

UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE CIENCIAS PURAS Y NATURALES

INFORMÁTICA



PRACTICA #1

PROGRAMACION WEB 3

Nombre Completo : ADUVIRI ZAPANA ADRIAN JORGE

C.I. : 14878180

Carrera : INFORMATICA

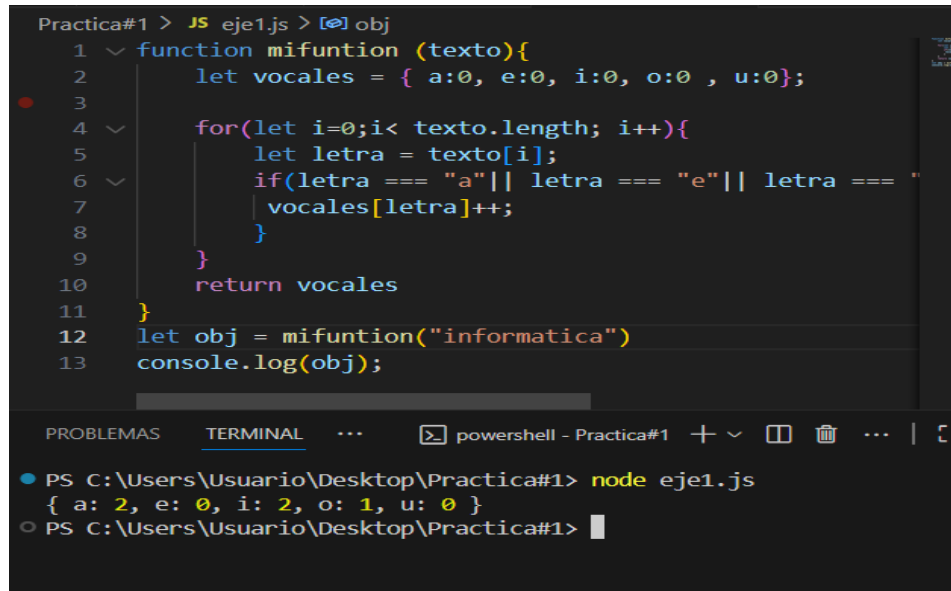
Materia : PROGRAMACION WEB 3

Fecha : 23/09/2025

1. Crear una función que cuente cuántas veces aparece cada vocal en un texto y devuelva el resultado en un objeto.

```
let obj = miFuncion("euforia")
```

```
console.log(obj) // { a: 1, e: 1, i: 1, o: 1, u: 1 }
```



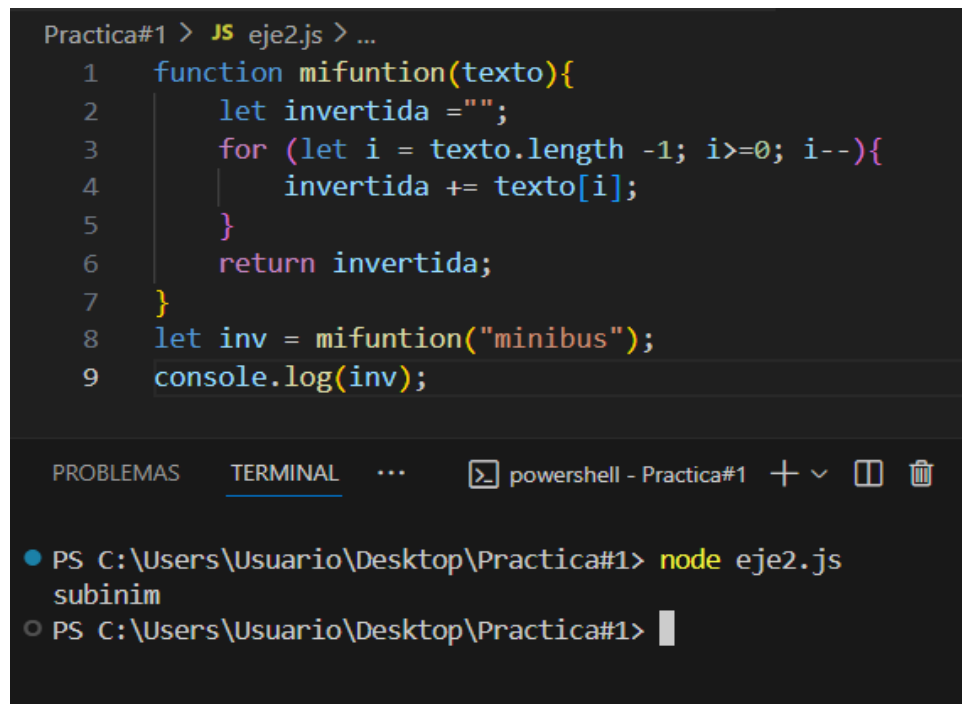
```
Practica#1 > JS eje1.js > [obj]
1  function mifuntion (texto){
2      let vocales = { a:0, e:0, i:0, o:0 , u:0};
3
4      for(let i=0;i< texto.length; i++){
5          let letra = texto[i];
6          if(letra === "a" || letra === "e" || letra === "i" || letra === "o" || letra === "u"){
7              vocales[letra]++;
8          }
9      }
10     return vocales
11 }
12 let obj = mifuntion("informatica")
13 console.log(obj);
```

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje1.js
{ a: 2, e: 0, i: 2, o: 1, u: 0 }
```

2. Crear una función que invierta el orden de las palabras en una frase.

```
let cad = miFuncion("abcd")
```

```
console.log(obj) // dcba
```



```
Practica#1 > JS eje2.js > ...
1  function mifuntion(texto){
2      let invertida = "";
3      for (let i = texto.length - 1; i >= 0; i--){
4          invertida += texto[i];
5      }
6      return invertida;
7  }
8  let inv = mifuntion("minibus");
9  console.log(inv);
```

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje2.js
subinim
```

3. Crear una función que reciba un arreglo de números y devuelva en un objeto a los pares e impares:

```
let obj = miFuncion([1,2,3,4,5])  
console.log(obj) // { pares: [2,4], impares: [1,3,5]}
```

```
Practica#1 > JS eje3.js > [?] obj  
1  function miFuncion(numeros) {  
2      let resultado = { pares: [], impares: [] };  
3  
4      for (let i = 0; i < numeros.length; i++) {  
5          if (numeros[i] % 2 === 0) {  
6              resultado.pares.push(numeros[i]);  
7          } else {  
8              resultado.impares.push(numeros[i]);  
9          }  
10     }  
11  
12     return resultado;  
13 }  
14  
15 let obj = miFuncion([4, 2, 7, 8, 5, 33, 44, 46, 55]);  
16 console.log(obj);  
17
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

- PS C:\Users\Usuario\Desktop\Practica#1> node eje3.js
- { pares: [4, 2, 8, 44, 46], impares: [7, 5, 33, 55] }
- PS C:\Users\Usuario\Desktop\Practica#1> █

4. Crear una función que reciba un arreglo de números y devuelva el número mayor y el menor, en un objeto.

```
let obj = miFuncion([3,1,5,4,2])  
console.log(obj) // { mayor: 5, menor: 1 }
```

```
Practica#1 > JS eje4.js > [?] obj  
1  function miFuncion(numeros){  
2      let mayor = numeros[0];  
3      let menor = numeros[0];  
4  
5      for (let i =1; i<numeros.length; i++){  
6          if(numeros[i]>mayor){  
7              mayor = numeros[i];  
8          }  
9          if(numeros[i]< menor){  
10             menor = numeros[i];  
11         }  
12     }  
13     return {mayor: mayor, menor: menor};  
14 }  
15 let obj=miFuncion([3,6,1,2,7,20]);  
16 console.log(obj);
```

PROBLEMAS TERMINAL ... [?] powershell - Practica#1 + - []

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje4.js  
{ mayor: 20, menor: 1 }  
PS C:\Users\Usuario\Desktop\Practica#1> █
```

5. Crear una función que determine si una cadena es palíndromo (se lee igual al derecho y al revés).

```
let band = miFuncion("oruro")
console.log(band) // true
let band = miFuncion("hola")
console.log(band) // false
```

```
Practica#1 > JS eje5.js > ...
1  function miFuncion(texto){
2      let invertida = "";
3      for(let i= texto.length -1; i>=0;i--){
4          invertida+=texto[i];
5      }
6      return texto === invertida;
7  }
8  let band = miFuncion("programa")
9  console.log(band);
10 band = miFuncion("ana");
11 console.log(band);
```

PROBLEMAS TERMINAL ... powershell - Practica#1 + v []

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje5.js
false
true
PS C:\Users\Usuario\Desktop\Practica#1>
```

6. Tomar los dos primeros elementos de un arreglo y almacenarlos en dos variables mediante desestructuración.

```
Practica#1 > JS eje6.js > ...
1  let arreglo =[17,64,32,12,65,86,91];
2  let[a,b]=arreglo;
3  console.log(a);
4  console.log(b);
```

PROBLEMAS TERMINAL ... powershell - Practica#1 + v []

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje6.js
17
64
PS C:\Users\Usuario\Desktop\Practica#1>
```

7. Almacenar el resto de los elementos de un arreglo sin tomar en cuenta los dos primeros elementos de un arreglo, mediante desestructuración.

```
Practica#1 > JS eje7.js > ...
1  let arreglo = [10, 20, 30, 40, 50];
2
3  let [a, b, ...resto] = arreglo;
4
5  console.log(a);
6  console.log(b);
7  console.log(resto);
```

PROBLEMAS TERMINAL ... powershell - Practica#1 + v □

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje7.js
10
20
[ 30, 40, 50 ]
PS C:\Users\Usuario\Desktop\Practica#1>
```

Callback y Promesas en JS

8. Realizar un código para ejecutar una función callback después 2 segundos.

```
Practica#1 > JS eje8.js > miCallback
1  function miCallback() {
2    console.log("Han pasado 2 segundos");
3  }
4
5  setTimeout(miCallback, 2000);
```

PROBLEMAS TERMINAL ... powershell - Practica#1 + v □

```
● PS C:\Users\Usuario\Desktop\Practica#1> node eje8.js
Han pasado 2 segundos
○ PS C:\Users\Usuario\Desktop\Practica#1>
```

9. Crear una promesa que devuelva un mensaje de éxito después de 3 segundos.

```
Practica#1 > JS eje9.js > ...
1  const miPromesa = new Promise((resolve, reject) => {
2    |   setTimeout(() => {
3    |     |   resolve("Éxito... se resolvió después de 3 segundos");
4    |     |   }, 3000);
5    |   });
6
7  miPromesa.then((mensaje) => {
8    |   console.log(mensaje);
9    |   });
10
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\Usuario\Desktop\Practica#1> node eje9.js
Éxito... se resolvió después de 3 segundos
PS C:\Users\Usuario\Desktop\Practica#1> 
```

10. ¿Cuándo es conveniente utilizar un callback, y cuando es necesario utilizar una promesa?

```
Practica#1 > JS eje10.js > ...
1  /*Callback
2  |  Se usa cuando una función necesita ejecutar otra función después de
3  |  que termine una tarea, sincrónica o asíncrona, y el código es simple.
4  |  */
5  function obtenerDatos(callback) {
6    |   console.log("Obteniendo datos...");
7    |   setTimeout(() => {
8    |     |   callback("Datos recibidos con callback");
9    |     |   }, 2000);
10   |   }
11   |   💡
12   |   // Uso
13   |   obtenerDatos((resultado) => {
14   |     |   console.log(resultado);
15   |     |   });
16   |   }
17   |   /*Promesa
18   |   Se usa para manejar operaciones asíncronas de manera más ordenada, evitando
19   |   el "callback hell" y permitiendo encadenar tareas fácilmente con .then() y .catch().
20   |   */
21   |   function obtenerDatos() {
22   |     |   return new Promise((resolve, reject) => {
23   |       |   console.log("Obteniendo datos...");
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\Usuario\Desktop\Practica#1> node eje10.js
Obteniendo datos...
Datos recibidos con callback
PS C:\Users\Usuario\Desktop\Practica#1> node eje10.js
Obteniendo datos...
Datos recibidos con promesa
PS C:\Users\Usuario\Desktop\Practica#1> 
```

11. Proporcione un ejemplo concreto de encadenamiento de promesas.

```
Practica#1 > JS eje11.js > then() callback
1  function obtenerDatos() {
2      return new Promise((resolve) => {
3          setTimeout(() => {
4              resolve("Datos obtenidos");
5          }, 1000);
6      });
7  }
8
9  function procesarDatos(datos) {
10     return new Promise((resolve) => {
11         setTimeout(() => {
12             resolve(`${datos} → Datos procesados`);
13         }, 1000);
14     });
15 }
16
17 obtenerDatos()
18     .then((resultado1) => {
19         console.log(resultado1);
20     })
21     .then((resultado2) => {
22         console.log(resultado2);
23     })
24     .then(() => {
25         console.log("Proceso finalizado");
26     });

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje11.js
Datos obtenidos
○ Datos obtenidos → Datos procesados
Proceso finalizado
● PS C:\Users\Usuario\Desktop\Practica#1> 
```

12. Proporcione un ejemplo concreto donde el anidamiento de callbacks se puede reescribir mejor con async/await haciendo el código más limpio y mantenible.

```
Practica#1 > JS eje12.js > paso1 > <function>
1  function paso1() {
2      return new Promise((resolve) => {
3          setTimeout(() => {
4              console.log("Paso 1 completado");
5              resolve();
6          }, 1000);
7      });
8  }
9
10 function paso2() {

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS C:\Users\Usuario\Desktop\Practica#1> node eje12.js
Paso 1 completado
Paso 2 completado
Paso 3 completado
Todos los pasos completados
PS C:\Users\Usuario\Desktop\Practica#1> 
```

13. Proporcione un ejemplo concreto donde el anidamiento de promesas se puede reescribir mejor con `async/await` haciendo el código más limpio y mantenible.

```
Practica#1 > JS eje13.js > paso1
1  function paso1() {
2      return new Promise((resolve) => {
3          setTimeout(() => {
4              console.log("Paso 1 completado");
5              resolve("Resultado 1");
6          }, 1000);
7      });
8  }
9
10 function paso2(resultadoAnterior) {

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

● PS C:\Users\Usuario\Desktop\Practica#1> node eje13.js
Paso 1 completado
Paso 2 completado con: Resultado 1
Paso 3 completado con: Resultado 2
Proceso terminado con: Resultado final
○ PS C:\Users\Usuario\Desktop\Practica#1> 
```

14. Proporcione un ejemplo para convertir una promesa en un callback.

```
Practica#1 > JS eje14.js > obtenerDatosPromesa
1  function obtenerDatosPromesa() {
2      return new Promise((resolve, reject) => {
3          setTimeout(() => {
4              const exito = true;
5              if (exito) {
6                  resolve("Datos obtenidos desde la promesa");
7              } else {
8                  reject("Ocurrió un error en la promesa");
9              }
10             }, 1000);
11     });
12 }

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

● PS C:\Users\Usuario\Desktop\Practica#1> node eje14.js
Resultado: Datos obtenidos desde la promesa
○ PS C:\Users\Usuario\Desktop\Practica#1> 
```


15. Proporcione un ejemplo para convertir un callback en una promesa.

```
Practica#1 > JS eje15.js > ...
1  function obtenerDatosCallback(callback) {
2      setTimeout(() => {
3          const exito = true;
4          if (exito) {
5              callback(null, "Datos obtenidos desde el callback");
6          } else {
7              callback("Ocurrió un error en el callback", null);
8          }
9      }, 1000);
10 }
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

- PS C:\Users\Usuario\Desktop\Practica#1> node eje15.js
Resultado: Datos obtenidos desde el callback
- PS C:\Users\Usuario\Desktop\Practica#1> █

16. Proporcione un ejemplo para migrar una función con promesas a async/await.

```
Practica#1 > JS eje16.js > ...
1  // --- Función original que devuelve una promesa ---
2  function obtenerDatosPromesa() {
3      return new Promise((resolve, reject) => {
4          setTimeout(() => {
5              const exito = true;
6              if (exito) {
7                  resolve("Datos obtenidos desde la promesa");
8              } else {
9                  reject("Ocurrió un error en la promesa");
10             }
11         });
12     });
13 }
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

- PS C:\Users\Usuario\Desktop\Practica#1> node eje16.js
Resultado con promesas: Datos obtenidos desde la promesa
Resultado con async/await: Datos obtenidos desde la promesa
- PS C:\Users\Usuario\Desktop\Practica#1> █