LMSGI

1. Conceptos básicos de Javascript

Alejandro Amat Reina

Javascript

- En este primer tema, veremos los fundamentos de **JavaScript**.
- JavaScript es un lenguaje interpretado
- Se ejecuta por un interprete, normalmente integrado en un navegador web (aunque existen otros ámbitos de uso).

EcmaScript

- Lo que se conoce como JavaScript es en realidad una implementación de ECMAScript, el estándar que define las características de dicho lenguaje.
- La versión mas reciente de ECMAScript specification es **ES2017** (Junio del 2017), el cual es también nuevo y no está todavía implementado en todos los navegadores.
- Para más información sobre las versiones de javascript visitad el siguiente enlace:
  - https://www.campusmvp.es/recursos/post/JavaScript-ECMAScript-ES6-Existe-ES7-Aclarando-las-diferentes-versiones-del-lenguaje.aspx

Editor de escritorio

- Para editar código javascript puedes utilizar el editor que más te guste (Webstorm, Brackets, Atom, Notepad++, etc.)
- Yo te recomiendo Visual Studio Code, porque se integra muy bien JavaScript, NodeJS, TypeScript y Angular
- Puedes descargarlo de:
  - https://code.visualstudio.com/Download

Editores web

- Puedes usar editores web para probar código que no sea demasiado complejo o extenso
- Dos famosos editores web son:
  - Fiddle: https://jsfiddle.net/
  - Plunker: https://plnkr.co/

Chrome devtools

- Tanto Chrome como Firefox integran un potente conjunto de herramientas para desarrolladores
- Puedes acceder a ellas pulsando la tecla F12
- Usaremos estas herramientas para depurar nuestro código Javascript y entender mejor lo que está pasando
- En el siguiente enlace tienes más información sobre esto:
  - https://developers.google.com/web/tools/chrome-devtools?hl=es

Integrar Javscript en el HTML

- Tenemos dos formas de incorporar código javascript en una página HTML, pero siempre usando la etiqueta <script>:
  - Dentro del mismo código HTML:

```
...
  <body>
  <script>
     console.log("Hola Mundo!");
  </script>
  </body>

...
```

  - En un archivo a parte (RECOMENDADO)

```
...
  <body>
  <script src="js/ejemplo1.js"></script>
  </body>

...
```

Archivo js/ejemplo1.js

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

Funciones de E/S

- En Javascript disponemos de varias funciones de entrada/salida para mostrar y recoger datos del usuario.

- En realidad todas las funciones que vamos a ver están en desuso, ya que en la actualidad para interactuar con el usuario se utilizan formularios web, pero las utilizaremos para poder ir haciendo los ejercicios, a medida que vamos viendo distintos conceptos.

Funciones de salida

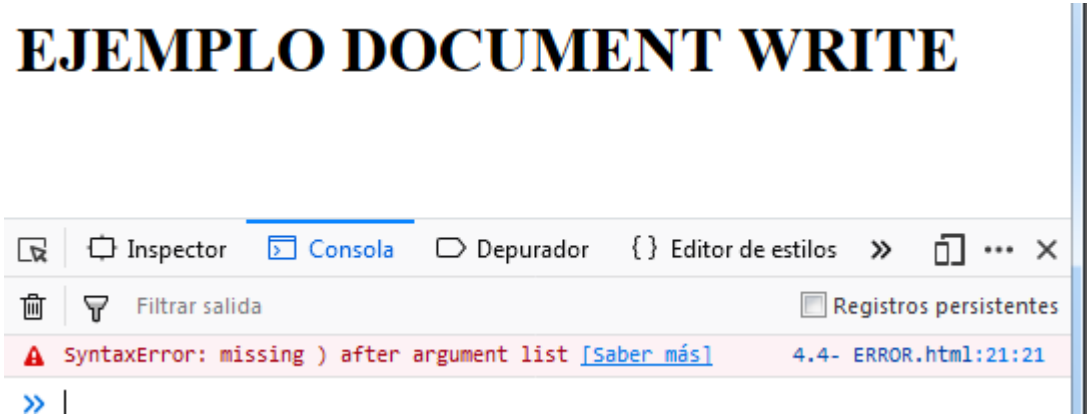
- **console.log(texto) y console.error(texto)**
  - Muestra el texto que le pasamos en la solapa consola de las herramientas del desarrollador (Muy útiles para testeo de variables).
- **alert(texto)**
  - Método del objeto window del DOM que muestra una ventana emergente con el texto indicado.

Funciones de entrada

- Los siguientes métodos pertenecen al objeto window del DOM, que tenemos disponible en cualquier punto de la ejecución de nuestros scripts:
  - **prompt(texto, valor por defecto)**
    - Muestra una ventana emergente con el texto indicado y una caja de entrada para que el usuario pueda escribir.
    - El segundo parámetro es opcional.
    - Devolverá el texto que escriba el usuario.
  - **confirm(texto)**
    - Muestra una ventana emergente con el texto indicado y con los botones de OK y CANCEL
    - Se utiliza para los mensaje de confirmación ante una acción.
    - Devolverá true o false dependiendo del botón pulsado.
  - **document.write(texto):**
    - Escribe el texto recibido en el documento.
    - Se suele utilizar junto con código HTML para que el navegador lo interprete como tal y le de formato o estilo.

Errores de sintaxis

- En la solapa consola de las herramientas de desarrollador nos aparecerán los errores sintácticos que cometamos.
- El error aparecerá cuando el interprete trate de ejecutar la instrucción que contenga el error
- En el error nos indicará el fichero y la línea donde se produzca el error



Ejercicio 1

- Para hacer algunos de los ejercicios usaremos como base el libro de javascript publicado por Javier Eguiluz en el siguiente enlace:
  - https://uniwebsidad.com/libros/javascript
- En primer lugar haremos el ejercicio que encontraremos al final del capítulo 2:
  - https://uniwebsidad.com/libros/javascript/capitulo-2
- Debes utilizar console.log en lugar de alert
- Introduce algún error sintáctico en el código y verifica cómo te lo muestran las herramientas de desarrollador en la solapa consola

Variables

- Para nombrar las variables usaremos letras o números sin espacios
- El primer caracter del nombre debe ser una letra, el carácter _ o el carácter $.
- Javascript es case sensitive, por lo que se distingue entre mayúsculas y minúsculas.
- Javascript es un lenguaje de tipado débil, es decir, el tipo de las variables dependerá del valor que les asignemos
- El tipo de una variable cambiará si le asignamos valores que tengan tipos distintos

Declaraciones de variables

- Podemos declarar variables de cuatro formas (usando cuatro palabras reservadas para ello):
  - global: son variables globales a todo el programa, no son nada recomendables.
  - var: Es la declaración usada antes de ES6, pero tiene un problema que veremos más adelante (mejor usar let).
  - let: Declaración que aparece con ES6, sustituye a var, ya que no tiene el problema del hoisting que veremos a continuación.
  - const: Mediante esta declaración definiremos constantes

Hoisting

- Javascript mueve todas las declaraciones de las variables al principio del ámbito actual.
- Esto se conoce como Hoisting.
- Antes de ejecutar el código se realizan dos tareas:
  - Se carga la declaración de funciones en memoria (por tanto, pueden ser accesibles desde cualquier posición).
  - Se mueve la declaración de variables al principio de las funciones (si no son locales a una función, se mueven al principio del bloque principal).
  - Esto ocurre sólo con las variables declaradas con var

```
function printHello() {
    console.log(hello);
    var hello = "Hello World";
}
printHello(); // Esto imprimirá undefined
```

Transformación de código

- En el código anterior, lo esperado es que se produjera un error, ya que estamos tratando de imprimir una variable antes de que exista.

- Sin embargo, imprimirá “undefined”, ya que internamente el código se ha transformado a este:

```
function printHello() {
    var hello = undefined;
    console.log(hello);
    hello = "Hello World";
}
printHello();
```

Variables let mejor que var

- Código con var:

```
if (2 > 1) {
    var nombre = "Pablo";
    console.log("Nombre dentro:", nombre); // Imprimirá Pablo
}
console.log("Nombre fuera:", nombre); // Imprimirá Pablo
```

- Código con let:

```
if (2 > 1) {
    let nombre = "Pablo";
    console.log("Nombre dentro:", nombre); // Imprimirá Pablo
}
console.log("Nombre fuera:", nombre); // Error variable no declarada
```

- El comportamiento con let es mucho más lógico

Variables let mejor que var

- Con var:

```
for(var i = 0; i<=10; i++) {
    console.log(i);
}
console.log(i); // Nos devolverá 11!.
```

- Con let:

```
for(let i = 0; i<=10; i++) {
    console.log(i);
}
console.log(i); // Nos devolverá Error!.
```

Tipos de variables

- typeof: Indica el tipo de dato que en ese momento tiene la variable.

```
let v1 = "Hola Mundo!";
console.log(typeof v1); // Imprime -> string
v1 = 123;
console.log(typeof v1); // Imprime -> number
```

- Hasta que le asignemos un valor, las variables tendrán un tipo especial conocido como **undefined**.
- Este valor es diferente de **null** (que si se considera como un valor).

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

Ejercicio 2

- Realizaremos ahora el ejercicio 2 del libro de javascript:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/tipos-de-variables
- Puedes leer el capítulo del libro referente al uso de strings para ver cómo se escapan las comillas
- Utiliza console.log en lugar de alert
- Utiliza let en lugar de var

Arrays

- En JavaScript, los arrays son un tipo especial de objetos.
- Podemos crear un array con la instancia de un objeto de clase Array.
- Estos no tienen un tamaño fijo, por tanto, podemos inicializarlo con un tamaño y luego añadirle más elementos.
- El constructor puede recibir 0 parámetros (array vacío), un número (el tamaño del array), o en cualquier otro caso, se creará un array con los elementos recibidos.
- Debes tener en cuenta que en JavaScript un array puede contener al mismo tiempo diferentes tipos de datos.

```
let a = new Array(); // Crea un array vacío
let b = new Array(10); // Crea un array de tamaño 10
let c = new Array("10", 5, 1); // Crea un array de tres elementos
inicializados a los valores pasados
```

Acceso a los elementos

- Accedemos a los elementos del array con el operador []
- Fíjate que cuando accedes a una posición del array que no ha sido definida, devuelve undefined.
- Para obtener la longitud del array accedemos a la propiedad length del objeto
- La longitud de un array depende de las posiciones que han sido asignadas.

```
let a = new Array(); // Crea un array vacío
a[0] = 13;
console.log(a.length); // Imprime 1
console.log(a[0]); // Imprime 13
console.log(a[1]); // Imprime undefined
```

Ejemplo

- Vamos a ver un ejemplo de lo que ocurre cuando asignas una posición mayor que la longitud y que no es consecutiva al último valor asignado.

```
let a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
a[20] = "Hello";
console.log(a.length); // Ahora imprime 21 (0-20). Las posiciones 0-19 tendrán el valor undefined
```

Reducir la longitud de un array

- Podemos reducir la longitud del array modificando directamente la propiedad **length**.
- Si reducimos la longitud de un array, las posiciones mayores a la nueva longitud serán consideradas como undefined.

```
let a = new Array("a", "b", "c", "d", "e"); // Array con 5 valores
console.log(a[3]); // Imprime "d"
a.length = 2; // Posiciones 2-4 serán destruidas
console.log(a[3]); // Imprime undefined
```

Creación alternativa

- Puedes crear un array usando corchetes en lugar de usar **new Array()**.
- Los elementos que pongamos dentro, separados por coma serán los elementos que inicialmente tendrá el array.

```
let a = ["a", "b", "c", "d", "e"]; // Array de tamaño 5, con 5 valores inicialmente
console.log(typeof a); // Imprime object
console.log(a instanceof Array); // Imprime true. a es una instancia de array
a[a.length] = "f"; // Insertamos in nuevo elemento al final
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]
```

Ejercicio 3

- Vamos a hacer el ejercicio 3 del libro de Javier Eguiluz:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/tipos-de-variables
- Debes utilizar los dos tipos de creación del array
- Debes usar console.log en lugar de alert

Tipos de datos numéricos (number)

- En Javascript 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
```

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)
```

El objeto Number

- Existe también un objeto del lenguaje llamado Number
- A través de este objeto 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)
```

- Puedes consultar las propiedades y métodos de este objeto en el siguiente enlace:
  - https://www.w3schools.com/jsref/jsref_obj_number.asp

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, y no es considerado como número valido
- Cuando hacemos una operación numérica con valores que no son números, Javascript intenta transformar esos valores a números (*conversión de tipos* implicita o cast implicito).
- Si no puede realizar el cast, nos devuelve un valor especial llamado NaN (Not a Number).

Ejemplos de cast implicito

```
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
```

Valores undefined y null

- En JavaScript cuando una variable (o parámetros de una función) han sido creados sin asignarles un valor, es inicializada con el valor especial undefined.
- No deberíamos confundir **undefined** con **null**.
- La segunda 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
```

Boolean

- En Javascript los booleanos se representan en minúsculas (**true**, **false**).
- Puedes negarlos usando el operador **!** antes del valor.
- Puedes consultar las propiedades y métodos de este objeto en el siguiente enlace:
  - https://www.w3schools.com/jsref/jsref_obj_boolean.asp

Strings

- En JavaScript los valores string son representados 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
```

Comillas en los strings

- 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, ocurriria 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"
```

El objeto String

- 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 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"
```

Conversión de tipo explicita (String)

- Puedes convertir un dato a **string** usando la funcion **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
```

Conversión de tipo explicita (Number)

- Puedes convertir un dato en **number** usando la función Number(value).
- Si el valor no es un número la función devolverá NaN
- 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
```

Conversión de tipo explicita (Booleano)

- 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 vacio** (“”), **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
```

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?
- Veamos los ejemplos:

```
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")
```

Conversiones implícitas en la suma

- Cuando hay un string, siempre se realizará una concatenación, por tanto, si el otro valor no es un string se intentará transformar en un string.
- Si no hay strings, y algún valor no es un número, lo intentará convertir a número e intentará hacer una suma.
- Si la conversión del valor a número falla, devolverá **NaN** (Not a Number).

Operadores aritméticos

- operadores aritméticos son: **resta** (-), **multiplicación** (*), **división** (/) y **módulo** (%).
- Estos operadores operan siempre con números, por tanto, cualquier operando que no sea un número debe ser convertido a número.

```
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 incremento y decremento

- 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)
```

Operador cambio de signo

- Podemos usar los signos – y + delante de un número para cambiar o mantener su signo.
- Si aplicamos estos operadores con un dato que no es un número, este 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 dato o el valor son diferentes devolverá falso.
- Devolverá true cuando ambos valores son idénticos y del mismo tipo.

Ejemplos

```
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 relacionales

- 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)
```

Funcionamiento

- Se puede usar el operador &&, o el operador || con valores que no son booleanos, pero se puede establecer equivalencia.
- El operador ||, al encontrarse un true o equivalente, lo devolverá sin seguir evaluando el resto.
- El operador && al evaluar las condiciones, si alguna de ellas es falsa o equivalente no seguirá 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"
```

Doble negación !!

- 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
```

Condicional simple (if else)

- Similar a la mayoría de los lenguajes de programación.
- Evalúa una condición, 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!");
}
```

Ejercicio 4

- Vamos a realizar el ejercicio 4 del libro de Javier Eguiluz:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/operadores
- Debes usar console.log en lugar de alert

Ejercicio 5

- Vamos a realizar el ejercicio 5 del libro de Javier Eguiluz:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/estructuras-de-control-de-flujo
- Usaremos console.log en lugar de alert

Ejercicio 6

- Vamos a realizar el ejercicio 6 del libro de Javier Eguiluz:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/estructuras-de-control-de-flujo
- Usaremos console.log en lugar de alert

Objetos Globales y funciones

- En JavaScript existen algunas funciones y objetos globales que pueden ser accedidas desde cualquier sitio.
- Estas funciones y objetos nos facilitarán mucho el trabajo con números, cadenas, etc.
- A continuación veremos un resumen de estos objetos y funciones.
- Algunos de ellos ya los hemos nombrado anteriormente
- Otros los estudiaremos con más detenimiento más adelante

Funciones globales para trabajar con números

- **parseInt(value)**
  - Transforma cualquier valor en un entero.
  - Devuelve el valor entero, o NaN si no puede ser convertido.
- **parseFloat(value)**
  - Igual que parseInt, pero devuelve un decimal.
- **IsNaN(value)**
  - Devuelve true si el valor es NaN.
- **isFinite(value)**
  - Devuelve true si el valor es un número finito o false si es infinito.
- **Number(value)**
  - Transforma un valor en un número (o NaN).

Ejercicio 7

- Crea un script que al cargar la página le pida al usuario un valor en euros y muestre por pantalla el valor convertido a dolares.
- Busca en Internet la correspondencia entre el euro y el dólar.
- Para pedir el valor utiliza la función prompt vista anteriormente.
- Para escribir en pantalla utiliza la función document.write.
- Debes verificar si el importe introducido es un número antes de hacer la conversión, si no lo es, se mostrará por pantalla el texto importe incorrecto.

Funciones globales para trabajar con strings

- **String(value)**
  - Convierte un valor en un string.
- **encodeURI(string)**
  - Transforma una cadena en una URL codificada, codificando caracteres especiales a excepción de: , / ? : @ & = + $ #.
- **decodeURI(string)**
  - Transforma una URL codificada en un string.
- Ejemplo:
  - URL Codificada: "http://domain.com?val=1%202%203&val2=r+y%256"
  - URL sin codificar: "http://domain.com?val=1 2 3&val2=r+y%6"
- **encodeURIComponent(string)**, **decodeURIComponent(string)**
  - Estas funciones también codifican y decodifican los caracteres especiales que encodeURI no hace. Se deben usar para codificar elementos de una url como valores de parámetros (no la url entera).
- Ejemplo:
  - URL sin codificar : "http://domain.com?val=1 2 3&val2=r+y%6"
  - URL Codificada : "http%3A%2F%2Fdomain.com%3Fval%3D1%202%203%26val2%3Dr%2By%256"

El objeto Math (Constantes)

- El objeto Math nos proporciona algunas constantes y métodos matemáticos bastante útiles
- Las principales constantes son las siguientes:
  - E (Numero de Euler)
  - PI
  - LN2 (algoritmo natural en base 2)
  - LN10
  - LOG2E (base-2 logaritmo de E)
  - LOG10E
  - SQRT1_2 (raíz cuadrada de $^1/_2$)
  - SQRT2.

El objeto Math (Métodos)

- **round(x)** → Redondea x al entero mas cercano
- **floor(x)** → Redondea x hacia abajo (5.99 → 5. Quita la parte decimal)
- **ceil(x)** → Redondea x hacia arriba (5.01 → 6)
- **min(x1,x2,...)** → Devuelve el numero mas bajo de los argumentos que se le pasan.
- **max(x1,x2,...)** → Devuelve el numero mas alto de los argumentos que se le pasan.
- **pow(x, y)** → Devuelve xy (x elevado a y).
- **abs(x)** → Devuelve el valor absoluto de x.
- **random()** → Devuelve un numero decimal aleatorio entre 0 y 1 (no incluidos).
- **cos(x)** → Devuelve el coseno de x (en radianes).
- **sin(x)** → Devuelve el seno de x.
- **tan(x)** → Devuelve la tangente de x.
- **sqrt(x)** → Devuelve la raiz cuadrada de x

Ejemplos

```
console.log("Raíz cuadrada de 9: " + Math.sqrt(9));
console.log("El valor de PI es: " + Math.PI);
console.log(Math.round(4.546342));
// Número aleatorio entre 1 y 10
console.log(Math.floor(Math.random() * 10) + 1);
```

Ejercicio 8

- Crea un script que calcule un número aleatorio entre 1 y 50
- Este número será el radio de una circunferencia
- A partir de este radio, debes calcular el diámetro, el área y el perímetro de la circunferencia.

Condicional múltiple (switch)

- Similar a otros lenguajes de programación.
- Se evalúa una variable y se ejecuta el bloque correspondiente al valor que tiene (puede ser numero, string,..).
- Al final de cada bloque pondremos la instrucción **break**, ya que, de no ponerlo continuaría ejecutando las instrucciones que haya en el siguiente bloque.
- Ejemplo en el que dos valores ejecutaran el mismo bloque de código:

```
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!");
}
```

Switch con condiciones

- En JavaScript 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 en 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");
}
```

Ejercicio 9

- Vamos a realizar un conversor multi moneda.
- Para ello, en primer lugar le preguntaremos al usuario el tipo de conversión a realizar.
- Los posibles valores serán:
  1. euro – dólar
  2. euro – libra
  3. euro – yen
- Una vez seleccionado el tipo de conversión se le preguntará el importe a convertir.
- Finalmente, se mostrará por pantalla el valor de la conversión en la moneda seleccionada.

Bucles

- En javascript tenemos los mismos bucles que en la mayoría de los lenguajes de programación:
  - While: Evalúa una condición antes de cada iteración del bucle y si la condición se cumple se ejecuta la iteración
  - Do … while: Igual que el while, pero evaluando la condición después de cada iteración del bucle
  - For: Bucle controlado por contador
  - For in: Adecuado para recorrer arrays o propiedades de objetos

while

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

Do ... while

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

Ejercicio 10

- Vamos a modificar el ejercicio 8:
  - Si el usuario introduce un valor erróneo en el tipo de conversión, se le volverá a pedir el tipo de conversión.
  - Si el usuario introduce un importe erróneo, se le volverá a pedir el importe.
  - El resultado de la conversión se mostrará utilizando la función confirm, y se le preguntará al usuario si desea realizar otra conversión.
  - Si responde que sí, se volverá a pedir el tipo de conversión.
  - Si responde que no, se mostrará por pantalla (con document.write) un mensaje de despedida.

for

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

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

Break y continue

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

Ejercicio 11

- Vamos a realizar el ejercicio 7 del libro de Javier Eguiluz:
  - https://uniwebsidad.com/libros/javascript/capitulo-3/estructuras-de-control-de-flujo
- Debes usar console.log en lugar de alert

Ejercicio 12

- Escribe un script que le pida una cadena de texto al usuario y la muestre al revés en la página.
- Para obtener la longitud de una cadena, puedes utilizar la propiedad length del objeto string
- Para obtener el carácter de la posición i de una cadena, puedes utilizar el método charAt del objeto string
- Para más información puedes consultar el siguiente enlace:
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String

For .. in

- Otra versión del for, es el bucle for..in.
- Con este bucle podemos iterar los índices de un array o las propiedades de un objeto (similar al bucle foreach de otros lenguajes, pero recorriendo los índices en lugar de los valores).

```
let ar = new Array(4, 21, 33, 24, 8);
for (var index in ar) { // Imprime 4 21 33 24 8
    console.log(ar[index]);
}
```

Ejercicio 13

- Crea un array que contenga los nombres de los días de la semana.
- Muestra todos los elementos del array por consola, recorriéndolo con un bucle for … in

For ... of

- Desde ES2015 podemos iterar por los elementos de un array o incluso por los caracteres de una cadena sin utilizar el índice.
- Para ello se utiliza el bucle **for..of** (que se comporta como un bucle foreach de otros lenguajes).

```
let a = ['item1', 'item2', 'item3', 'item4']

for (let item of a)
   console.log(item);
```

```
let cadena = "Álex";

for (let letra of cadena)
   console.log(letra + ' ');
```

Ejercicio 14

- Crea un string que contenga tu nombre
- Utilizando un bucle for … of crea otro string que contenga tu nombre al revés.