LENGUAJES DE MARCAS

Fundamentos de JavaScript

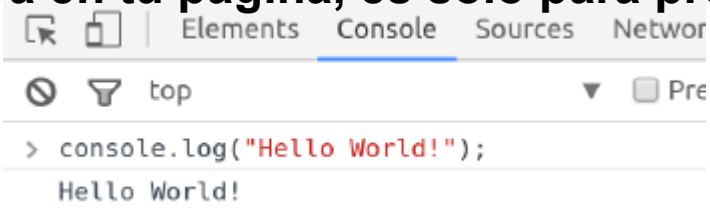
Pablo Matías Garramone Ramírez

Introducción

- En este primer documento, vamos a aprender los fundamentos de **JavaScript** en cuanto a sintaxis se refiere.
- La version de JavaScript utilizada durante el curso será ES2015 (ES6). Versión donde se introdujeron numerosos cambios en el lenguaje y una modernización necesaria después de pasar muchos anos sin apenas cambios desde la primera versión del lenguaje en 1997.
- JavaScript es un lenguaje interpretado, ejecutado por un interprete normalmente integrado en un navegador web (pero no solo 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 mas reciente de ECMAScript specifcation es **ES2017** (Junio del 2017) el cual es también nuevo y no esta todavía implementado en todos los navegadores.

Editores y Herramientas

- Dispondremos de tres opciones para trabajar:
  - **La consola del Navegador:** Abre la consola del Navegador (Chrome o Firefox) y pulsa F12 (Herramientas del desarrollador). Ve a la pestaña Consola, aquí puedes escribir código JavaScript y ver el resultado, es interesante para testear partes de tu código (**recuerda que esto NO se verá en tu página, es solo para probar**)



  - **Editores de Escritorio:** Pudes usar el que mas 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.

    Puedes descargarlo de: https://code.visualstudio.com/Download
  - **Editores WEB:** Podemos usar editores web, al menos para probar codigo que no sea muy complejo o muy grande.

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

Pablo Matías Garramone Ramírez

Integrando JavaScript con HTML

- **Igual que vimos con CSS tenemos dos formas para poder incorporar código en una página HTML, pero en este caso siempre usando la etiqueta <script>** ***(Ejemplos 1 y 2)***
  - **Dentro del mismo código HTML:**

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

  - **En un archivo a parte (RECOMENDADO)**

```
<!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!");
```

Pablo Matías Garramone Ramírez

Etiqueta <noscript>

- La etiqueta <noscript> se utiliza para poner código HTML que será renderizado solo 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.

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

Pablo Matías Garramone Ramírez

Principales Funciones Entrada/Salida

- **console.log(texto) y console.error():** Es una función que sirve SOLO y ESCLUSIVAMENTE para mensajes de depuración o testeo para el programador. Carece de utilidad visual alguna en la aplicación. Se utiliza para 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.
- **alert(texto):** Método del objeto Window del DOM que muestra una ventana emergente con el texto indicado con un botón de OK. ***(3-evento.html)***
- **prompt(texto, valor x defecto):** Método del objeto Window del DOM que muestra una ventana emergente con el texto indicado y una caja de entrada para que el usuario pueda escribir, tiene un botón de OK y CANCEL. Este método se suele asociar una variable para guardar lo que el usuario haya escrito. Se utiliza para pedir datos al usuario. El segunda parámetro es opcional y deja rellenada la caja con ese valor por defecto. ***(4.1-PROMPT.htm)***
- **confirm(texto):** Método del objeto Window del DOM que 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. Devolver true (OK) o false CANCEL) dependiendo el botón pulsado. ***(4.2 - CONFIRM.html)***
- **document.write(texto):** Método del objeto document del DOM, que escribe en la pantalla lo indicado. Se suele utilizar junto con código HTML para que el navegador lo interprete como tal y le de formato o estilo. Con writeln() escribe un retorno de carro al final de la línea. ***(4.3- document_write.html)***

Pablo Matías Garramone Ramírez

Depurar Errores

- Debemos programa javaScript con la ventana de desarrollo (F12) habilitada para ver si hay errores cuando probemos nuestra página.
- Ver el ejemplo **4.4- ERROR.html** y encontrar el error.

Pablo Matías Garramone Ramírez

Variables. Nombrarlas

- Son elementos (posiciones de memoria), donde almacenamos información para luego trabajar sobre ella en el programa.
- Para darles nombre debemos usar letras o números sin espacios y tienen formato CamelCase, es decir, la primera letra en minúsculas, también puede empezar por subrayado (_nombre) o por dólar ($nombre).
- JS es case sensitive, por lo que se distingue entre mayúsculas y minúsculas.
- No existen tipos de datos, de hecho se pueden cambiar al vuelo dentro del programa. Puedes asignar un string y posteriormente un número a la misma variable.

Variables. Typeof, undefined y null

- 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
```

- ¿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**.
- Este valor es diferente de **null** (el cual se considera un valor). Que significa valor ninguno o sin valor (no confundir con 0).

```
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");
}
```

Pablo Matías Garramone Ramírez

Variables. Ámbito

- Es la zona de vida de una variable, donde existe.
- Puede ser:
  - Global: Accedemos al valor de la variable en todo el código del programa
  - Local: Accedemos al valor de la variable sólo en el lugar donde se ha creado y elementos anidados.
- Cuando usamos una variable JS va a buscar su declaración, con var o let, en cuanto lo encuentra para de buscar. Si no hacemos las declaración JS la declara como global implícitamente.
- Siempre prevalece lo local sobre lo global. Realmente es una jerarquía de niveles, JS cuando encuentra una variable primero busca en el nivel donde se encuentra, si no busca en el nivel superior y va subiendo hasta el nivel global. Es decir vamos siempre de a dentro hacía a fuera, por ello las variables globales se ven en todo el programa y las locales no.
- **4.5- Alcance_variables.html**

Pablo Matías Garramone Ramírez

Variables. Ámbito II

- Cuando se declara una variable en el bloque principal (fuera de cualquier función), esta 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, como veremos enseguida)

```
let  global = "Hello";
function cambiaGlobal() {
   global = "GoodBye";
}

cambiaGlobal();
console.log(global); // Imprime "GoodBye"
console.log(window.global); // Imprime "GoodBye"
```

- 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 actualizara el valor de la variable global.

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

let person = "John";
setPerson();
console.log(person); // Imprime John
```

Pablo Matías Garramone Ramírez

Variables. global, let, var y const

- Para declarar variables debemos indicar global, let, var o const.
- Global son variables globales a todo el programa, no son nada recomendables.
- Se recomienda let sobre var, por el problema del “hostling”, es un comportamiento particular de las variables declaradas con var, donde JavaScript NO da error cuando usamos una variable antes de declararla (lo veremos ahora)
- Con let lo evitamos ya que forzamos a que la variable será local al ámbito que se declara.
- Con const mantendremos fijo el valor asignado a esa variable durante todo el programa

Pablo Matías Garramone Ramírez

Variables. Hostling

- Aparece cuando declaramos variables con VAR.
- Antes de ejecutar el código, JS lo analiza y realiza dos cosas:
  - Carga la declaración de funciones en memoria (por tanto, pueden ser accesibles desde cualquier posición).
  - Mueve la declaración de variables al principio de las funciones (si no es local a una función, la mueve al principio del programa/bloque principal).

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

- Sin embargo, esto debería imprimir un error porque estamos intentando imprimir la variable *hello* antes de que exista. En este caso imprime "undefined", porque en el primer paso ha transformado el código internamente a este:

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

Pablo Matías Garramone Ramírez

Variables. let mejor que var

- Código con VAR:

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

  - Pondrá “Pablo” en ambos casos, cuando intuitivamente el de fuera debería fallar.

- Código con LET:

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

  - Dará error el segundo console.log, con un funcionamiento lógico

Pablo Matías Garramone Ramírez

Variables. let mejor que var II

- 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!.
```

Pablo Matías Garramone Ramírez

Variables. Use strict

- ¿Qué ocurre si se nos olvida poner **let** o **var**?. JavaScript declarara esa variable como **global**. Esto **NO** es lo recomendado porque las variables globales son peligrosas.
- Por lo tanto, es recomendable que usemos siempre **let** la primera vez que vayamos a necesitar una variable local.
- Para evitar que se nos olvide declarar con **let**, podemos usar una declaración especial: **'use strict'** al comienzo de nuestros archivos 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
>
```

Pablo Matías Garramone Ramírez

Variables. Constantes

- Cuando a lo largo de una función o bloque, una variable no va a cambiar de valor, o cuando queremos defnir un valor global inmutable (por ejemplo el numero 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.
- const PI = 3.1416;

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

Pablo Matías Garramone Ramírez

FUNCIONES

- Se utiliza la palabra reservada **function** antes del nombre de la función.
- Los argumentos se pasan entre paréntesis tras el nombre de la función (recuerda que en JS no hay tipos de datos)
- Posteriormente va el cuerpo de la función entre llaves.
- El nombre de la función, como las variables, debe seguir el formato CamelCase con la primera letra en minúscula.

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

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

Pablo Matías Garramone Ramírez

FUNCIONES. Declaración y llamada

- Puedes declarar el prototipo de la función donde quieras.
- No necesitas tener la función declarada antes de llamarla. Ya que JS primero procesa las declaraciones de las variables y funciones y después ejecuta el resto del código.
- Para llamar a una función simplemente pones el nombre de la función junto con el valor de los parámetros que hayas designado.
- Podemos hacerlo con más o menos parámetros de los que realmente tiene:
  - Si enviamos de más: Los sobrantes los ignora
  - Si enviamos de menos: Los que faltan les asigna el valor undefined.
- sayHello(); // Imprime undefined

Pablo Matías Garramone Ramírez

FUNCIONES. Retorno de valores

- Las funciones puedes hacer su código sin devolver nada o devolver algún valor que queramos enviar al que la llamó.
- Para devolver algo dentro de una función utilizaremos la palabra reservada **return**.
- Para recibir el valor enviado por la función, la llamada deberemos asignársela a una variable o meterla dentro del parámetro de otra función.
- Si intentamos recuperar algo de una función que no devuelve nada nos dará undefined.

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

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

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

console.log(totalPrice(5.95, 6)) // Imprime 35.7
```

Pablo Matías Garramone Ramírez

FUNCIONES. Funciones anónimas

- Como su nombre indica, son funciones sin nombre.
- Podemos asignar dicha función como valor a una variable, ya que es un tipo de valor (como puede ser un *string* o numero), 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
```

Pablo Matías Garramone Ramírez

FUNCIONES. Funciones anónimas vs con nombre

- No tienen diferencia en cuanto a rendimiento u operatividad.
- Con las funciones anónimas ahorras código y se suelen usar si no van a ser llamadas repetidas veces, también son el primer paso para las funciones flecha (ahora veremos).
- Si vamos a usar mucho la función o hacer recursividad es conveniente funciones con nombre.
- La principal diferencia es el hostling antes mencionado. La declaración de variables y funciones se van SIEMPRE al principio del ámbito.

```
function miFunc() {
   alert(v());
   var v = function() {return 5;};
}
```

Dará Error

```
function miFunc() {
   alert(v());
   function v() { return 5; };
}
```

Funcionará

Pablo Matías Garramone Ramírez

FUNCIONES. Funciones anónimas vs con nombre II

- En el primer caso, JS realmente está ejecutando esto:

```
function miFunc() {
   var v;
   alert(v());
   v = function() { return 5;};
}
```

- La declaración (que no la asignación!!!) se mueve al principio. Por eso da error de que la función v() no esta definida.
- En cambio en el segundo caso está ejecutando esto, y funciona sin problema:

```
function miFunc() {
   function v() { return 5; };
   alert(v());
}
```

- Por el hostling, la función es como si estuviera arriba del todo del bloque.

Pablo Matías Garramone Ramírez

FUNCIONES. Funciones anónimas vs con nombre III

- Como hemos dicho, el rendimiento es idéntico en ambas, pero no debemos caer en el error usando funciones anónimas como este:

```
for(var i=0; i<colElementos.length; i++){
   colElementos[i].onclick = function { .... };
}
```

- Ya que estamos creando una función entera nueva en cada pasada del bucle. Es mucho más eficiente declarar la función fuera, ya sea anónima o con nombre, y dentro del bucle simplemente usarla.

```
var manejador = function() {...};

for(var i=0; i<colElementos.length; i++){
   colElementos[i].onclick = manejador;
}
```

```
function manejador() {...};

for(var i=0; i<colElementos.length; i++){
   colElementos[i].onclick = manejador;
}
```

- **funciones.js**

Pablo Matías Garramone Ramírez

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 fecha).
- Otros lenguajes como C# o Java también las soportan.
- Ofrecen la posibilidad de crear funciones anónimas pero con algunas ventajas.
- Veamos una función hecha como anónima y justo debajo como arrow:

```
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
```

Pablo Matías Garramone Ramírez

FUNCIONES. Lambda o arrow functions II

- Cuando declaramos una función lambda, la palabra reservada **function** no se usa. **funciones lambda.js**
- Si solo se recibe un parámetro, los paréntesis pueden ser omitidos.
- Después de los parámetros debe ir una fecha (**=>**), y el contenido de la función.
- Si solo 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 implicita (devuelve el resultado de esa instrucción).

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

- Si hay mas 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
```

Pablo Matías Garramone Ramírez

FUNCIONES. Arrow functions. Ventajas

- Conseguimos un código más compacto, y conseguimos expresar las funciones de un modo resumido.
- Es ideal para el uso de las propiedades asíncronas del lenguaje JavaScript que usa mucho la parte del servidor Node.js con el uso de promesas.
- Al ser una función anónima se puede usar en el mismo lugar que se precisan, sin necesidad de declararlas antes.
- Otra de las diferencias que en estas funciones no podemos usar el elemento this o arguments. Si vemos objetos lo veremos. En caso que sea necesario el uso de this, deberemos usar funciones normales o anónimas.

Pablo Matías Garramone Ramírez

FUNCIONES. 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";
  ...
}
```

- Desde **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
```

Pablo Matías Garramone Ramírez

ESTRUCTURA CONDICIONALES. IF

- Se comporta como en cualquier otro lenguaje.
- Normalmente el código se ejecuta de forma secuencial línea a línea, pero si deseamos que una o varías líneas se ejecuten solo ante una determinada situación necesitamos utilizar estas estructuras condicionales.
- Esa determinada condición la evaluamos y puede devolver TRUE (entonces ejecuta el código) o FALSE (lo ignora y se lo salta).

  IF <CONDICION> <CODIGO A REALIZAR>
- Si queremos evaluar más condiciones y no solo 1 (solo se ejecuta el primero que cumpla, y si no cumple ninguno no se ejecuta nada), podemos enlazarlo con:

  IF <CONDICION> <CODIGO A REALIZAR 1>
  ELSE IF <CONDICION2> <CODIGO A REALIZAR 2>
  ELSE IF <CONDICION3> <CODIGO A REALIZAR 3>
  …
- Incluso si queremos poner un “cajón desastre” (obligando a que algo se ejecute), es decir si no se cumple nada de lo anterior, que haga otra cosa, usaríamos ELSE:

  IF <CONDICION> <CODIGO A REALIZAR 1>
  ELSE IF <CONDICION2> <CODIGO A REALIZAR 2>
  ELSE IF <CONDICION3> <CODIGO A REALIZAR 3>
  …
  ELSE <CODIGO A REALIZAR N>

```
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!");
}
```

Pablo Matías Garramone Ramírez

ESTRUCTURA CONDICIONALES. SWITCH

- Se evalúa una variable o condición y se ejecuta el código correspondiente al valor que tiene o resulta.
- Se suele necesitar un “break” al final de cada bloque ya que si no lo pones sigue ejecutando los siguientes bloques.
- Opcionalmente con en el IF, puedes poner un cajón desastre usando el elemento “default”.

```
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!");
}
```

Pablo Matías Garramone Ramírez

ESTRUCTURA CONDICIONALES. SWITCH II

- En JavaScript podemos hacer que SWITCH funcione exactamente igual que IF. Poniendo TRUE en la condición y poniendo condiciones en cada case como si de un IF se tratara (Ojo con el break):

```
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");
}
```

Pablo Matías Garramone Ramírez

BUCLES. WHILE

- Como muchos lenguajes, disponemos de otra estructura para cambiar el flujo del código, en este caso no para ejecutar una cosa u otra, si no para ejecutar una o varías línea de forma repetitiva.
- Es muy habitual en la algoritmia que necesitemos hacer una misma cosa muchas veces, para eso aparecen los bucles.
- Con WHILE, ponemos la condición que “mientras” se cumpla la condición (si no la cumple de entrada ni siquiera entrará 1 vez), lo que hay dentro entre llaves se ejecutara, en el momento de que deje de cumplirse se saldrá del WHILE y seguirá el código normalmente.
- Por lo que es muy importante, que dentro del bucle hagamos algo para que la condición en algún momento deje de cumplirse o entraremos en el temible “bucle infinito” que dejará colgado el programa e incluso la máquina!. *(value++ es lo mismo que value=value+1)*

```
let value = 1;

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

Pablo Matías Garramone Ramírez

BUCLES. DO WHILE

- Es muy parecido al anterior, solo que en este caso la condición se coloca al final del bucle.
- Lo que garantiza que al menos 1 vez siempre se ejecute.

```
let value = 1;

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

Pablo Matías Garramone Ramírez

BUCLES. FOR

- Es de los más utilizados, y muy útil cuando la condición de finalización es sencilla, llegar a un número concreto partiendo de otro, en caso de que las condiciones sean más complejas puede ser recomendable usar WHILE.
- Inicializamos uno o mas valores, establecemos la condición de fnalización 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);
}
```

- En JS No hay problema en inicializar más de 1 variable y poner más de 1 condición e incremento o decremento (separándolo por 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
*/
```

Pablo Matías Garramone Ramírez

BUCLES. BREAK y CONTINUE

- Dentro de un bucle, podemos usar las instrucciones de **break** y **continue**.
- **Break**: Provocará que salga del bucle de forma inmediata tras ejecutarse aunque no haya terminado de iterar aún.

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The number is 0
The number is 1
The number is 2

- **Continue**: 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**).

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

Pablo Matías Garramone Ramírez

TIPOS DE DATOS. NUMBER

- En JS no hay diferencia de números enteros y decimales. Todos son de tipo NUMBER.

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

- Podemos utilizar notación científica (exponencial).

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

- Para JS todos los números son OBJETOS (elementos que tienen métodos). Por lo que podemos usar ciertos métodos poniendo un punto a continuación del número (**importante tenemos que poner el número entre paréntesis para poder acceder a los métodos**, *en estos ejemplos estamos fijando el número de decimales o pasando a notación científica*):

```
console.log(3.32924325.toFixed(2)); // Imrpime 3.33
console.log(5435.45.toExponential()); // Imrpime 5.43545e+3
console.log((3).toFixed(2)); // Imrpime 3.00
```

Pablo Matías Garramone Ramírez

NUMBER. Operaciones con números

- Podemos utilizar las operaciones típinas de sumas (+), restar (-), multiplicar (*), dividir (/), resto (%)…
- Pero para ello debemos operar siempre con números, si operamos con tipos que no son números (por ejemplo algo entre comillas dobles) JS intenta hacer una conversión implícita a number, y si no lo consigue devuelve un valor especial NaN (Not a number).
- Si colocamos un + delante de una variable la trata de convertir 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
```

Pablo Matías Garramone Ramírez

NUMBER. Operaciones con números

- Podemos utilizar las operaciones típinas de sumas (+), restar (-), multiplicar (*), dividir (/), resto (%)…
- Pero para ello debemos operar siempre con números, si operamos con tipos que no son números (por ejemplo algo entre comillas dobles) JS intenta hacer una conversión implícita a number, y si no lo consigue devuelve un valor especial NaN (Not a number).
- Disponemos de la función **isNaN(valor)** que nos indica si un valor es NaN dará true y si no dará false.
- Si colocamos un + delante de una variable la trata de convertir 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
```

Pablo Matías Garramone Ramírez

Tipos de Datos. Undefined vs NULL

- Todos los tipos de datos disponen de dos valores muy distintos que no debemos confundir. Y que muchos lenguajes no distinguen, pero JS sí.
- Undefined es cuando una variable nunca se ha inicializado y por lo tanto carece de valor
- NULL es un valor que indica que la variable tiene el valor vacío. (No confundir con el 0 que es un número).
- NULL es considerado como un objeto (una referencia vacía) mientras que undefined es un valor especial que indica que la variable no se ha incializado.

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

value = null;
console.log(typeof value); // Imprime object
```

Pablo Matías Garramone Ramírez

Tipos de Datos. BOOLEAN

- En javaScript los booleanos se representan en minúscula y solo pueden tener los valores true, false, null o undefined.
- Puedes negarlos anteponiendo el carácter ! antes del valor.
- Pueden ser el fruto de una operación, tal como pasa cuando usamos las condiciones en las estructuras condicionales IF o SWITCH o en los bucles WHILE o FOR

Pablo Matías Garramone Ramírez

Tipos de Datos. STRINGS

- En JS los strings van con comillas simples o dobles.
- El operador para concatenar strings es +

```
let s1 = "Esto es un string";
let s2 = 'Esto es otro string';

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

- Cuando un string esta con comillas dobles puedes poner simples dentro y viceversa, pero si tienes dobles y quieres escribir dobles o simples y quieres escribir simples estas obligado a escaparlas con el carácter \

```
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"
```

- JavaScript soporta string **multilinea** con **sustitución de variables**. Ponemos el string entre caracteres ` (backquote) en lugar de entre comillas simples o dobles. La variable (o cualquier expresion 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
```

Pablo Matías Garramone Ramírez

Tipos de Datos. STRINGS Métodos

- Tal como ocurre con los números los strings son objetos, por lo tanto puedes acceder a una serie de métodos.
- Es importante comentar que lo que hagas con estos métodos no cambian el valor original del string, para ello deberías asignarlo a una variable.

```
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"]
```

Pablo Matías Garramone Ramírez

Tipos de Datos. STRINGS Métodos II

```
// 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(str.startsWith('Esto')); // Imprime true

// Devuelve si la cadena acaba en la subcadena recibida
console.log(str.endsWith("string")); // Imprime true

// Devuelve si la cadena contiene la subcadena recibida
console.log(str.includes("es")); // Imprime true

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

Pablo Matías Garramone Ramírez

Conversión de Tipos. (conversiones.js)

- 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 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
```

Pablo Matías Garramone Ramírez

Operadores. Suma +

- Como hemos mencionado se utiliza para sumar números o para concatenar strings.
- ¿Qué ocurre si intentamos sumar un número a un string o a otra cosa que no sea número o string?:
  - Cuando detecta que un operando es string, siempre se realizará una **concatenación**, por tanto se intentara transformar el otro valor en un string (si ya no lo es).
  - En caso de ser un número y otra cosa que no sea string, intentara hacer una suma (convirtiendo el valor no numéricos a número). Si la conversión falla, devolverá **NaN.**

```
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")
```

Pablo Matías Garramone Ramírez

Operadores. Operadores aritméticos

- Ante cualquier otra operación aritmética que no sea suma (+), es decir, **resta** (-), **multiplicación** (*), **división** (/) o **resto** (%), los operadores operan siempre con números, por tanto, cada operando debe ser convertido a numero (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)
```

Pablo Matías Garramone Ramírez

Operadores. 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 numero.
- 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
```

Pablo Matías Garramone Ramírez

Operadores relacionales. Comparación igualdad

- 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
```

Pablo Matías Garramone Ramírez

Operadores relacionales. Comparación

- 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 numero 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
```

Pablo Matías Garramone Ramírez

Operadores. Operadores booleanos

- Los operadores booleanos son **negacion** (!), **y AND** (&&) **o OR** (||), recuerda que este último también se puede utilizar para poner valores por defecto en una función. Estos operadores normalmente son usados de forma combinada con los operadores relacionales formando una condición mas 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 ultima 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 negacion !!** para transformar cualquier valor a booleano. La primera negacion 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(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"
```

Pablo Matías Garramone Ramírez