Bienvenides a fundamentos de programacion, capitulo 5!

En esta clase vamos a profundizar en las estructuras de datos array y object

Arrays

Las distintas formas de inicializar un array:

```
const arr1 = []
// Inicializa un array vacio

const arr2 = [1, 2, 3]
// Inicializa un array con 3 elementos

const arr3 = new Array()
// Inicializa un array vacio (igual que arr1)

const arr4 = new Array(10)
// Inicializa un array de 10 elementos, pero esos elementos son undefined
```

Métodos de arrays

Así como vimos que para agregar y sacar elementos de un arreglo tenemos los métodos **push** y **pop** respectivamente, los arrays cuentan con una gran variedad de métodos que son muy útiles, vamos a ver las más usadas.

> forEach

Esta método nos permite **recorrer** el array, ejecutando una función por cada elemento.

Ej: Imprimir todos los elementos de un array dado, indicando en que posición está cada uno

```
const arr1 = ["a", "b", "c"];

function imprimirElemento(element, index) {
    console.log(`En la posición ${index} se encuentra el elemento ${element}`)
}

arr1.forEach(imprimirElemento)
```

Como podemos ver, el método **forEach** recibe como parametro una función, esa función la va a llamar internamente el método forEach por cada elemento del array y va a recibir por parametros el elemento en sí y la posición de ese elemento en el array. También lo podríamos hacer así:

También lo podríamos hacer así:

```
const arr1 = [1, 2, 3];

arr1.forEach(function(element, index) {
    console.log(`En la posición ${index} se encuentra el elemento ${element}`)
})
```

Así lo vamos a encontrar más comunmente, pero para que sea un poco más facil de entender, por el momento vamos a seguir usando la forma anterior, es decir, declarando una función previamente.

> map

El método **map** sirve para "transformar" cada elemento del array en otra cosa, dando como resultado un nuevo array con todos los elementos transformados.

Ej: Transformar a string todos los números dentro del array

```
const arr1 = [1, 2, 3]

function transformarElemento(element) {
    return element.toString();
}

const arr2 = arr1.map(transformarElemento)
```

Notemos que el método **map** no modifica el array original, sino que devuelve uno nuevo. Por otro lado, la función **transformarElemento** solo estamos usando el parametro **element**, como en este caso no es necesario usar el **index**, ni lo ponemos. Además, la función tiene que retornar el elemento transformado.

> filter

El método **filter** sirve para obtener un sub-array solo con los elementos que cumplan una cierta condición.

Ej: Quedarse solo con los números mayores a 10

```
const arr1 = [11, 2, 4, 20, 4, 10, 99, 1, 3]

function elementoMayorADiez(element) {
    return element > 10
}

const arr2 = arr1.filter(elementoMayorADiez)
```

Notemos que el método **filter** no modifica el array original, sino que devuelve uno nuevo. Por otro lado, la función **elementoMayorADiez** tiene que retornar **true** si el elemento cumple la condición de filtrado, o **false** en caso contrario.

> find

El método **find** sirve para encontrar un elemento en un array

Ej: buscar el elemento "abc" en el array

```
const arr1 = ["qwe", "zxc", "abc", "xyz"]

function esElementoBuscado(element) {
    return element === "abc"
}

const elementoBuscado = arr1.find(esElementoBuscado)
```

Notemos que el método **filter** retorna el elemento buscado (**undefined** si no encuentra nada). Por otro lado, la función **esElementoBuscado** tiene que retornar **true** si el elemento cumple la condición de búsqueda, o **false** en caso contrario.

> includes

El método **includes** sirve para determinar si un array contiene un elemento dado o no.

Ej: Determinar si el arreglo contiene el string "abc" y "asd"

```
const arr1 = ["qwe", "zxc", "abc", "xyz"]

const contieneABC = arr1.includes("abc") // true
const contieneASD = arr1.includes("asd") // false
```

Notemos que el método **includes** retorna **true** o **false**.

> some

El método **some** sirve para determinar si ALGUNO de los elementos del array cumple una cierta condición

Ej: determinar si el array contienie algun número mayor a 10

```
const arr1 = [11, 2, 4]

function elementoMayorADiez(element) {
    return element > 10
}

const algunoEsMayorADiez = arr1.some(elementoMayorADiez) // true
```

Notemos que el método **some** retorna **true** o **false**.

> every

El método **every** sirve para determinar si TODOS de los elementos del array cumplen una cierta condición

Ej: determinar si el array contienie algun número mayor a 10

```
const arr1 = [11, 2, 4]

function elementoMayorADiez(element) {
    return element > 10
}

const todosSonMayorADiez = arr1.every(elementoMayorADiez) // false
```

Notemos que el método **every** retorna **true** o **false**.

> slice

El método **slice** sirve para "cortar" un pedazo del array y así obtener un sub-array

Ej: obtener los primeros 3 elementos del array

```
const arr1 = ["a", "b", "c", "d", "e", "f"]

const arr2 = arr1.slice(0, 3)
```

Notemos que el método **slice** no modifica el array original, sino que devuelve uno nuevo. **slice** recibe dos números, el primero indica a partir de qué índice queremos empezar a "cortar"; el segundo determina cuantos elementos queremos incluir en nuestro corte.

> reduce

El método **reduce** sirve para obtener un único resultado a partir de procesar todos los elementos de un array

Ej: obtener la suma de todos los elementos del array

```
const arr1 = [10, 2, 6, 8, 3]

function sumarElementos(sumaAcumulada, elemento) {
    return sumaAcumulada + elemento
}

const suma = arr1.reduce(sumarElementos, 0)
```

Notemos que el método **reduce** devuelve un único valor. La función **sumarElementos** recibe el resultado parcial y el elemento como parametro. El método **reduce** no solo recibe una función como parametro, sino tambien un valor de inicialización para el resultado parcial.

...y muchas más

Estos son las métodos más comunes para arrays, pero hay bastantes más, para más información visitar https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array

Encadenar métodos

Todos aquellos métodos que devuelvan un nuevo array, pueden ser encadenadas

Ej: Dado un array de números, obtener un nuevo array con todos los números mayores a 10 transformados a string

```
const arr1 = [12, 2, 6, 83, 3, 23, 4, 52]

function esMayorADiez(element) {
    return element > 10
}

function pasarAString(element) {
    return element.toString()
}

const arr2 = arr1.filter(esMayorADiez).map(pasarAString)
```

Métodos útiles para objetos

Breve repaso sobre como inicializar objetos

```
const obj1 = {} // Inicializa objeto vacío

const obj2 = {
    nombre: "pepe",
    edad: 99
} // Inicializa objeto con atributos

obj2.equipo = "boca" // Agrega un atributo al objeto

console.log(obj2)
```

Object.keys

La clase Object tiene el método estático **keys** que nos permite obtener un array con todos los nombres de los atributos de un objeto

```
const obj = {
    nombre: "pepe",
    edad: 99,
    otraKey: "valor"
}

const keys = Object.keys(obj)

console.log(keys) // ["nombre", "edad", "otraKey"]
```

Notemos que al ser un método estático de la clase Object, lo tenemos que llamár así `Object.keys`. Más adelante cuando veamos programación orientada a objetos, vamos a entender esto mejor, por el momento dejemoslo pasar.

Object.values

La clase Object tiene el método estático **values** que nos permite obtener un array con todos los valores de los atributos de un objeto

```
const obj = {
    nombre: "pepe",
    edad: 99,
    otraKey: "valor"
}

const values = Object.values(obj)

console.log(values) // ["pepe", 99, "valor"]
```

Notemos que al ser un método estático de la clase Object, lo tenemos que llamár así `Object.values`. Más adelante cuando veamos programación orientada a objetos, vamos a entender esto mejor, por el momento dejemoslo pasar.

Object.hasOwnProperty

Los objetos poseen método **hasOwnProperty** que nos permite saber si el objeto contiene o no un cierto atributo

```
const obj = {
    nombre: "pepe",
    edad: 99,
    otraKey: "valor"
}

const tieneNombre = obj.hasOwnProperty("nombre")
const tieneApellido = obj.hasOwnProperty("apellido")

console.log(tieneNombre) // true
console.log(tieneApellido) // false
```

Break para tomar awita

5'

Spread operator

El operador **spread** permite "separar" los elementos de un array o los atributos de un objeto.

Ej: Agregar un elemento a un array usando spread operator

```
const arr1 = [5, 8, 3]

const arr2 = [...arr1, 4]

console.log(arr2) // [5, 8, 3, 4]
```

Ej: Agregar un atributo a un objeto usando spread operator

```
const obj1 = { nombre: "pepe", edad: 99 }

const obj2 = {...obj1, otraKey: "valor"}

console.log(obj2) // { nombre: "pepe", edad: 99, otraKey: "valor" }
```

Otro ejemplo

La función **Math.max** recibe una cierta cantidad de números y retorna el máximo.

```
Math.max(5, 3, 7, 4, 8, 1) // Retorna 8
```

Pero si queremos obtener el máximo y todos nuestro números están dentro de un array, podemos hacer lo siguiente

```
const numeros = [5, 3, 7, 4, 8, 1]
Math.max(...numeros) // Retorna 8
```

Otro ejemplo con objetos

Con el spread operator tambien podemos modificar atributos ya existentes de un objeto

```
const obj1 = { nombre: "pepe", edad: 99 }

const obj2 = {
    ...obj1,
    edad: 88,
    otraKey: "valor"
}

console.log(obj2) // { nombre: "pepe", edad: 88, otraKey: "valor" }
```

Comparaciones

Para tener en cuenta, si comparamos dos objetos o arrays distintos, por más que tengan los mismos elementos o atributos, SON DISTINTOS.

```
const obj1 = { nombre: "pepe", edad: 99 }
const obj2 = { nombre: "pepe", edad: 99 }

console.log(obj1 === obj2) // false

const arr1 = [1, 2, 3]
const arr2 = [1, 2, 3]

console.log(arr1 === arr2) // false
```

¿const o let?

Como habrán notado, en muchos de los ejemplos, usamos **const** para declarar arrays y objects. Esto permite agregar o sacar elementos/atributos pero no permite asignar un nuevo array/object a la variable

```
const arr = [1, 2, 3]
arr.push(4) // Todo OK
arr = [...arr, 5] // Esto no se puede hacer, tira error

const obj = { nombre: "pepe", edad: 99 }
obj.otraKey = "value"; // Todo OK
obj = {...obj, edad: 88} // Esto no se puede hacer, tira error
```

Resolvamos un ejercicio juntos

Dada un un array de compras, obtener todas las compras realizadas por "pepe" y devolver un arreglo con los nombres de los productos que compró

```
const compras = [
    { usuario: "pepe", producto: "mochila", costo: 1800 },
    { usuario: "maria", producto: "lapicera", costo: 50 },
    { usuario: "juan", producto: "cuaderno", costo: 200 },
    { usuario: "pepe", producto: "carpeta", costo: 300 },
    { usuario: "laura", producto: "escuadra", costo: 150 },
]
```

Resolución:

```
function esCompraDePepe(compra) {
    return compra.usuario === "pepe"
}

function obtenerProducto(compra) {
    return compra.producto
}

const resultado = compras.filter(esCompraDePepe).map(obtenerProducto)
```

Ejercicios para resolver

1. Alumnos aprobados

Dado el siguiente array de alumnos de una clase Obtener un array con los números de alumno de aquellos alumnos que hayan aprobado la materia. Para approbar la matería la nota final debe ser mayor a 6.

```
const alumnos = [
    { numAlumno: "1234/0", nombre: "maria", nota: 7 },
    { numAlumno: "4459/2", nombre: "juan", nota: 3 },
    { numAlumno: "5877/1", nombre: "pepe", nota: 5 },
    { numAlumno: "5512/0", nombre: "josefina", nota: 9 },
    { numAlumno: "9874/0", nombre: "jaime", nota: 7 },
]
```

resultado esperando

```
["1234/0", "5512/0", "9874/0"]
```

2. Stock agotado

Dado el siguiente array de productos en stock, agregar a cada producto un atributo booleano **agotado** en true si el stock es mayor a 0, caso contrario false.

```
const productos = [
    { id: "1", nombre: "lapiceras", stock: 164 },
    { id: "2", nombre: "marcadores", stock: 0 },
    { id: "3", nombre: "cartulinas", stock: 25 },
    { id: "4", nombre: "cartucheras", stock: 0 },
    { id: "5", nombre: "mochilas", stock: 4 }
]
```

resultado esperando

```
[
    { id: "1", nombre: "lapiceras", stock: 164, agotado: false },
    { id: "2", nombre: "marcadores", stock: 0, agotado: true },
    { id: "3", nombre: "cartulinas", stock: 25, agotado: false },
    { id: "4", nombre: "cartucheras", stock: 0, agotado: true },
    { id: "5", nombre: "mochilas", stock: 4, agotado: false }
]
```

3. Jugadores sospechosos

En nuestro juego online tenemos un array de usuarios sospechosos que están usando un item que fue prohibido, obtener la lista de IDs de usuarios que en su inventario tengan dicho item. El item prohibido se llama "Katana de fuego".

```
const jugadores = [
    { ID: "1", clase: "mago", nivel: 35, inventario:
        ["Manzana", "Poción de maná", "Vara mágica"] },
    { ID: "2", clase: "ladron", nivel: 65, inventario:
        ["Daga", "Katana de fuego", "Poción de vida"] },
    { ID: "4", clase: "curandero", nivel: 73, inventario:
        ["Vara mágica", "Armadura ligera"] },
    { ID: "3", clase: "espadachin", nivel: 36, inventario:
        ["Casco de hierro", "Katana de fuego", "Pocíón de velocidad"] },
    { ID: "5", clase: "mago", nivel: 26, inventario:
        ["Carta de PecoPeco", "Oridecon", "Poción de concentración"] },
]
```

resultado esperando

```
["2", "3"]
```

4. Silabas

Nuestro software de reconocimiento de voz funciona de una forma un poco particular, y el texto reconocido nos lo envía en un arreglo de silabas. Dado el siguiente array, obtener un string con todo el texto de corrido.

```
const silabas = ["¡Ho", "la, ", "mun", "do! ", "¿To", "do", "bien?", "Me", "a", "le", "gro."]
```

resultado esperado

```
"¡Hola, mundo! ¿Todo bien? Me alegro."
```

5. Pedidos

Tenemos un array con pedidos para nuestro negocio de comidas. Obtener un array con solo aquellos pedidos que incluyan algun condimento extra.

```
const pedidos = [
    { id: "1", pedido: "hamburgesa", extras: ["mayonesa"] },
    { id: "1", pedido: "pancho" },
    { id: "1", pedido: "pizza" },
    { id: "1", pedido: "pancho", extras: ["ketchup"] },
    { id: "1", pedido: "empanadas" },
]
```

resultado esperando

```
[
    { id: "1", pedido: "hamburgesa", extras: ["mayonesa"] },
    { id: "1", pedido: "pancho", extras: ["ketchup"] }
]
```

6. Usuario y su perfil

Tenemos dos objetos, uno con la información de usuario, y otro con la información personal de dicho usuario, queremos obtener un solo objeto con toda la información.

```
const usuario1 = { id: "123", username: "pepe24" }
const usuario1perfil = { nombre: "pepe", edad: 27, profesion: "programador" }
```

resultado esperado

```
{ id: "123", username: "pepe24", nombre: "pepe", edad: 27, profesion: "programador" }
```