JavaScript

Introducción

- JavaScript (JS)
- Creado originalmente para que las webs estuvieran vivas
- No está relacionado con Java
- Programas son scripts

Características

- Multiplataforma
- Orientado a objetos
- Completa integración con HTML/ CSS
- Las cosas simples son sencillas
- Soporte de casi todos los navegadores
- Version de servidor
    - node.js

Datos

- JS en el navegador no tiene acceso a funciones del SO
    - Diferentes pestañas / ventanas en general no se conocen entre ellas
    - Una web puede comunicarse con el servidor del que vino

Añadir etiqueta <script> justo antes del closing body </body>

Codigo incrustado en HTML

```
<script>
  alert( 'Hello, world!' );
</script>
```

Script externo

- Se pone justo antes del closing body (</body>)

```
<script src="/path/to/script.js"></script>
```

Comentarios

```
console.log("Hello World!") //soy un comentario /*yo también*/
```

Prompt

- Le pide al usuario que introduzca información

```
//prompt(title, [default]);
let result = prompt("¿Cuantos años tienes?",""); console.log(result);
console.log(typeof(result)); //string
```

confirm

- Le solicita al usuario que confirme

```
//result = confirm(question);
let str = confirm("¿Estás de acuerdo?");
console.log(str) //true o false
```

console.log

- Imprime un mensaje por la consola del navegador

```
//console.log(mensaje);
console.log("Esto se imprime en consola")
```

Statements

- Declaraciones
- Separadas por ; aunque se puede omitir en un salto de línea a veces (no recomendable)

Variables

- Almacenamiento de datos con nombre
    - let message;
    - message = "Hello";
    - console.log(message);
- Declaración múltiple
    - let user = "John", age = 25, message = "Hello";
- Puede cambiar el tipo de datos, aunque mucho cuidado
    - let message;
        - console.log(typeof(message)) //undefined
    - message = "hello"
        - console.log(typeof(message)) //string
    - message = 1234
        - console.log(typeof(message)) //number

Sintaxis

- Solo pueden contener letras, dígitos o $ o _
- El primer carácter no puede ser un dígito
- Se recomienda usar lower camel Case para múltiples palabras

Constantes

- Para constantes conocidas recomendable que solo contengan mayúsculas y las palabras separadas por _
  - const COLOR_RED = "#F00";
- Para otras constantes usar lower camelCase
  - const pageLoadTime = /* time taken by a webpage */
- Cuando declaras la constante hay que darle un valor
- No se puede cambiar el valor

let vs var

- En genreal no usar var
- var no tiene scope de bloque
- var se puede redeclarar, let no
- con var no hace falta poner var al declarar la variable

Tipos de datos

- Number
  - Tanto para integers como como flotante
  - valores enteros
- String

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
```

```
let str = `Hello`;
let phrase = `can embed another ${str}`;
let text = `the result is ${1 + 2}`
```

  - No hay tipo catacter
- boolean

```
let nameFieldChecked = true;
let ageFieldChecked = false;
```

  - Pueden ser el resultado de una comparación

```
let isGreater = 4 > 1;
```

- Null

```
let age = null;
```

    - No es una referencia a un objeto que no existe
    - valor especial que representa nada, vacío, valor desconocido
    - typeof(null) devuelve object
- undefined
    - Representa que no se ha asignado ningún valor
        - Declaración

```
let age;
console.log(age); //undefined
let nombre = undefined; //No deberíamos hacer esto
```

- conversiones
    - console.log(String("23")) // '23'
    - console.log(Number("23") + 1) // 24
    - Conversion a numer usando +

```
let apples = "2";
let oranges = "3";
console.log(+apples + +oranges); //5
```

        - Conversión boolean
            - console.log(Boolean(0)) //false
            - console.log(Boolean(1)) //true
        - Conversiones explícitas
            - console.log("2" + 2) //22
                - concatena

Operadores

- Suma +
- Resta -
- Multiplicación
- División /
- Resto %
    - console.log(5 % 2) //1
- Exponente **
    - console.log(2 ** 3) //8
- Incremento/Decremento

    - ++
    - --

Operadores de bit

- AND &
- OR |
- XOR ^
- NOT ~
- Left Shift <<
- Right Shift >>
- Zero-Fill Right Shift >>>

Comparaciones

- Mayor que >
- Menor que <
- Mayor o igual que >=
- Menor o igual que <=
- Igual ==
- Distinto !=

String comparaciones

- Letra a letra
- Segun el orden Unicode
- console.log('Z' > 'A') //true

Tipos distintos

- Se convierten a números
    - console.log('2' > 1)//true
- Si no se pueden convertir a número devuelven false
    - console.log('Hola' > 34) //false
- Para que no haga la conversion usar la igualdad estricta ===
    - console.log('01' === 1) //false

Condiciones if

- Se evalúa la expresión y se convierte a boolean
-

```
if (year == 2015) console.log('You are right!');
----
if (year == 2015) {
```

```
  console.log('That's correct!');
  console.log('You're so smart!');
} else if (year == 2016){
  console.log();
} else {
  console.log();
}
```

Operador ?

```
let accessAllowed = (age > 18) ? true : false;

let message = (age < 3) ? 'Hi, baby!' : (age < 18) ? 'Hello!' :
(age < 100) ? 'Greetings!' :
'What an unusual age!';
```

Operador lógicos

- OR ||
- Se evalúa de izq a derecha
- Devuelve el primer valor del primer operando que se evalúe a true
- Si se llega al final, se devuelve el valor del último operando

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";
console.log( firstName || lastName || nickName || "Anonymous"); // SuperCoder
```

- AND &&
- Se evalúa de izq a derecha
- Devuelve el primer valor del primer operando que se evalúe a false
- Si se llega al final, se devuelve el valor del último operando

```
alert(1 && 0) // 0
```

- NOT!
  - Convierte el operando a booleano
  - Devuelve el valor inverso
  - alert(!true) //false
  - 0 false
  - 1 true

- Switch
  - Reemplaza múltiples if con igualdad estricta ===
  - Empieza ejecutando desde la primera condición true
  - break para terminar la ejecución del switch
  - default equivalente al else

```
switch (a) {
  case 4:
    console.log( 'Solo se ejecuta el 4' );
    break;
  case 5:
console.log( 'Se ejecuta el 5 y el default' );
    break;
  default:
    console.log( "Default" );
}
```

Bucles

- while se ejecuta mientras la condición sea true

```
let i = 3;
while (i) {
    alert( i );
    i--;
}

------

let i = 3;
while(i) alert(i--)
```

Do while

- se ejecuta al menos una vez

```
let i = 0; do {
  console.log( i );
  i++;
} while (i < 3);
```

for

```
for (let i = 0; i < 3; i++) {
  console.log(i);
}
```

```
console.log(i) //Reference Error i is not defined eee

------

let i = 0;
for (; i < 3;) {
  alert( i++ );
}
```

break

- fuerza de terminación del bucle

```
let sum = 0; while (true) {
  let value = +prompt("Enter a number", '');
  if (!value) break;
  sum += value;
}
console.log( 'Sum: ' + sum );
```

Continue

- fuerza al bucle a seguir con la siguiente iteración

```
for (let i = 0; i < 10; i++) {
    if (i % 2 == 0) continue; console.log(i); // 1, then 3, 5, 7, 9
}
```

Etiquetas

- Labels
- identifica el bucle
- Referencias para break / continue
- No se pueden usar para salar a cualquier lado

Funciones

- Principales bloques de construcción
- Reutilización de código
- Las variables declaradas dentro no son accesibles fuera

```
function showMessage() {
  let mensaje = 'Hello everyone!';
  console.log(mensaje);
```

```
}
```

- Las variables exteriores son accesibles y modificables denro

```
let userName = 'John';
function showMessage() {
userName = "Bob";
let message = 'Hello, ' + userName; console.log(message);
}

console.log( userName ); // John
showMessage();
console.log( userName ); // Bob
```

- Si se declara la misma variable dentro y fuera, la de dentro oscurece (shadows) la de fuera
- Sucede lo mismo en condicionales y bucles

```
let userName = 'John';
function showMessage() {
  let userName = "Bob";
  let message = 'Hello, ' + userName;
  console.log(message);
}

console.log( userName ); // John showMessage();
console.log( userName ); // John
```

Parámetros

- Se puede llamar a la función con menos párametros y entonces se consideran undefined

```
function showMessage(from, text) {
  console.log(from + ': ' + text);
}
showMessage('Ann', 'Hello!'); //Ann: Hello!
showMessage("Ann"); //Ann: undefined
```

- Se puede incluir un valor por defecto

```
function showMessage(from, text = "no hay texto") {
  alert( from + ": " + text );
}
showMessage('Ann', 'Hello!'); //Ann: Hello!
showMessage("Ann"); // Ann: no hay texto
```

- El valor por defecto puede ser una llamada a otra función

```
function showMessage(from, text = anotherFunction()) {
  // anotherFunction() only executed if no text given
  // its result becomes the value of text
}
```

Múltiples parámetros

```
function foo() {
  for (let i = 0; i < arguments.length; i++) {
    console.log(arguments[i]); }
}
foo('Hola');
foo(1,2,3,4,5,6,7,8,9);
```

Return

- Para devolver un resyltado en la función
- Si no existe la función devuelve undefined
- Se puede usar vacío (return;) para terminr la ejecución de la funcion

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  }
}
```

Se puede asignar una función a una variable

- Añadir el ; al final

```
let sayHi = function() {
  console.log( "Hello" );
};
console.log(sayHi); //se imprime el código de la función
console.log(sayHi());// Hello\nundefined
```

- Una declaración de función se puede usar antes, una expresión, no

```
sayHi("John"); // Hello, John
saludar("John"); // ReferenceError: saludar is not defined

function sayHi(name) {
alert( `Hello, ${name}` );
}

let saludar = function(name) {
  alert( `Hello, ${name}` );
};
```

Arrows

- Equivalente a funciones lambda
- Funciones anónimas
- Forma concisa de declarar una función

```
let sum = (a, b) => a + b;
----
let sum = function(a, b) {
  return a + b;
};
```

- Pueden usar múltiples líneas con {} en ese caso necesitan return

```
let sum = (a, b) => {
  let result = a + b;
  return result;
};
console.log( sum(1, 2) ); // 3
```

Arrays

- Crear arrays

```
let fruits = ['Apple', 'Banana'];
```

- Acceder a un elemento

```
console.log(fruits[0]);
```

- Longitud

```
console.log(fruits.length);
```

- Recorrerlo

```
fruits.forEach(function(item, index, array) {
    console.log(item, index);
});
```

- Añadir un elemento

```
let newLength = fruits.push('Orange');
```

- Eliminar el último elemento

```
let last = fruits.pop();
```

- Buscar un índice de un elemento

```
let pos = fruits.indexOf('Banana');
```

- Eliminar un elemento

```
let removedItem = fruits.splice(pos, 1);
```

Objetos

- Almacenar colleciones de varios datos

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

- Se pueden inicializar con datos metiante clave: valor separados por coma

```
let user = {
    name: "John",
    age: 30
};
```

- Para acceder a la información

```
console.log(user.name);
console.log(user["name"]);
```

- anadir nuevas propiedades

```
let user = {
  name: "John",
  age: 30
};
user.isAdmin = true;
```

- borrar propiedades con delete

```
delete user.age;
```

- Una propiedad puede tener más de una palabra, pero entonces hay que usar comillas

```
let user = {
    name: "John",
    age: 30,
    "likes birds": true
};
```

- A estas priopedades hay que acceder con [ ]

```
console.log(user["likes birds"]);
console.log(user["age"]);
```

    - Operador in que devuelve true si existe

Recorer objetos

```
let user = { name: "John", age: 30, isAdmin: true };
for (let key in user) {
  alert(key);  // name, age, isAdmin
  alert(user[key]); // John, 30, true
}
```

- Se pueden anidar los objetos

```
let user = {
  name: "John",
  birthday: {
        year: 1990,
        month: "November"
  }
};
console.log(user.birthday.year);
console.log(user["birthday"]["month"]);
```

- API
    - Hay un montón de interfaces y APIs predefinidas
    - Propiedad onload
        - Para cuando el recurso se ha cargado
    - Propiedad onclick

Callback

- Funciones pasadas como argumentos a otra función
- Se invocan cuando se completa alguna acción o rutina
- Pueden ser síncronas
- Habitualmente se usan asíncronamente

```
function greeting(name) {
  alert('Hello ' + name);
}
function processUserInput(callback) {
  let name = prompt('Please enter your name.');
  callback(name);
}
processUserInput(greeting);
```

- Teclado
  - Eventos del teclado
    - Keydown
      - Cuando se pulsa una tecla
    - Kepress
      - Mientras se pulsa
    - Keyup
      - Cuando se libera la tecla

```
document.addEventListener('keydown', logKey);
function logKey(e) {
  console.log(e.code);
}
```

- Ratón
  - Eventos del ratón
    - Click
      - Cuando se pulsa y luego se libera el botón del ratón
    - dbClick
      - doble click
    - Mouseup
      - cuando se liebra el boton del ratón
    - Mousedown
      - cuando se pulsa el botón del ratón
    - Mousemove
      - mientras se mueve el raton

```
document.addEventListener('click', click);
function click(e) {
  console.log(e);
}
```

Canvas

- Elemento HTML quen nos permite dibujar gráficos

```
<canvas id="myCanvas" width="500" height="500">fallback content</canvas>
```

- Por defecto es un réctangulo blanco
- Necesitan la etiqueta cierra

- Para trabajar con canvas con JS necesitamos el contecto 2d

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
```

```
<!DOCTYPE html>
    <html>
    <head>
          <meta charset="utf-8">
      <title>Canvas</title>
    </head>
    <body>
    <canvas width="500" height="500" id="myCanvas"></canvas>     <script
type="text/javascript">
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
</script>
</body>
</html>
```

jQuery

- Librería JavaScript
- Fast, small and feature-rich
- Recorrer y manipular HTML y CSS
- Gestión de eventos
- Efectos y animaciones

Introducción

- Se puede descargar
- Enlace a un CDN
    - Añadirlo al final del body
- $(document)
- $("p")
- $("p").css("background-color", "yellow");