

Laboratorio N°3

INFORME JACAScript MÓDULO 3

JUAN URIEL MALAVER RAMÍREZ

Introducción.

En la era digital, dominar lenguajes de programación como JavaScript es esencial para desarrollar aplicaciones web dinámicas e interactivas. Este informe se enfoca en explorar las bases y técnicas de JavaScript, destacando conceptos clave como funciones, promesas, y manipulación del DOM. A lo largo del documento, se presentarán ejemplos prácticos y explicaciones que proporcionarán una comprensión de cómo estas tecnologías se integran para crear experiencias web funcionales.

Objetivos

- Aprender y Aplicar Conceptos de JavaScript: Familiarizarte con las funciones, promesas, y estructuras de control en JavaScript para crear aplicaciones web dinámicas e interactivas.
- Manipulación del DOM: Dominar la interacción con el Document Object Model (DOM) para actualizar dinámicamente el contenido de la página y manejar eventos del usuario.
- Manejo de Datos: Aprender a convertir entre objetos y JSON para el intercambio eficiente de datos y la comunicación con APIs.

Informe

Ejercicio 1 Variables y Tipos de Datos

Práctica con los conceptos básicos de variables y tipos de datos en Javascript.

a) Declara variables usando **let** y **const** para:

✓ Nombre de usuario (String)

Explicación:

Se usa LET para crear la variable llamada nombreUsuario, y se le asigna una cadena de caracteres (String) que siempre van entre comillas "Juan".

Código:

```
let nombreUsuaio = "Juan";
```

✓ Edad (number)

Explicación:

Se usa LET para crear una variable llamada edad y se le asigna un número (number).

Código:

```
let edad = 32;
```

✓ ¿Está activo? (boolean)

Explicación:

Se usa CONST para crear una constante llamada estaActivo y se le asigna un valor boolean (true or false).

Código:

```
const estaActivo = true;
```

- ✓ Crea un objeto simple con datos de un producto:
 - Nombre
 - Precio
 - Disponible.

Explicación:

Se crea una variable producto, con las propiedades nombre, representado por String, precio por number y disponible con boolean true.

Código:

```
let producto = {  
  nombre: "Lápiz",  
  precio: 20,  
  disponible: true  
};
```

Ejercicio 2 Operadores y Estructuras de Control

Práctica el uso de operadores y estructuras condicionales.

a) Crea un programa que:

- ✓ Verifique si un usuario es mayor de edad.

Explicación:

Se crea una variable edad, se le asigna el valor de 16, se crea una condicional if donde su expresión dice que, si edad es mayor o igual a 18, entonces imprima en el caso verdadero que es mayor de edad y en el caso falso que imprima que no es mayor de edad. Como el valor asignado a edad es 16, significa que debe imprimir en consola el caso falso ya que no cumple con la expresión mayor o igual a 18.

Código:

```
let edad = 16;

if(edad >= 18){
  console.log("Es mayor de edad")
}else {
  console.log("No es mayor de edad")
}
```

Imágenes de prueba:

```
> let edad = 16;
< undefined
> if (edad >= 18) {console.log('Es mayor de edad')}
  else{'No es mayor de edad'}
< 'No es mayor de edad'
```

- ✓ Compare dos precios y muestre el más bajo.

Explicación:

Se crean dos variables con su respectivo valor, luego se crea una variable identificada como menor precio, se crea la condicional si una de las variables tiene menos valor que la otra, entonces menor precio es de la primer variable si no el menor precio es de la segunda variable.

Código:

```
const manzana = 1000;
const pera = 1500;

let menorPrecio;
if (manzana < pera ) {
  menorPrecio = manzana;
} else {
  menorPrecio = pera;
}

console.log("El precio más bajo es:", menorPrecio);
```

Imágenes de prueba:

```
> const manzana = 1000;
  const pera = 1500;

  let menorPrecio;
  if (manzana < pera ) {
    menorPrecio = manzana;
  } else {
    menorPrecio = pera;
  }

  console.log("El precio más bajo es:", menorPrecio);
El precio más bajo es: 1000 VM40:11
```

- ✓ Use un bucle para contar del 1 al 5.

Explicación:

Utilizo un bucle FOR que itera desde 1 hasta 5. En cada iteración, se asigna el valor actual de i a la variable contar y luego se imprime contar en la consola. Esto hace que se impriman los números del 1 al 5 de manera secuencial.

Código:

```
for(let i = 1; i <= 5; i++){  
  let contar = i;  
  console.log(contar);  
}
```

Imágenes de prueba:

```
> for(let i = 1; i <= 5; i++){  
  let contar = i;  
  console.log(contar);  
}  
1 VM541:3  
2 VM541:3  
3 VM541:3  
4 VM541:3  
5 VM541:3
```


Ejercicio 3 Funciones.

Practica la función y uso de funciones.

a) Crea las siguientes funciones:

- ✓ Una función tradicional que salude al usuario.

Explicación:

La función saludar toma un parámetro llamado nombre. Cuando se llama a la función con un argumento, en este caso "Juan", imprime en la consola "Bienvenido, Juan!".

Código:

```
function saludar(nombre) {  
    console.log("Bienvenido, " + nombre + "!");  
}  
saludar("Juan");
```

Imágenes de prueba:

```
function saludar(nombre) {  
    console.log("Bienvenido, " + nombre + "!");  
}  
  
saludar("Juan");  
Bienvenido, Juan! VM79:2
```

- ✓ Una función flecha que calcule el total con impuesto

Explicación:

Esta función calculoConImpuesto toma dos parámetros: el precio y el porcentaje de impuesto. Calcula el impuesto y lo suma al precio, devolviendo el total.

Código:

```
const calculoConImpuesto = (precio, impuesto) => {  
  return precio + (precio * (impuesto / 100));  
};  
  
const precio = 100;  
const impuesto = 19;  
let totalConImpuesto = calculoConImpuesto(precio, impuesto);  
  
console.log("El total con impuesto es:", totalConImpuesto);
```

Imágenes de prueba:

```
> const calculoConImpuesto = (precio, impuesto) => {  
  return precio + (precio * (impuesto / 100));  
};  
  
const precio = 100;  
const impuesto = 19;  
let totalConImpuesto = calculoConImpuesto(precio,  
impuesto);  
  
console.log("El total con impuesto es:",  
totalConImpuesto);  
  
El total con impuesto es: 119
```

[VM87:9](#)

- ✓ Una función que reciba un callback simple.

Explicación:

La función `procesarCallback` recibe dos argumentos: un mensaje y un callback. Imprime el mensaje y luego ejecuta el callback, que en este caso es la función `saludoCallback` que imprime "¡Callback ejecutado!".

Código:

```
function procesarCallback(mensaje, callback) {  
  console.log("Mensaje recibido: " + mensaje);  
  callback();  
}  
  
function saludoCallback() {  
  console.log("¡Callback ejecutado!");  
}  
  
procesarCallback("Hola, mundo", saludoCallback);
```

Imágenes de prueba:

```
> function procesarCallback(mensaje, callback) {  
  console.log("Mensaje recibido: " + mensaje);  
  callback();  
}  
  
function saludoCallback() {  
  console.log("¡Callback ejecutado!");  
}  
  
procesarCallback("Hola, mundo", saludoCallback);  
Mensaje recibido: Hola, mundo      VM99:2  
¡Callback ejecutado!              VM99:7  
undefined
```

Ejercicio 4 JSON y Promesas.

Trabaja con JSON y promesas simples.

a) Realiza las siguientes operaciones:

- ✓ Convierte un objeto a string JSON.

Explicación:

La función `JSON.stringify()` toma el objeto usuario y lo convierte en una cadena JSON. Cuando imprimes `usuarioJSON`, obtendrás esto: `{"nombre":"Juan","edad":"32"}`

Código:

```
const usuario = {
  nombre: "Juan",
  edad: "32"
};
//CONVERSIÓN A JSON
const usuarioJSON = JSON.stringify(usuario);
console.log(usuarioJSON);
```

Imágenes de prueba:

```
const usuario = {
  nombre: "Juan",
  edad: "32"
};
//CONVERSIÓN A JSON

const usuarioJSON = JSON.stringify(usuario);
console.log(usuarioJSON);

{"nombre":"Juan","edad":"32"} VM42:8
undefined
```

- ✓ Convierte un string JSON objeto.

Explicación:

La función `JSON.parse()` toma la cadena JSON `userJSON` y la convierte en un objeto. Cuando imprimes `user`, verás el objeto original.

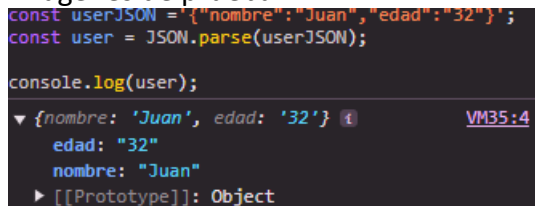
Código:

```
//CONVERSIÓN A JSON

const userJSON = '{"nombre":"Juan","edad":"32"}';
const user = JSON.parse(userJSON);

console.log(user);
```

Imágenes de prueba:



The screenshot shows a code editor with the following code:

```
const userJSON = '{"nombre":"Juan","edad":"32"}';
const user = JSON.parse(userJSON);

console.log(user);
```

Below the code, the console output is displayed as a JSON object:

```
{
  "nombre": "Juan",
  "edad": "32"
}
```

The console also shows the object's prototype as `[[Prototype]]: Object`.

- ✓ Crea una promesa simple que se resuelva después de 2 segundos.

Explicación:

Esta promesa se resuelve después de 2 segundos usando `setTimeout`. Primero, se crea una nueva promesa que espera 2 segundos antes de llamar a `resolve` con el mensaje "Promesa resuelta!". Luego, `miPromesa` usa `then` para manejar el éxito de la promesa. Y se imprime el mensaje en la consola.

Código:

```
const miPromesa = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Promesa resuelta!");
  }, 2000);
});

miPromesa.then((mensaje) => {
  console.log(mensaje);
});
```

Imágenes de prueba:

```
const miPromesa = new Promise((resolve, reject) =>{  
  setTimeout(() => {  
    resolve("Promesa resuelta!");  
  }, 2000);  
});
```

```
miPromesa.then((mensaje) => {  
  console.log(mensaje);  
});
```

▼ *Promise* {<pending>} ⓘ

- ▶ *[[Prototype]]*: *Promise*
- [[PromiseState]]*: "pending"
- [[PromiseResult]]*: undefined

Promesa resuelta!

[VM75:8](#)

Conclusiones

- **Funciones y Promesas:** Hemos explorado cómo crear funciones tradicionales y funciones flecha en JavaScript, así como la implementación de promesas para manejar operaciones asíncronas. Las promesas nos permiten manejar flujos de trabajo asíncronos de una manera más organizada y predecible.
- **Interactividad y Manipulación del DOM:** Al crear una calculadora de calorías, hemos aprendido a capturar datos de formularios y utilizar estos datos para realizar cálculos. Este ejercicio demuestra cómo interactuar con el DOM (Document Object Model) para crear experiencias de usuario dinámicas.
- **Manipulación de Objetos y JSON:** Convertir objetos a cadenas JSON y viceversa es una habilidad crucial para la comunicación con APIs y el almacenamiento de datos en aplicaciones web. Esto nos permite trabajar con datos de una manera más estructurada y eficiente.
- **Estructuras de Control y Bucles:** El uso de estructuras de control como if-else y bucles como for y while nos ha mostrado cómo dirigir el flujo de ejecución de nuestros programas y realizar tareas repetitivas de manera eficiente.

Referencias

Talleres Módulo1, Módulo 2 y Módulo 3

Julian F. Latorre

Bootcamp Desarrollo Web Full Stack