TXULETA

.splice => (índice actuación, cantidad de elementos a eliminar, elementos a usar)

> **(i, 1)** => elimina i;
> **(y, 0)** => añade y al inicio
> **(i, 0, z)** => añade z a la posición de i

.includes(“subcadena”) => comprueba subcadena dentro de cadena
.concat => concatena uno o varios arrays (preferible usar **operador spread**)
.indexOf*(“subcadena”)* => Devuelve EL PRIMER ÍNDICE que coincide con la subcadena
.lastIndexOf(“subcadena)
.trim() => Importante no olvidar los paréntesis => Elimina espacios. Recomendable para verificación de entrada de usuario
.split(“delimitador”) => Recomendable usar cadena vacía para separar cadena en un array de palabras
.reduce => Usado para sumatorias

```
const calcularPromedio = (notas) => {
   return notas.reduce((suma, nota) => suma + nota) / notas.length;
};
```

Para usar .reduce con array de objetos, se le pasa como argumento el objeto, y junto al acumulador una variable iterable de objeto

```
const calcularNotaMedia = (estudiantes) => {
   let suma = estudiantes.reduce((acumulador, estudiante) => (
       acumulador + estudiante.nota), 0);

   let media = suma / estudiantes.length;
   return media;
};
```

.push() => Añade al final
.pop() => Elimina último valor y lo devuelve
.shift() => Elimina el primer valor y lo devuelve
.unshift => Agrega al principio

Nullish coalescing operator

> Abreviación => ??=
>
> Si no está definida la variable, le atribuye el valor a la derecha

nom ??= “Pepe”;

Flechas

const flecha = (variables) => { return res };

console.log(flecha(variables));

Parámetros REST

Usados dentro de función => ***engloba número indefinido de variables*** (SIEMPRE deben ir al final si van acompañados de otras variables)

Operador spread

=> Duplica un array => let copiaArray = [...arrayOriginal]

=> “Esparce” los valores de un array (si pasamos un array a una operación, evalúa el array como tipo, no lo que incluye) => NaN

Desestructuración de objetos

Asigna propiedades de objetos a variables, siguiendo la correspondencia de los nombres de las propiedades.

Funciona con string!!!!!! :))))

let [a, b, c] = “abc” => a = a; b = b; c = c

```
let [...arrayPalabra] = palabra;
```

Puedes crear un objeto directamente =>

let objetoUsuario = {};

[objetoUsuario.nombre, objetoUsuario.apellido] = (“John”, “Smith”)

.forEach =>

productos.forEach(cosas, funciones, magia);

Importante => productos.forEach(producto, **índice**); **Usa el índice =)**

```
jugador.mochila.forEach((item, indice) => {
   console.log(`Item ${indice + 1}: ${item.nombre} | Precio: ${item.precio}`);
});
```

Si usas {} => Necesita return, puedes usar condicionales y bloques de código en general

Si usas () => El return es explícito, solo usa una expresión

Otros =>

.filter => filtra según condición
.some => Al menos una condición se cumple
.every => Todas las condiciones se cumplen

Ejemplo

```
const filtrarProductos = (productos) => {
   return productos.filter((producto) => producto.stock > 0);
};
```

.map() =>

Devuelve un array con los elementos resultantes de la función aplicada a cada elemento del array original

Ejemplo

```
let progreso = arrayPalabra.map(() => "_");
```

Se puede usar sin parámetro para casos como este en el que a cada índice lo sustituye con barra baja