

```
function (datasetsWithSubject) {
    if (datasetsWithSubject.length > 0) {
        subjectAverage = 0;
        datasetsWithSubjectLength = datasetsWithSubject.length;
        datasetsWithSubject.forEach((dataset) => {
            subjectAverage += parseFloat(dataset["average"]);
        });
    }
}
```

JS

Javascript

Variables

Para definir variables que no cambian su valor usamos

```
const
```

Para definir variables que si cambian su valor usamos

```
let
```

Ya no usamos var...

Comandos útiles

Para importar un script usamos el tag

```
<script src="archivo.js"></script>
```

Para mostrar contenido en la consola usamos

```
console.log("hello world")
```

Para obtener el valor de un elemento de HTML usando su Id

```
const myTable = document.getElementById('first-table')
```

Generación de HTML

Si queremos insertar filas a una tabla

```
newRow = myTable.insertRow(0)
```

Eso inserta una fila hasta arriba, para ponerla al final podemos usar el índice -1.

Para crear una celda a esa nueva fila

```
newCell = newRow.insertCell(0)
```

En este caso estamos definiendo la primera columna (índice 0).

Para modificar el texto de un elemento de HTML

```
elemento.innerText = "Hello World"
```

Para modificar el HTML de un elemento

```
elemento.innerHTML = "<h1>Hello World</h1>"
```

Para modificar la clase asociada a un elemento

```
elemento.className = "mi-clase"
```

Map

La función map me permite aplicarle una función (callback) a todos los elementos de una estructura de datos.

```
const modificado = actual.map((a) => ({
  key1: a.val1,
  key2: a.val2,
  ...
}));
```

```
const doble = simple.map((item) => (item * 2))
```

Reduce

La función reduce me permite simplificar una estructura y convertirla en un solo dato

```
const total = resultados.reduce((carry, item) => (carry += item
```

Json

Formato utilizado para transportar y almacenar datos. Significa JavaScript Object Notation.

Es la forma más común de comunicar datos de un servidor a una página web.

Es fácil de entender ya que se describe a sí mismo.

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

Promise API

Set de funcionalidades para el manejo de operaciones asíncronas. Una promesa es un valor que puede estar disponible en este momento, en un futuro o nunca.

Código asíncrono:

```
console.log("UNO")  
  
setTimeout(() => {  
  console.log("DOS")  
}, 5000)  
  
console.log("TRES")
```

Para evitar esto se usan las promesas.

```
const promesa = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    console.log("DOS");  
    resolve();  
  }, 5000);  
};  
  
console.log("UNO");
```

```
promesa.then((result) => {
    console.log("TRES");
});
```

Async/Await

Versión mejorada de usar promesas.

```
function contarDos() {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("DOS");
            resolve();
        }, 5000);
    });
}

async function contar() {
    console.log("UNO");

    await contarDos();

    console.log("TRES");
}

contar();
```

Fetch

Función que se utiliza para realizar llamadas remotas (HTTP requests). Escenario perfecto para utilizar async/await.

```
async function consultarApi() {  
    const response = await fetch("URL", {  
        method: "GET",  
        headers: {  
            "key": "value",  
        },  
    });  
  
    return response.json();  
}
```

Para utilizar esos datos usamos la promesa que nos regresa la función

```
consultarApi().then((datos) => {  
    // Muchas operaciones...  
});
```