

UNIVERSIDAD MAYOR DE SAN ANDRES

CARRERA DE INFORMATICA

Programación Web 3

INF-133



Practica N°1

Apellidos: Apaza Quiroz

Nombre: Oscar

C.I.: 10060807

Fecha: 24 de 09 de 2025

Sigla: INF-133

Docente: Lic. Jhonny Felipez Andrade

LA PAZ - BOLIVIA

2025

Practica N°1

Java Script

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 }
```

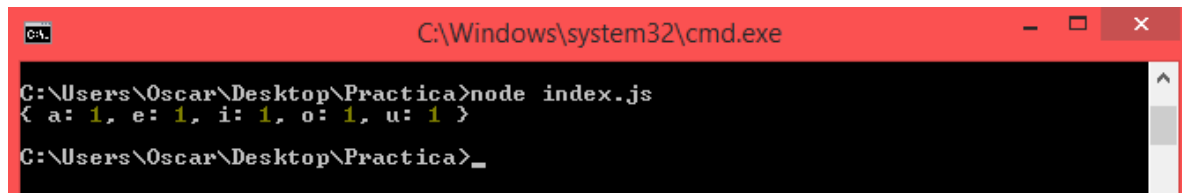
Código:

```
function miFuncion(texto) {
  texto = texto.toLowerCase();
  let contador = { a: 0, e: 0, i: 0, o: 0, u: 0 };
  for (let letra of texto) {
    if (contador.hasOwnProperty(letra)) {
      contador[letra]++;
    }
  }

  return contador;
}

let obj = miFuncion("euforia");
console.log(obj);
```

Salida:



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "C:\Users\Oscar\Desktop\Practica>". The user has entered "node index.js", and the output is "{ a: 1, e: 1, i: 1, o: 1, u: 1 }". The prompt is now "C:\Users\Oscar\Desktop\Practica>_".

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

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

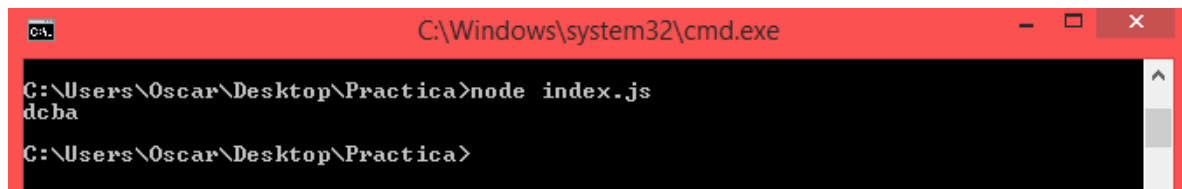
```
console.log(obj) // dcba
```

Código:

```
function invertirCadena(cadena) {
  let invertida = "";
  for (let i = cadena.length - 1; i >= 0; i--) {
    invertida = invertida+cadena[i];
  }
}
```

```
}  
  return invertida;  
}  
  
let cad = invertirCadena("abcd");  
console.log(cad);
```

Salida:



A screenshot of a Windows command prompt window. The title bar is red and shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the following text: 'C:\Users\Oscar\Desktop\Practica>node index.js', followed by the output 'dcba' on the next line, and then 'C:\Users\Oscar\Desktop\Practica>' on the third line.

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]}
```

Código:

```
function separar(nums) {  
  let resultado = {  
    pares: [],  
    impares: []  
  };  
  
  for (let num of nums) {  
    if (num % 2 == 0) {  
      resultado.pares.push(num);  
    } else {  
      resultado.impares.push(num);  
    }  
  }  
  return resultado;  
}  
  
let obj = separar([1,2,3,4,5]);  
console.log(obj);
```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
{ pares: [ 2, 4 ], impares: [ 1, 3, 5 ] }

C:\Users\Oscar\Desktop\Practica>
```

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 }
```

Código:

```
function obtenerMayorMenor(arreglo) {
  let mayor = arreglo[0];
  let menor = arreglo[0];

  for (let num of arreglo) {
    if (num > mayor) {
      mayor = num;
    }
    if (num < menor) {
      menor = num;
    }
  }

  return { mayor, menor };
}

let obj = obtenerMayorMenor([3,1,5,4,2]);
console.log(obj);
```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
{ mayor: 5, menor: 1 }

C:\Users\Oscar\Desktop\Practica>
```

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
```

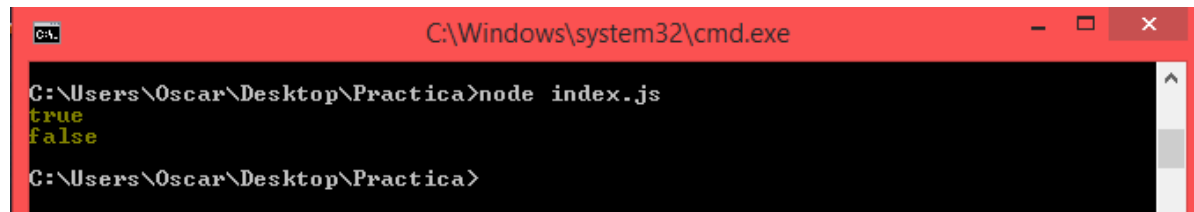
```
let band = miFuncion("hola")
```

```
console.log(band) // false
```

Código:

```
function palindromo(cadena) {  
  let invertir = "";  
  for (let i = cadena.length - 1; i >= 0; i--) {  
    invertir=invertir+cadena[i];  
  }  
  return cadena == invertir;  
}  
  
let band1 = palindromo("oruro");  
console.log(band1);  
  
let band2 = palindromo("hola");  
console.log(band2);
```

Salida:



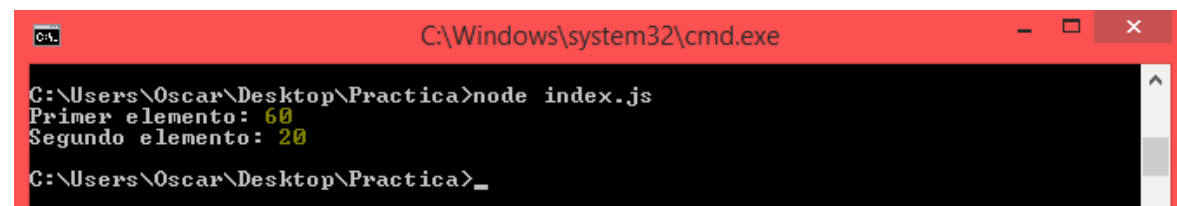
```
C:\Windows\system32\cmd.exe  
  
C:\Users\Oscar\Desktop\Practica>node index.js  
true  
false  
  
C:\Users\Oscar\Desktop\Practica>
```

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

Código:

```
function extraer(arreglo) {  
  let [primero, segundo] = arreglo;  
  console.log("Primer elemento:", primero);  
  console.log("Segundo elemento:", segundo);  
}  
  
extraer([60,20,30,40,50]);
```

Salida:



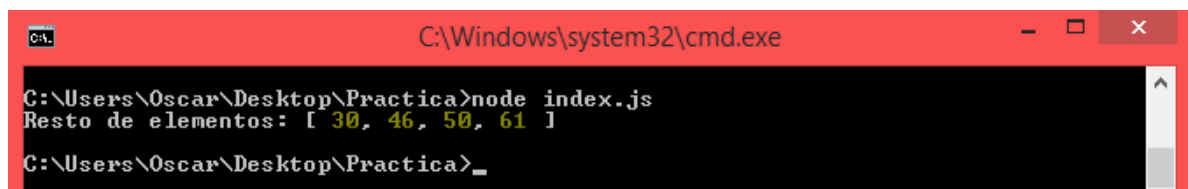
```
C:\Windows\system32\cmd.exe  
  
C:\Users\Oscar\Desktop\Practica>node index.js  
Primer elemento: 60  
Segundo elemento: 20  
  
C:\Users\Oscar\Desktop\Practica>_
```

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

Código:

```
function extraer(arreglo) {  
  let [primero, segundo, ...resto] = arreglo;  
  console.log("Resto de elementos:", resto);  
}  
  
extraer([5,10,30,46,50,61]);
```

Salida:



```
C:\Windows\system32\cmd.exe  
  
G:\Users\Oscar\Desktop\Practica>node index.js  
Resto de elementos: [ 30, 46, 50, 61 ]  
  
G:\Users\Oscar\Desktop\Practica>_
```

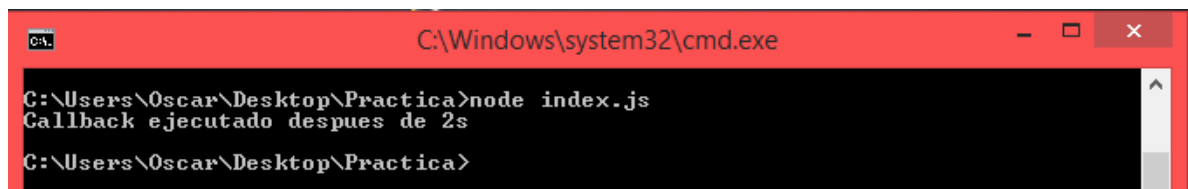
Callback y Promesas en JS

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

Código:

```
function ejecutar(callback) {  
  setTimeout(() => {  
    callback();  
  }, 2000);  
}  
  
ejecutar(() => {  
  console.log("Callback ejecutado despues de 2s");  
});
```

Salida:



```
C:\Windows\system32\cmd.exe  
  
G:\Users\Oscar\Desktop\Practica>node index.js  
Callback ejecutado despues de 2s  
  
G:\Users\Oscar\Desktop\Practica>
```

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

Código:

```
function msg() {  
  return new Promise((resolve) => {
```

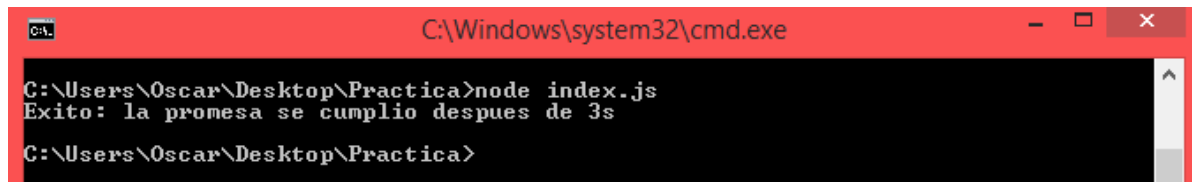
```

    setTimeout(() => {
      resolve("Exito: la promesa se cumplio despues de 3s");
    }, 3000);
  });
}

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

```

Salida:



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "C:\Users\Oscar\Desktop\Practica>". The user has entered "node index.js", and the output is "Exito: la promesa se cumplio despues de 3s". The prompt is now "C:\Users\Oscar\Desktop\Practica>".

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

Callbacks:

Son útiles cuando solo hay una acción simple y rápida después de otra. Ejemplo: ejecutar una función después de 1 segundo.

Problema: si hay muchas operaciones anidadas → aparece el *Callback Hell* (código difícil de leer).

Además, los callbacks dependen de pasar funciones como parámetros, lo que puede complicar la depuración y el manejo de errores.

Promesas:

Se usan cuando hay tareas asíncronas encadenadas (ej: pedir datos a un servidor, procesarlos, luego guardarlos).

Ventaja: `.then()` y `.catch()` hacen el código más limpio y fácil de mantener. Permiten manejar errores de manera centralizada y se integran fácilmente con `async/await`, lo que hace el código mucho más legible y parecido a la programación síncrona.

11. Proporcione un ejemplo concreto de encadenamiento de promesas.

Código:

```

function paso1() {
  return new Promise((resolve) => {
    setTimeout(() => resolve("Paso 1 completado"), 1000);
  });
}

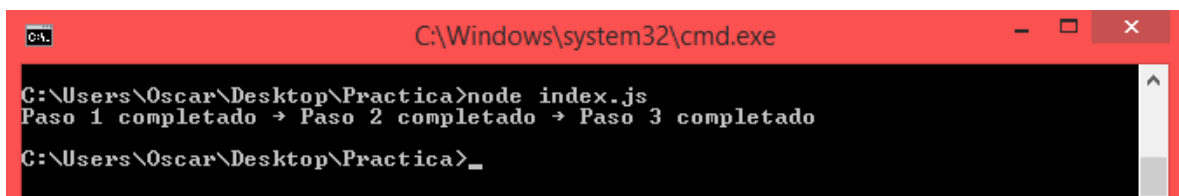
```

```
function paso2(mensaje) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(mensaje + " → Paso 2 completado"), 1000);
  });
}

function paso3(mensaje) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(mensaje + " → Paso 3 completado"), 1000);
  });
}

paso1()
  .then((r) => paso2(r))
  .then((r) => paso3(r))
  .then((r) => console.log(r));
```

Salida:



```
C:\Windows\system32\cmd.exe
G:\Users\Oscar\Desktop\Practica>node index.js
Paso 1 completado → Paso 2 completado → Paso 3 completado
G:\Users\Oscar\Desktop\Practica>_
```

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

Código:

```
function tarea1(callback) {
  setTimeout(() => {
    console.log("Tarea 1 lista");
    callback();
  }, 1000);
}

function tarea2(callback) {
  setTimeout(() => {
    console.log("Tarea 2 lista");
    callback();
  }, 1000);
}

function tarea3(callback) {
  setTimeout(() => {
    console.log("Tarea 3 lista");
```



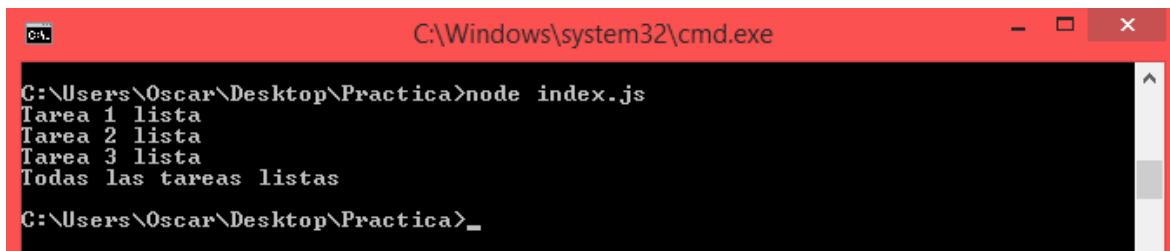
```

        callback();
    }, 1000);
}

tarea1(() => {
    tarea2(() => {
        tarea3(() => {
            console.log("Todas las tareas listas");
        });
    });
});
});
});

```

Salida:



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the execution of a Node.js script. The prompt is "C:\Users\Oscar\Desktop\Practica>node index.js". The output of the script is displayed as follows:

```

Tarea 1 lista
Tarea 2 lista
Tarea 3 lista
Todas las tareas listas
C:\Users\Oscar\Desktop\Practica>_

```

Código:

```

//Con async/await (más limpio):
function tarea(ms, nombre) {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log(`${nombre} lista`);
            resolve();
        }, ms);
    });
}

async function ejecutar() {
    await tarea(1000, "Tarea 1");
    await tarea(1000, "Tarea 2");
    await tarea(1000, "Tarea 3");
    console.log("Todas las tareas listas");
}

ejecutar();

```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
Tarea 1 lista
Tarea 2 lista
Tarea 3 lista
Todas las tareas listas

C:\Users\Oscar\Desktop\Practica>_
```

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

Código:

```
function obtenerUsuario() {
  return new Promise((resolve) => {
    setTimeout(() => resolve("Usuario: Oscar"), 1000);
  });
}

function obtenerPedidos(usuario) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(`${usuario} → Pedidos obtenidos`), 1000);
  });
}

function procesarPedidos(pedidos) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(`${pedidos} → Pedidos procesados`), 1000);
  });
}

// Anidamiento con promesas
obtenerUsuario()
  .then((usuario) => {
    return obtenerPedidos(usuario).then((pedidos) => {
      return procesarPedidos(pedidos).then((res) => {
        console.log(res);
      });
    });
  });
```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
Usuario: Oscar → Pedidos obtenidos → Pedidos procesados

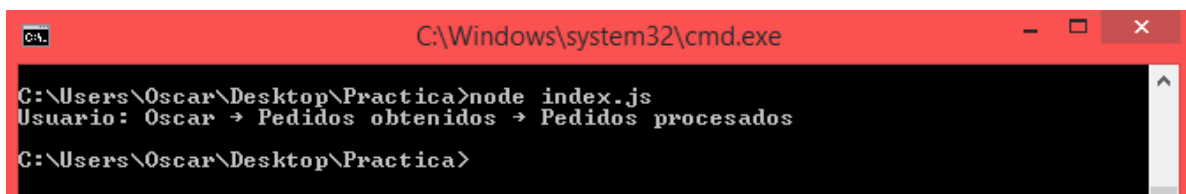
C:\Users\Oscar\Desktop\Practica>_
```

Código:

```
//Con async/await (más limpio):
async function main() {
  const usuario = await obtenerUsuario();
  const pedidos = await obtenerPedidos(usuario);
  const resultado = await procesarPedidos(pedidos);
  console.log(resultado);
}

main();
```

Salida:

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the execution of a Node.js script. The prompt is "C:\Users\Oscar\Desktop\Practica>". The user has entered "node index.js". The output is "Usuario: Oscar → Pedidos obtenidos → Pedidos procesados". The prompt is now "C:\Users\Oscar\Desktop\Practica>".

```
C:\Windows\system32\cmd.exe
C:\Users\Oscar\Desktop\Practica>node index.js
Usuario: Oscar → Pedidos obtenidos → Pedidos procesados
C:\Users\Oscar\Desktop\Practica>
```

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

Código:

```
function obtenerNumero() {
  return new Promise((resolve) => {
    setTimeout(() => resolve(35), 1000);
  });
}

function obtenerNumeroCallback(callback) {
  obtenerNumero()
    .then((resultado) => callback(null, resultado))
    .catch((error) => callback(error, null));
}

obtenerNumeroCallback((err, num) => {
  if (err) console.log("Error:", err);
  else console.log("Número obtenido:", num);
});
```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
Número obtenido: 35

C:\Users\Oscar\Desktop\Practica>
```

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

Código:

```
function sumarCallback(a, b, callback) {
  setTimeout(() => {
    let suma = a + b;
    callback(null, suma);
  }, 1000);
}

function sumarPromesa(a, b) {
  return new Promise((resolve, reject) => {
    sumarCallback(a, b, (err, resultado) => {
      if (err) reject(err);
      else resolve(resultado);
    });
  });
}

sumarPromesa(5,7).then((res) => console.log("Suma:", res));
```

Salida:

```
C:\Windows\system32\cmd.exe

C:\Users\Oscar\Desktop\Practica>node index.js
Suma: 12

C:\Users\Oscar\Desktop\Practica>_
```

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

Código:

```
function obtenerUsuario() {
  return new Promise((resolve) => {
    setTimeout(() => resolve("Usuario: Oscar"), 1000);
  });
}

obtenerUsuario().then((usuario) => {
```

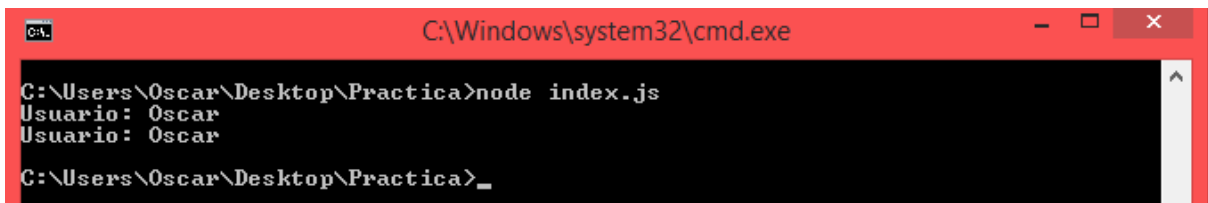
```
    console.log(usuario);
});

//Versión con async/await:
function obtenerUsuario() {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Usuario: Oscar"), 1000);
    });
}

async function main() {
    let usuario = await obtenerUsuario();
    console.log(usuario);
}

main();
```

Salida:



The screenshot shows a Windows Command Prompt window with a red title bar. The title bar text is "C:\Windows\system32\cmd.exe". The window contains the following text:

```
C:\Users\Oscar\Desktop\Practica>node index.js
Usuario: Oscar
Usuario: Oscar
C:\Users\Oscar\Desktop\Practica>_
```