Programación orientada a objetos).

Parámetros por defecto

Si un parámetro se declara en una función y no se pasa cuando la llamamos, se establece su valor como `undefined`.

```
function Persona(nombre) {  
  this.nombre = nombre;  
  
  this.diHola = function() {  
    console.log("Hola! Soy " + this.nombre);  
  }  
}  
let p = new Persona();  
p.diHola(); // Imprime "Hola! Soy undefined"
```

Una solución usada para establecer un valor por defecto era usar el operador `'||'` (or), de forma que si se evalúa como `undefined` (false), se le asigna otro valor.

```
function Persona(nombre) {  
  this.nombre = nombre || "Anónimo";  
  ...  
}
```

Sin embargo, en ES2015 tenemos la opción de establecer un valor por defecto.

```
function Persona( nombre = "Anónimo") {  
  this.nombre = nombre;  
  ...  
}
```

También podemos asignarle un valor por defecto basado en una expresión.

```
function getPrecioTotal(precio, impuesto = precio * 0.07) {  
  return precio + impuesto;  
}  
console.log(getPrecioTotal(100)); // Imprime 107
```

Estructuras condicionales

La estructura `if` se comporta como en la mayoría de los lenguajes de programación. Lo que hace es evaluar una condición lógica, devolviendo un booleano

como resultado y si es cierta, ejecuta el código que se encuentra dentro del bloque if. De forma optativa podemos añadir el bloque else if, y un bloque else.

```
let price = 65;

if(price < 50) {
  console.log("Esto es barato!");
} else if (price < 100) {
  console.log("Esto no es barato...");
} else {
  console.log("Esto es caro!");
}
```

La estructura switch tiene un comportamiento similar al de otros lenguajes de programación. Como sabemos, se evalúa una variable y se ejecuta el bloque correspondiente al valor que tiene (puede ser número, string,..). Normalmente, se necesita poner la instrucción break al final de cada bloque, ya que de no ponerlo continuaría ejecutando las instrucciones que haya en el siguiente bloque. Un ejemplo donde dos valores ejecutarían el mismo bloque de código es el siguiente:

```
let userType = 1;

switch(userType) {
  case 1:
  case 2: // Tipos 1 y 2 entran aquí
    console.log("Puedes acceder a esta zona");
    break;
  case 3:
    console.log("No tienes permisos para acceder aquí");
    break;
  default: // Ninguno de los anteriores
    console.error("Tipo de usuario erróneo!");
}
```

En JavaScript (a diferencia de muchos otros lenguajes), puedes hacer que el switch se comporte como un if. Esto se hace evaluando un booleano (normalmente true) en lugar de otro tipo de valor, de forma que el case se evalúan condiciones:

```
let age = 12;

switch(true) {
  case age < 18:
    console.log("Eres muy joven para entrar");
    break;
  case age < 65:
    console.log("Puedes entrar");
    break;
  default:
    console.log("Eres muy mayor para entrar");
}
```

Bucles

Tenemos el típico bucle while que evalúa una condición y se repite una y otra vez hasta que la condición sea falsa (o si la condición es falsa desde un primer

momento, no entra a realizar el bloque de instrucciones que contiene).

```
let value = 1;
```

```
while (value <= 5) { // Imprime 1 2 3 4 5
  console.log(value++);
}
```

Además de `while`, podemos usar `do..while`. La comprobación de la condición se realiza al final del bloque de instrucciones, por lo tanto, siempre se va a ejecutar el código al menos una vez.

```
let value = 1;
```

```
do { // Imprime 1 2 3 4 5
  console.log(value++);
} while (value <= 5);
```

El bucle `for` funciona igual que en otros lenguajes de programación. Inicializamos uno o más valores, establecemos la condición de finalización y el tercer apartado es para establecer el incremento o decremento.

```
let limit = 5;
```

```
for (let i = 1; i <= limit; i++) { // Imprime 1 2 3 4 5
  console.log(i);
}
```

Como sabrás, puedes inicializar una o más variables y también ejecutar varias instrucciones en cada iteración separándolas con comas.

```
let limit = 5;
```

```
for (let i = 1, j = limit; i <= limit && j > 0; i++, j--) {
  console.log(i + " - " + j);
}
/* Imprime
1 - 5
2 - 4
3 - 3
4 - 2
5 - 1
*/
```

Dentro de un bucle, podemos usar las instrucciones de `break` y `continue`. La primera de ellas saldrá del bucle de forma inmediata tras ejecutarse, y la segunda, irá a la siguiente iteración saltándose el resto de instrucciones de la iteración actual (ejecuta el correspondiente incremento si estamos dentro de un bucle `for`).

Tipos de datos básicos

Números

En JS no hay diferencia entre números enteros y decimales (`float`, `double`). El tipo

de dato para cualquier número es number.

```
console.log(typeof 3); // Imprime number
console.log(typeof 3.56); // Imprime number
```

Además puedes imprimir números en notación científica (exponencial):

```
let num = 3.2e-3; // 3.2*(10^-3)
console.log(num); // Imprime 0.0032
```

Los números son objetos

En JavaScript todo es un objeto, incluso los valores primitivos que hay en otros lenguajes (Java, C++, etc.). Por ejemplo, si ponemos un punto después de escribir un número, podemos acceder a algunos métodos o propiedades.

```
console.log(3.32924325.toFixed(2)); // Imprime 3.33
console.log(5435.45.toExponential()); // Imprime 5.43545e+3
console.log((3).toFixed(2)); // Imprime 3.00 (Un entero necesita estar dentro de un paréntesis para poder acceder a sus propiedades)
```

Existe también un objeto global del language llamado Number, donde podemos acceder a otras propiedades bastante útiles para trabajar con números.

```
console.log(Number.MIN_VALUE); // Imprime 5e-324 (El número más pequeño)
console.log(Number.MAX_VALUE); // Imprime 1.7976931348623157e+308 (El número más grande)
```

Hay también valores especiales para los números fuera de rango (Infinito y -Infinito). Podemos comparar si un número tiene uno de estos valores directamente usando === Infinity ó valor === -Infinity por ejemplo. Sin embargo, hay una función llamada isFinite(value) que nos devuelve falso cuando el valor es Infinity ó -Infinity.

```
console.log(Number.MAX_VALUE * 2); // Imprime Infinity
console.log(Number.POSITIVE_INFINITY); // Imprime Infinity
console.log(Number.NEGATIVE_INFINITY); // Imprime -Infinity
console.log(typeof Number.POSITIVE_INFINITY); // Imprime number
```

```
let number = Number.POSITIVE_INFINITY / 2; // Sigue siendo todavía infinito!!
if(isFinite(number)) { // Es igual que (number !== Infinity && number !== -Infinity)
  console.log("El número es " + number);
} else { // Enters here
  console.log("El número no es finito");
}
```

Operaciones con números

Podemos realizar las operaciones típicas (+, -, *, /, %, ...). Pero, ¿Qué ocurre si uno de los operandos no es un número?. Por ejemplo, si un número se pone entre comillas, es considerado un string. Cuando hacemos una operación numérica con valores que no son números, intenta transformar esos valores a números (*cast* implícito). Si no puede, nos devuelve un valor especial llamado NaN (Not a Number).

```
let a = 3;
```

```
let b = "asdf";  
let r1 = a * b; // b es "asdf", y no será transformado a número  
console.log(r1); // Imprime NaN  
  
let c;  
let r3 = a + c; // c es undefined, no será transformado a número  
console.log(r3); // Imprime NaN  
  
let d = "12";  
console.log(a * d); // Imprime 36. d puede ser transformado al número 12  
console.log(a + d); // Imprime 312. El operador + concatena si hay un string  
console.log(a + +d); // Imprime 15. El operador '+' delante de un valor lo transforma en numérico
```

Para comprobar si un número es NaN, se puede utilizar el método que devuelve un booleano (true si es NaN): `Number.isNaN(valor)`

undefined y null

En JavaScript cuando una variable (o parámetros de una función) han sido definida sin asignarle valor, se inicializa con el valor especial `undefined`.

No deberíamos confundir `undefined` con `null`. El segundo es un tipo de valor, que explícitamente asignas a una variable. Vamos a verlo mediante un ejemplo.

```
let value; // Value no ha sido asignada (undefined)  
console.log(typeof value); // Imprime undefined  
  
value = null;  
console.log(typeof value); // Imprime object
```

Como puedes ver, `null` es considerado un tipo de objeto (una referencia vacía). `undefined` es un valor especial que nos avisa que la variable no ha sido inicializada.

Boolean

En JS, los booleanos se representan en minúsculas (`true`, `false`). Puedes negarlos usando el operador `!` antes del valor.

Strings

Los valores string se representan dentro de 'comillas simples' o "comillas dobles". Podemos utilizar el operador `+` para concatenar cadenas.

```
let s1 = "Esto es un string";  
let s2 = 'Esto es otro string';  
  
console.log(s1 + " - " + s2); // Imprime: Esto es un string – Esto es otro string
```

Cuando el string se encuentra dentro de comillas dobles, podemos usar comillas simples dentro y viceversa. Sin embargo, si quieres poner comillas dobles dentro de una cadena declarada a su vez dentro de comillas dobles, necesitas *escaparlas* para no cerrar el string previo, ocurriría lo mismo si fueran comillas simples.


```
console.log("Hello 'World'"); // Imprime: Hello 'World'  
console.log('Hello \World'); // Imprime: Hello 'World'
```

```
console.log("Hello \"World\""); // Imprime: Hello "World"  
console.log('Hello "World"'); // Imprime: Hello "World"
```

Como en el caso de los números, los strings son objetos y tienen algunos métodos útiles que podemos utilizar. Todos estos métodos no modifican el valor de la variable original a menos que la reasignes.

```
let s1 = "Esto es un string";  
// Obtener la longitud del string  
console.log(s1.length); // Imprime 17  
  
// Obtener el carácter de una cierta posición del string (Empieza en 0)  
console.log(s1.charAt(0)); // Imprime "E"  
  
// Obtiene el índice de la primera ocurrencia  
console.log(s1.indexOf("s")); // Imprime 1  
  
// Obtiene el índice de su última ocurrencia  
console.log(s1.lastIndexOf("s")); // Imprime 11  
  
// Devuelve un array con todas las coincidencias en de una expresión regular  
console.log(s1.match(/.s/g)); // Imprime ["Es", "es", " s"]  
  
// Obtiene la posición de la primera ocurrencia de una expresión regular  
console.log(s1.search(/[aeiou]/)); // Imprime 3  
  
// Reemplaza la coincidencia de una expresión regular (o string) con un string (/g opcionalmente reemplaza todas)  
console.log(s1.replace(/i/g, "e")); // Imprime "Esto es un streng"  
  
// Devuelve un substring (posición inicial: incluida, posición final: no incluida)  
console.log(s1.slice(5, 7)); // Imprime "es"  
  
// Igual que slice  
console.log(s1.substring(5, 7)); // Imprime "es"  
  
// Como substring pero con una diferencia (posición inicial, número de caracteres desde la posición inicial)  
console.log(s1.substr(5, 7)); // Imprime "es un s"  
  
// Transforma en minúsculas, toLowerCase no funciona con caracteres especiales (ñ, á, é, ...)  
console.log(s1.toLocaleLowerCase()); // Imprime "esto es un string"  
  
// Transforma a mayúsculas  
console.log(s1.toLocaleUpperCase()); // Imprime "ESTO ES UN STRING"  
  
// Devuelve un string eliminando espacios, tabulaciones y saltos de línea del principio y final  
console.log(" String con espacios ".trim()); // Imprime "String con espacios"  
  
// Devuelve si una cadena empieza por una determinada subcadena  
console.log(s1.startsWith("Esto")); // Imprime true  
  
// Devuelve si la cadena acaba en la subcadena recibida  
console.log(s1.endsWith("string")); // Imprime true  
  
// Devuelve si la cadena contiene la subcadena recibida  
console.log(s1.includes("es")); // Imprime true
```



```
// Genera una nueva cadena resultado de repetir la cadena actual N veces  
console.log("la".repeat(6)); // Imprime "lalalalalala"
```

También, ahora los string soportan caracteres con más de dos bytes (4 caracteres hexadecimales), a veces llamados caracteres de plano astral en unicode (*astral plane*), se deben escribir entre llaves `\u{}`. Ejemplo:

```
let uString = "Unicode astral plane: \u{1f3c4}";  
console.log(uString); // Imprime "Unicode astral plane: 🏄" (icono del surfista)
```

Estos caracteres especiales devuelven un valor de dos caracteres cuando se mide la longitud de un string:

```
let surfer = "\u{1f3c4}"; // Un carácter: 🏄  
console.log(surfer.length); // Imprime 2
```

Sin embargo, si transformamos un string en un array (de caracteres), este será dividido correctamente (cada carácter en una posición del array):

```
let surfer2 = "\u{1f30a}\u{1f3c4}\u{1f40b}"; // TRES caracteres: 🏊🏄🏃  
console.log(surfer2.length); // Imprime 6  
console.log(Array.from(surfer2).length); // Imprime 3 (convertido en array correctamente)
```

Template literals (sustitución de variables y multilínea)

Desde ES2015, JavaScript soporta string multilínea con sustitución de variables. Ponemos el string entre caracteres ``` (backquote) en lugar de entre comillas simples o dobles. La variable (o cualquier expresión que devuelva un valor) va dentro de `${}` si se quiere sustituir por su valor.

```
let num = 13;
```

```
console.log(`Example of multi-line string  
the value of num is ${num}`);
```

```
Example of multi-line string  
the value of num is 13
```

Conversión de tipos

Puedes convertir un dato a string usando la función `String(value)`. Otra opción es concatenarlo con una cadena vacía, de forma que se fuerce la conversión

```
let num1 = 32;  
let num2 = 14;
```

```
// Cuando concatenamos un string, el otro operando es convertido a string  
console.log(String(32) + 14); // Imprime 3214  
console.log("" + 32 + 14); // Imprime 3214
```

Puedes convertir un dato en number usando la función `Number(value)`. Puedes también añadir el prefijo `+` antes de la variable para conseguir el mismo resultado.

```
let s1 = "32";  
let s2 = "14";
```

```
console.log(Number(s1) + Number(s2)); // Imprime 46  
console.log(+s1 + +s2); // Imprime 46
```

La conversión de un dato a booleano se hace usando la función `Boolean(value)`. Puedes añadir `!!` (doble negación), antes del valor para forzar la conversión. Estos valores equivalen a `false`: string vacío (`""`), `null`, `undefined`, `0`. Cualquier otro valor debería devolver `true`.

```
let v = null;  
let s = "Hello";
```

```
console.log(Boolean(v)); // Imprime false  
console.log(!!s); // Imprime true
```

Ámbito de las variables

Variables globales

Cuando se declara una variable en el bloque principal (fuera de cualquier función), ésta es creada como global. Las variables que no son declaradas con la palabra reservada `let` son también globales (a menos que estemos usando el `strict mode`, que no permite hacer esto). Cuando ejecutamos JavaScript en un navegador, el objeto global `window` mantiene todas las variables globales (de hecho es implícito por defecto: `window.variable` → `variable`).

```
let global = "Hello";  
  
function cambiaGlobal() {  
  global = "GoodBye";  
}  
  
cambiaGlobal();  
console.log(global); // Imprime "GoodBye"  
console.log(window.global); // Imprime "GoodBye"
```

Intenta declarar una variable global dentro de una función usando `strict mode`. Si no usas `strict` (Es recomendable que lo uses), podrías crear una variable global así.

```
'use strict';  
  
function changeGlobal() {  
  global = "GoodBye";  
}  
  
changeGlobal(); // Error → Uncaught ReferenceError: global is not defined
```

Variables definidas en funciones

Todas las variables que se declaran dentro de una función son locales.

```
function setPerson() {  
  let person = "Peter";  
}
```

```
setPerson();  
console.log(person); // Error → Uncaught ReferenceError: person is not defined
```

Si una variable global con el mismo nombre existe, la variable local no actualizará el valor de la variable global.

```
function setPerson() {  
  let person = "Peter";  
}  
  
let person = "John";  
setPerson();  
console.log(person); // Imprime John
```

Operadores

Suma '+'

Este operador puede usarse para sumar números o concatenar cadenas. Pero, ¿Qué ocurre si intentamos sumar un número con un string, o algo que no sea un número o string?. Vamos a ver qué pasa:

```
console.log(4 + 6); // Imprime 10  
console.log("Hello " + "world!"); // Imprime "Hello world!"  
console.log("23" + 12); // Imprime "2312"  
console.log("42" + true); // Imprime "42true"  
console.log("42" + undefined); // Imprime "42undefined"  
console.log("42" + null); // Imprime "42null"  
console.log(42 + "hello"); // Imprime "42hello"  
console.log(42 + true); // Imprime 43 (true => 1)  
console.log(42 + false); // Imprime 42 (false => 0)  
console.log(42 + undefined); // Imprime NaN (undefined no puede ser convertido a number)  
console.log(42 + null); // Imprime 42 (null => 0)  
console.log(13 + 10 + "12"); // Imprime "2312" (13 + 10 = 23, 23 + "12" = "2312")
```

Cuando un operando es string, siempre se realizará una concatenación, por tanto se intentará transformar el otro valor en un string (si no lo es). En caso contrario, intentará hacer una suma (convirtiendo valores no numéricos a número). Si la conversión del valor a número falla, devolverá NaN (Not a Number).

Operadores aritméticos

Otros operadores aritméticos son: resta (-), multiplicación (*), división (/), y resto (%). Estos operadores operan siempre con números, por tanto, cada operando debe ser convertido a número (si no lo era previamente).

```
console.log(4 * 6); // Imprime 24  
console.log("Hello " * "world!"); // Imprime NaN  
console.log("24" / 12); // Imprime 2 (24 / 12)  
console.log("42" * true); // Imprime 42 (42 * 1)  
console.log("42" * false); // Imprime 0 (42 * 0)  
console.log("42" * undefined); // Imprime NaN  
console.log("42" - null); // Imprime 42 (42 - 0)  
console.log(12 * "hello"); // Imprime NaN ("hello" no puede ser convertido a número)
```

```
console.log(13 * 10 - "12"); // Imprime 118 ((13 * 10) - 12)
```

Operadores unarios

En JavaScript podemos preincrementar (++variable), postincrementar (variable+), predecrementar (--variable) y postdecrementar (variable--).

```
let a = 1;
let b = 5;
console.log(a++); // Imprime 1 y incrementa a (2)
console.log(++a); // Incrementa a (3), e imprime 3
console.log(++a + ++b); // Incrementa a (4) y b (6). Suma (4+6), e imprime 10
console.log(a-- + --b); // Decrementa b (5). Suma (4+5). Imprime 9. Decrementa a (3)
```

También, podemos usar los signos - y + delante de un número para cambiar o mantener el signo del número. Si aplicamos estos operadores con un dato que no es un número, éste será convertido a número primero. Por eso, es una buena opción usar +value para convertir a número, lo cual equivale a usar Number(value).

```
let a = "12";
let b = "13";
let c = true;
console.log(a + b); // Imprime "1213" console.log(+a +
+b); // Imprime 25 (12 + 13) console.log(+b + +c); //
Imprime 14 (13 + 1). True -> 1
```

Operadores relacionales

El operador de comparación, compara dos valores y devuelve un booleano (true o false) Estos operadores son prácticamente los mismos que en la mayoría de lenguajes de programación, a excepción de algunos, que veremos a continuación.

Podemos usar == o === para comparar la igualdad (o lo contrario !=, !==). La principal diferencia es que el primero, no tiene en cuenta los tipos de datos que están siendo comparados, compara si los valores son equivalentes. Cuando usamos ===, los valores además deben ser del mismo tipo. Si el tipo de valor es diferente (o si es el mismo tipo de dato pero diferente valor) devolverá falso. Devuelve true cuando ambos valores son idénticos y del mismo tipo.

```
console.log(3 == "3"); // true
console.log(3 === "3"); // false
console.log(3 != "3"); // false
console.log(3 !== "3"); // true
// Equivalente a falso (todo lo demás es equivalente a cierto)
console.log("" == false); // true
console.log(false == null); // false (null no es equivalente a cualquier boolean).
console.log(false == undefined); // false (undefined no es equivalente a cualquier boolean).
console.log(null == undefined); // true (regla especial de JavaScript)
console.log(0 == false); // true
console.log({} >= false); // Object vacío -> false
console.log([] >= false); // Array vacío -> true
```

Otros operadores relaciones para números o strings son: menor que (<), mayor que (>), menor o igual que (<=), y mayor o igual que (>=). Cuando comparamos un

string con estos operadores, se va comparando carácter a carácter y se compara su posición en la codificación Unicode para determinar si es menor (situado antes) o mayor (situado después). A diferencia del operador de suma (+), cuando uno de los dos operandos es un número, el otro será transformado en número para comparar. Para poder comparar como string, ambos operandos deben ser string.

```
console.log(6 >= 6); // true
console.log(3 < "5"); // true ("5" → 5)
console.log("adiós" < "bye"); // true
console.log("Bye" > "Adiós"); // true
console.log("Bye" > "adiós"); // false. Las letras mayúsculas van siempre antes
console.log("ad" < "adiós"); // true
```

Operadores booleanos

Los operadores booleanos son negación (!), y (&&), o (||). Estos operadores normalmente son usados de forma combinada con los operadores relacionales formando una condición más compleja, la cual devuelve true o false.

```
console.log(!true); // Imprime false
console.log(!(5 < 3)); // Imprime true (!false)

console.log(4 < 5 && 4 < 2); // Imprime false (ambas condiciones deben ser ciertas)
console.log(4 < 5 || 4 < 2); // Imprime true (en cuanto una condición sea cierta, devuelve cierta y deja de comparar)
```

Se puede usar el operador y, o el operador o con valores que no son booleanos, pero se puede establecer equivalencia (explicada anteriormente). Con el operador or, en encontrarse un true o equivalente, lo devolverá sin seguir evaluando el resto. El operador and al evaluar las condiciones, si alguna de ellas es falsa o equivalente no sigue evaluando. Siempre se devuelve la última expresión evaluada.

```
console.log(0 || "Hello"); // Imprime "Hello"
console.log(45 || "Hello"); // Imprime 45
console.log(undefined && 145); // Imprime undefined
console.log(null || 145); // Imprime 145
console.log("" || "Default"); // Imprime "Default"
```

Usamos la doble negación !! para transformar cualquier valor a booleano. La primera negación fuerza el casting a boolean y niega el valor. La segunda negación, vuelve a negar el valor dejándolo en su valor equivalente original.

```
console.log(!null); // Imprime false
console.log(!undefined); // Imprime false
console.log(!undefined === false); // Imprime true
console.log(!""); // Imprime false
console.log(!0); // Imprime false
console.log(!"Hello"); // Imprime true
```

El operador boolean || (or) puede ser usado para simular valores por defecto en una función. Por ejemplo:

```
function sayHello(name) {
  // Si nombre es undefined o vacío (""), Se le asignará "Anonymous" por defecto
```

```
let sayName = name || "Anonymous";  
console.log("Hello " + sayName);  
}
```

```
sayHello("Peter"); // Imprime "Hello Peter"  
sayHello(); // Imprime "Hello Anonymous"
```

Fundamentos de JavaScript.

Programación con JavaScript
Cefire 2021

Autor: Arturo Bernal Mayordomo