

1.-

Ejercicio1

```
function contarVocales(texto) {
  const vocales = { a: 0, e: 0, i: 0, o: 0, u: 0 };
  texto = texto.toLowerCase();

  for (let letra of texto) {
    if (vocales.hasOwnProperty(letra)) {
      vocales[letra]++;
    }
  }

  return vocales;
}

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

```
{ a: 1, e: 1, i: 1, o: 1, u: 1 }
```

2.-

```
function invertirCadena(cadena) {
  return cadena.split('').reverse().join('');
}
```

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

```
C:\Users\Pedro\Desktop\Tareasnodejs\Practica1>node ejercicio2.js
dcba
```

3.-

```
function separarParesImpares(arr) {
  const resultado = { pares: [], impares: [] };
  for (let num of arr) {
    if (num % 2 === 0) {
      resultado.pares.push(num);
    } else {
      resultado.impares.push(num);
    }
  }

  return resultado;
}
```

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

```
C:\Users\Pedro\Desktop\Tareasnodejs\Practica1>node ejercicio3.js
pares: [ 2, 4 ], impares: [ 1, 3, 5 ]
```

4.-

```
function mayorYMenor(arr) {
  let mayor = arr[0];
  let menor = arr[0];

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

  return { mayor, menor };
}
```

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

```
C:\Users\Pedro\Desktop\Tareasnodejs
{ mayor: 5, menor: 1 }
```

5.-

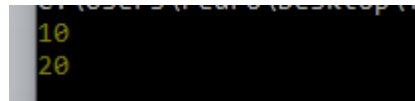
```
function esPalindromo(cadena) {
  cadena = cadena.toLowerCase();
  let cadenaInvertida = cadena.split('').reverse().join('');
  return cadena === cadenaInvertida;
}

console.log(esPalindromo("oruro"));
console.log(esPalindromo("hola"));
```

```
C:\Users\Pedro\Desktop\Ta
true
false
```

6.-

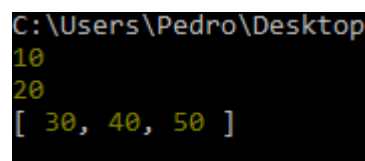
```
let numeros = [10, 20, 30, 40, 50];  
  
let [primero, segundo] = numeros;  
  
console.log(primero);  
console.log(segundo);
```



```
C:\Users\Pedro\Desktop\trabaja con promesas  
10  
20
```

7.-

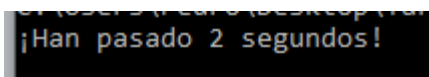
```
let numeros = [10, 20, 30, 40, 50];  
let [primero, segundo, ...resto] = numeros;  
  
console.log(primero);  
console.log(segundo);  
console.log(resto);
```



```
C:\Users\Pedro\Desktop  
10  
20  
[ 30, 40, 50 ]
```

8.-

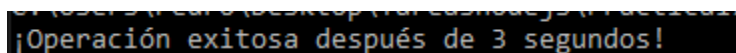
```
function ejecutarDespues2Segundos(callback) {  
  setTimeout(callback, 2000);  
}  
  
ejecutarDespues2Segundos(() => {  
  console.log("¡Han pasado 2 segundos!");  
});
```



```
C:\Users\Pedro\Desktop\trabaja con promesas  
¡Han pasado 2 segundos!
```

9.-

```
function mensajeExito() {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve("¡Operación exitosa después de 3 segundos!");  
    }, 3000);  
  });  
}  
mensajeExito()  
  .then((mensaje) => {  
    console.log(mensaje);  
  })  
  .catch((error) => {  
    console.log("Ocurrió un error:", error);  
  });
```



```
C:\Users\Pedro\Desktop\trabaja con promesas  
¡Operación exitosa después de 3 segundos!
```

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

Un **callback** es una función que se pasa como argumento a otra función para que se ejecute después de que termine una tarea.

Cuándo usarlo: Para tareas **simples y rápidas** que no dependen de muchas operaciones asíncronas.

```
// Ejemplo de Callback
function saludar(nombre, callback) {
  console.log("Hola " + nombre);
  callback();
}

saludar("Juan", () => {
  console.log("Esto se ejecuta después de saludar");
});
```

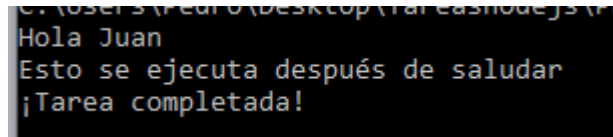
Una **promesa** es un objeto que representa el **resultado futuro** de una operación asíncrona. Permite manejar **operaciones que tardan en completarse**

Cuándo usarlo: Para tareas **asíncronas más complejas**.

Cuando necesitas encadenar varias operaciones sin anidar funciones.

```
// Ejemplo de Promise
function tareaAsincrona() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("¡Tarea completada!");
    }, 2000);
  });
}

tareaAsincrona()
  .then(resultado => console.log(resultado))
  .catch(error => console.log(error));
```



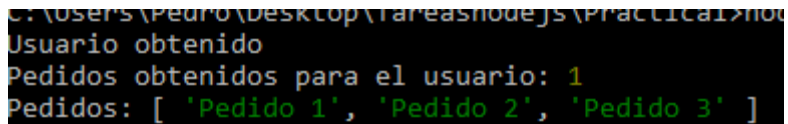
```
c:\Users\Pedro\Desktop\tareas\node.js (Practical 1)
Hola Juan
Esto se ejecuta después de saludar
¡Tarea completada!
```

11.-

```
function obtenerUsuario() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Usuario obtenido");
      resolve({ id: 1, nombre: "Juan" });
    }, 1000);
  });
}

function obtenerPedidos(usuarioId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Pedidos obtenidos para el usuario:", usuarioId);
      resolve(["Pedido 1", "Pedido 2", "Pedido 3"]);
    }, 1000);
  });
}

obtenerUsuario()
  .then(usuario => {
    return obtenerPedidos(usuario.id);
  })
  .then(pedidos => {
    console.log("Pedidos:", pedidos);
  })
  .catch(error => {
    console.log("Ocurrió un error:", error);
  });
```



```
c:\Users\Pedro\Desktop\tareas\node.js (Practical 1)
Usuario obtenido
Pedidos obtenidos para el usuario: 1
Pedidos: [ 'Pedido 1', 'Pedido 2', 'Pedido 3' ]
```

12.-

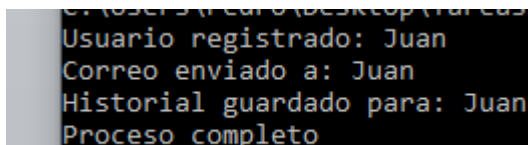
```
function registrarUsuario(nombre) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Usuario registrado:", nombre);
      resolve(nombre);
    }, 1000);
  });
}

function enviarCorreo(usuario) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Correo enviado a:", usuario);
      resolve(usuario);
    }, 1000);
  });
}

function guardarHistorial(usuario) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Historial guardado para:", usuario);
      resolve();
    }, 1000);
  });
}

async function procesoCompleto() {
  try {
    const usuario = await registrarUsuario("Juan");
    await enviarCorreo(usuario);
    await guardarHistorial(usuario);
    console.log("Proceso completo");
  } catch (error) {
    console.log("Ocurrió un error:", error);
  }
}

procesoCompleto();
```



```
c:\Users\Pedro\Desktop\tareas\node.js (Practical 1)
Usuario registrado: Juan
Correo enviado a: Juan
Historial guardado para: Juan
Proceso completo
```

13.-

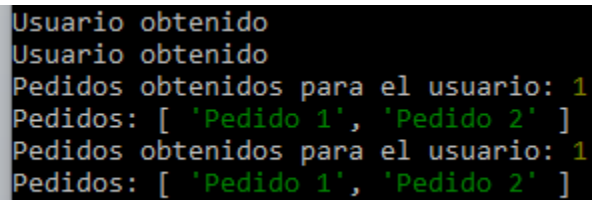
```
function obtenerUsuario() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Usuario obtenido");
      resolve({ id: 1, nombre: "Juan" });
    }, 1000);
  });
}

function obtenerPedidos(usuarioId) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Pedidos obtenidos para el usuario:", usuarioId);
      resolve(["Pedido 1", "Pedido 2"]);
    }, 1000);
  });
}

obtenerUsuario().then((usuario) => {
  obtenerPedidos(usuario.id).then((pedidos) => {
    console.log("Pedidos:", pedidos);
  }).catch((error) => {
    console.log("Error en pedidos:", error);
  });
}).catch((error) => {
  console.log("Error en usuario:", error);
});

async function mostrarPedidos() {
  try {
    const usuario = await obtenerUsuario();
    const pedidos = await obtenerPedidos(usuario.id);
    console.log("Pedidos:", pedidos);
  } catch (error) {
    console.log("Ocurrió un error:", error);
  }
}

mostrarPedidos();
```



```
Usuario obtenido
Usuario obtenido
Pedidos obtenidos para el usuario: 1
Pedidos: [ 'Pedido 1', 'Pedido 2' ]
Pedidos obtenidos para el usuario: 1
Pedidos: [ 'Pedido 1', 'Pedido 2' ]
```

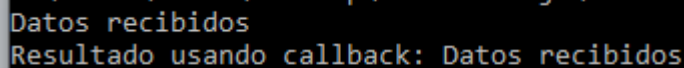
14.-

```
function obtenerDatos() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Datos recibidos");
    }, 1000);
  });
}

// Uso de promesa callback
obtenerDatos().then((resultado) => {
  console.log(resultado);
});

function obtenerDatosConCallback(callback) {
  obtenerDatos()
    .then((resultado) => callback(null, resultado)) // Primer argumento null = sin error
    .catch((error) => callback(error));
}

obtenerDatosConCallback((error, resultado) => {
  if (error) {
    console.log("Ocurrió un error:", error);
  } else {
    console.log("Resultado usando callback:", resultado);
  }
});
```



```
Datos recibidos
Resultado usando callback: Datos recibidos
```

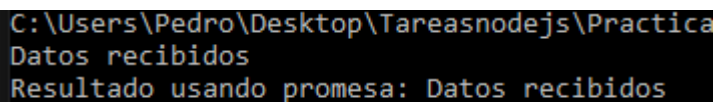
15.-

```
function obtenerDatosCallback(callback) {
  setTimeout(() => {
    const datos = "Datos recibidos";
    callback(null, datos);
  }, 1000);
}

// Convertir el callback en promesa
obtenerDatosCallback((error, resultado) => {
  if (error) {
    console.log("Ocurrió un error:", error);
  } else {
    console.log(resultado);
  }
});

function obtenerDatosPromesa() {
  return new Promise((resolve, reject) => {
    obtenerDatosCallback((error, resultado) => {
      if (error) {
        reject(error);
      } else {
        resolve(resultado);
      }
    });
  });
}

obtenerDatosPromesa()
  .then((resultado) => console.log("Resultado usando promesa:", resultado))
  .catch((error) => console.log("Ocurrió un error:", error));
```



```
C:\Users\Pedro\Desktop\Tareasnodejs\Practica
Datos recibidos
Resultado usando promesa: Datos recibidos
```

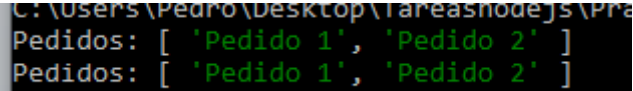
16.-

```
function obtenerUsuario() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({ id: 1, nombre: "Juan" });
    }, 1000);
  });
}

function obtenerPedidos(usuarioId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(["Pedido 1", "Pedido 2"]);
    }, 1000);
  });
}

obtenerUsuario()
  .then((usuario) => obtenerPedidos(usuario.id))
  .then((pedidos) => console.log("Pedidos:", pedidos))
  .catch((error) => console.log("Ocurrió un error:", error));

//funcion migrada a async/await
async function mostrarPedidos() {
  try {
    const usuario = await obtenerUsuario();
    const pedidos = await obtenerPedidos(usuario.id);
    console.log("Pedidos:", pedidos);
  } catch (error) {
    console.log("Ocurrió un error:", error);
  }
}
mostrarPedidos();
```



```
C:\Users\Pedro\Desktop\tareas\nodejs\Practica16>
Pedidos: [ 'Pedido 1', 'Pedido 2' ]
Pedidos: [ 'Pedido 1', 'Pedido 2' ]
```