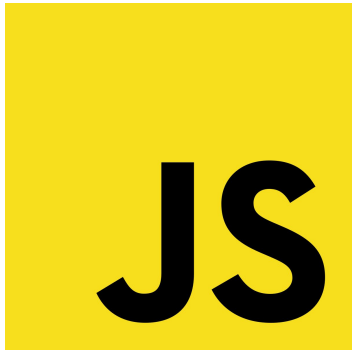
Introducción a Javascript



Qué es JavaScript?

- Lenguaje de programación multiplataforma
  - Web (JS)
    - Para manipular el DOM (Document Object Model). Aplicaciones del lado del cliente.
  - Servidor (Node.JS)
    - Para desarrollar aplicación del lado del servidor
- Tiene alguna relación con Java?
  - Java utiliza un modelo de clases, fuertemente tipado.
  - Un programa típico de Java, consta exclusivamente de clases y sus métodos.
  - Javascript es un lenguaje de scripting, con sintaxis más sencilla y funcionalidad especializada incorporada.
  - Javascript es un lenguaje más libre, no requiere definir interfaces ni es necesario tipificar las variables, parámetros y retornos.
- Concepto de ECMAScript

Instalación de Node.js

https://nodejs.org/en/

Descarga de versión LTS (Long stable version)

- Concepto de npm
- Comando npm init
- Que es package.json?
- Que es package-lock.json

Declaración de variables (1/3)

- Javascript es case-sensitive.
- Se recomienda la utilización de punto y coma luego de cada declaración, aunque no es obligatorio.
- Existen 3 tipos de declaraciones:
  - var -> para variables globales, que no necesitan una asignación.
  - let -> para variables locales, que no necesitan una asignación.
  - const -> para constantes de solo lectura, necesitan asignación.
- Los nombres de las variables pueden empezar con letras, guion bajo o signo dólar. No pueden comenzar con números pero si tenerlos luego del primer carácter.
- Tipos de datos: booleanos (true o false), null, undefined, Number, BigInt, String, Object
- Javascript es tipado dinámicamente (no se necesita especificar tipos)

Declaración de variables (2/3)

- Utilización de console.log
- Concepto de undefined y de null
  - undefined se comporta como false en contexto booleano, pero como NaN en contextos numéricos
  - null se comporta como false en contexto booleano, pero como 0 en contextos numéricos.
- Ámbitos (scope):
  - Dependiendo el tipo de declaración y el ámbito, la declaración será válida o no. Por ejemplo

```
if (true) {
  var x = 5;
}
console.log(x); // x es 5
```

```
if (true) {
  let y = 5;
}
console.log(y); // ReferenceError: y no está definida
```

Declaración de variables (3/3)

- Concepto de hoisting (elevación)
  - Las declaraciones de las variables “var” se “elevan” a la parte superior de la función o declaración.
  - Las variables “const” y “let” no se inician al hacer esto, por lo tanto se obtiene un ReferenceError.
  - Con las funciones pasa lo mismo (con sus declaraciones pero no con sus expresiones)
- No se puede declarar una función con el mismo nombre que una constante en el mismo ámbito.
- Los strings se pueden declarar con comillas simples o dobles si es constante, pero también se pueden declarar de la forma:

  `Esto es un template string ${variable1}`

Control de flujo y manejo de errores

- Una declaración de bloque, es un grupo de instrucciones delimitada por llaves { }
  - Se utilizan en funciones, expresiones condicionales o ciclos. Por ejemplo:

```
while (x < 10) {
  x++;
}
```

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

```
if (condition_1) {
  statement_1;
} else if (condition_2) {
  statement_2;
} else if (condition_n) {
  statement_n;
} else {
  statement_last;
}
```

Control de flujo y manejo de errores

- = vs == vs ===
  - = es un operador de asignación, y se utiliza para asignar variables
  - == es un operador de comparación, e ignora el tipo de las variables
    - Por ejemplo 1 == “1”
  - === es un operador de comparación, pero no ignora el tipo
    - “1” === “1” -> true
    - 1 === “1” -> false
- OJO: Los operaciones de comparación no sirven para Arrays y Objetos. Los arrays son de tipo Objeto, y por lo tanto se comparan por referencia, no por valor. Para comparar arrays y objetos existen funciones nativas que veremos más adelante.

Control de flujo y manejo de errores (1/2)

- = vs == vs ===
  - = es un operador de asignación, y se utiliza para asignar variables
  - == es un operador de comparación, e ignora el tipo de las variables
    - Por ejemplo 1 == “1”
  - === es un operador de comparación, pero no ignora el tipo
    - “1” === “1” -> true
    - 1 === “1” -> false
- OJO: Los operaciones de comparación no sirven para Arrays y Objetos. Los arrays son de tipo Objeto, y por lo tanto se comparan por referencia, no por valor. Para comparar arrays y objetos existen funciones nativas que veremos más adelante.

Control de flujo y manejo de errores (2/2)

- “throw” es una expresión para lanzar una excepción. Se debe acompañar del valor que se lanzará. En JS se pueden lanzar prácticamente todos los tipos como excepción, pero existen algunos tipos de excepción ya definidos (como “Error”)
- La declaración try / catch marca un bloque de expresiones, donde se detecta y manejan las excepciones que ocurran dentro del ámbito del “try”
    - Utilización de finally.

```
openMyFile();
try {
  writeMyFile(theData); // Esto puede arrojar un error
} catch (e) {
  handleError(e); // Si ocurrió un error, manéjalo
} finally {
  closeMyFile(); // Siempre cierra el recurso
}
```

Funciones

Se pueden crear de dos formas:

```
function nombre(parametro1, parametro2) {          -> función por declaración

}


const nombre = (parametro1, parametro2) => {       -> función por expresión

}
```

Arrays (1/2)

- La longitud y tipo de un array en javascript es variable.
- let frutas = ["Manzana", "Banana"];
- Operaciones habituales:
  - frutas.length
  - frutas[0]
  - frutas[frutas.length - 1]
  - frutas.pop()
  - frutas.shift()
  - frutas.unshift(“pera”)
  - frutas.push(“kiwi”)

Arrays (2/2)

- Recorrido de arrays
  - frutas.forEach()
  - frutas.map()
- Para encontrar un elemento:
  - frutas.find()
  - frutas.filter() -> para varios elementos que cumplan cierta condición
- Para ver si un elemento existe:
  - frutas.includes()

Objetos

- Se pueden declarar de dos formas:
  - const objeto = new Object();
  - const objeto = {} -> notación literal
- const player = {
  name: "Manz",
  life: 99,
  power: 10,
};

- Para acceder a sus propiedades:
  - console.log(player.name);
  - console.log(player["name"])

- Las propiedades de los objetos también pueden ser funciones
- const user = {
  name: "Manz",
  talk: function() { return "Hola"; }
};
- user.talk()

Trabajar con varios archivos

- En javascript existen varios **sistemas de módulos.**
  - Es decir, existen distintas formas de importar de otros archivos.
  - Los más populares son CommonJS (CJS) y ES Modules (ESM)
- CommonJS -> forma más antigua

```
// module-name.js
module.exports = {
  /* ... */
}

// index.js
const module = require("./module-name.js");
const package = require("package");
```

Trabajar con varios archivos

- ESM -> Introducida en 2015

```
// module.js
export const data = 42;
export const method = () => console.log("Hello");

// index.js
import { data, method } from "./module.js";
```

- Además de exportar múltiples funciones, clases o variables, se puede utilizar “export default”

```
export default function cube(x) {
  return x * x * x;
}

import cube from "./my-module.js";
```

Utilización de librerías

- Ejemplo de librería: **lodash**
  - https://lodash.com/
  - Ejemplo:
    - import _union from ‘lodash/union’
    - _union([2], [1, 2]);
      Tendrá como output [2,1]

Ejercicio práctico

Crear un paquete utilizando nodejs. Definir dos archivos: constants.js y index.js
En constants.js estará definida una lista de personas (representadas como un array de objetos), con propiedades de nombre completo, edad y altura en metros.
En index.js se deberán definir funciones para calcular quién es el más viejo y quien es el más alto, además, deberá definirse una función para que utilizando estos datos, se muestren por consola los dos resultados (utilizando un template string para darle formato).