UD1. Javascript

ELEMENTOS DEL LENGUAJE

Operadores de comparación

Operator	Description	Example	Result
==	Equal to	1 == 1	true
===	Equal in value and type	1 === '1'	false
!=	Not equal to	1 != 2	true
!==	Not equal in value and type	1 !== '1'	true
>	Greater than	1 > 2	false
<	Less than	1 < 2	true
>=	Greater than or equal to	1 >= 1	true
<=	Less than or equal to	2 <= 1	false

Estructuras de control

```
if ( a === 1 ) { ... } else { … }
var h = a < b ? 5 : 10 ;
for (let i = 0 ; i < 10 ; i++) {...}
while ( i <= 10 ) {...}
do {...} while (i <= 10)
```

Comunicación con el usuario

- alert, confirm, Prompt: https://www.w3schools.com/js/js_popup.asp

- console.log(), .error(), .warm(), debug(), info()
  - console.log("%s is %d years old.", "Bob", 42)
  - console.log("%cThis is green text on a yellow background.", "color:green; background-color:yellow");
  - console.dir()
- https://developer.mozilla.org/es/docs/Web/API/Console

- CSS en la consola: https://javascript.plainenglish.io/a-prettier-console-log-786f46d0bc3c https://javascript.plainenglish.io/adding-css-to-console-log-dde2e167ee7a

Template Literals

```
console.log('"We don\'t make mistakes. We just have happy accidents." - Bob Ross'); // escapando la ''
console.log("\"We don't make mistakes. We just have happy accidents.\" - Bob Ross"); // escapando las ""
console.log(`"We don't make mistakes. We just have happy accidents." - Bob Ross`); // escapando con `
console.log('Homer J. Simpson\n' + '742 Evergreen Terrace\n' + 'Springfield'); // Multiples líneas
console.log(`Homer J. Simpson
742 Evergreen Terrace
Springfield`); // Con ` se puede hacer literal
let a = 2; console.log('La variable a vale: '+a); // concatenando string y número
console.log('La variable a vale:',a); // log acepta varios argumentos
console.log(`La variable a vale: ${a}`); //La mejor forma, con ${}
console.log(`${host}/login/oauth/authorize?client_id=${clientId}&scope=${scope}`); // mejor para muchas
variables
let edat = 19; console.log(`El alumno es: ${edat < 18 ? 'menor' : 'mayor' }`) // se puede insertar una
expresión
```

Arrays

- Se implementan mediante **[ ]**, separado por comas.
- Se puede acceder a un elemento com en C o Java: a[0] = 1;
- No es preciso definir la longitud en su declaración.
- Pueden tener cualquier tipo de datos, incluso otros arrays y objetos o funciones.
- Un array es un objecto y se puede construir con new:

```
var coches= new Array (“Ford", "Volvo", “Mercedes");
```

- Métodos interesantes (objetos):
  - a.length
  - a.sort()
  - a.push()

https://www.w3schools.com/js/js_arrays.asp

Recorrer Arrays

- `for (let i =0; i< a.length; i++){ console.log(a[i]);}`
- `for (let i of a){console.log(i);}`
- `a.forEach(i => console.log(i))`

Buscar en Arrays

```
const alligator = ["thick scales", 80, "4 foot tail", "rounded snout"];

alligator.includes("thick scales"); // devuelve true
alligator.find(el => el.length < 12); // devuelve '4 foot tail'
alligator.find((el, idx) => typeof el === "string" && idx === 2); // devuelve '4 foot tall'
alligator.indexOf("rounded snout"); // devuelve 3
alligator.filter(el => el === 80); // devuelve [80, 80]
```

El método **.includes()** devuelve un valor **booleano** y es ideal para indicarle si un elemento **existe** o **no** en una matriz.

Buscar en Arrays

```
const alligator = ["thick scales", 80, "4 foot tail", "rounded snout"];

alligator.includes("thick scales"); // devuelve true
alligator.find(el => el.length < 12); // devuelve'4 foot tail'
alligator.find((el, idx) => typeof el === "string" && idx === 2); // devuelve'4 foot tall'
alligator.indexOf("rounded snout"); // devuelve 3
alligator.filter(el => el === 80); //devuelve[80, 80]
```

Esta sencilla función en nuestro método **find** busca **cada elemento de la matriz**, con el alias de **'el'** que se le asigne y se detiene cuando encuentra el primero que sea verdadero. En nuestro caso, verdadero tiene una propiedad de **longitud inferior a 12** (los números **no** tienen una propiedad de longitud).

Buscar en Arrays

```
const alligator = ["thick scales", 80, "4 foot tail", "rounded snout"];

alligator.includes("thick scales"); // devuelve true
alligator.find(el => el.length < 12); // devuelve'4 foot tail'
alligator.find((el, idx) => typeof el === "string" && idx === 2); // devuelve'4 foot tall'
alligator.indexOf("rounded snout"); // devuelve 3
alligator.filter(el => el === 80); //devuelve[80, 80]
```

indexOf() es un método muy útil. Puede decirle dónde se encuentra el elemento en la matriz y puede indicar si dicho elemento existe. Podemos saber que el elemento existe si devuelve un número positivo y si devuelve -1, indicaría que el elemento no existe.

Buscar en Arrays

```
const alligator = ["thick scales", 80, "4 foot tail", "rounded snout"];

alligator.includes("thick scales"); // devuelve true
alligator.find(el => el.length < 12); // devuelve'4 foot tail'
alligator.find((el, idx) => typeof el === "string" && idx === 2); // devuelve'4 foot tall'
alligator.indexOf("rounded snout"); // devuelve 3
alligator.filter(el => el === 80); //devuelve[80, 80]
```

El método **filter()** es como el método **find()**, en el sentido de que requiere una función pasada y una condición para lo que se devolverá. La principal diferencia es que **filter()** siempre **devuelve una matriz**, incluso si solo hay un elemento que coincida. Pero devolverá todos los **elementos que coincidan**, mientras que find() solo devuelve la primera coincidencia.

Otras operaciones con Arrays

- splice() → Elimina elementos o agrega nuevos a partir de una posición. (Modifica un array)
- slice() → Extrae una porción del array (No lo modifica, devuelve un nuevo array)
- flat() → Convierte un array multidimensional en un array de menor dimensiones.
- flatMap() → Aplica una funció a cada elemento y quita una dimensión al array. (no lo modifica)
- join() → Transforma un array en una cadena.
- split() → Transforma una cadena en un array.

Use strict

- **use strict**, **class**, **let**… son funcionalidades de ES6 para hacer un JS más parecido a los **lenguajes tradicionales**.
- No se pueden utilizar variables **no** declaradas.
- **Obliga** a utilizar **var**, **let**, **const**.
- Puede estar en una **función** o de forma **global**.
- **No** permite utilitzar **this.** en funciones fuera de objetos.
- **No** es necesario si utilizamos **módulos**.