Practica 1

Nombre Completo: Alexander Condori Martinez

CI: 8343695

Carrera: Informatica

Materia: Programación en JavaScript

Fecha: 23 de septiembre de 2025

Soluciones:

1. Contar cuántas veces aparece cada vocal en un texto

```
function miFuncion(t) {
  let r = { a: 0, e: 0, i: 0, o: 0, u: 0 };
  for (let x of t.toLowerCase()) {
    if (r.hasOwnProperty(x)) {
      r[x]++;
    }
  }
  return r;
}

let obj = miFuncion("euforia");
console.log(obj); // { a: 1, e: 1, i: 1, o: 1, u: 1 }
```

---

2. Invertir el orden de los caracteres en una frase

```
function miFuncion(t) {
  return t.split("").reverse().join("");
}

let cad = miFuncion("abcd");
console.log(cad); // dcba
```

---

3. Separar números pares e impares en un objeto

```
function miFuncion(a) {
  let r = { pares: [], impares: [] };
  for (let x of a) {
    if (x % 2 === 0) {
      r.pares.push(x);
    } else {
      r.impares.push(x);
    }
  }
  return r;
}
```

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

---

4. Obtener el número mayor y menor de un arreglo

```
function miFuncion(a) {
  return {
    mayor: Math.max(...a),
    menor: Math.min(...a)
  };
}

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

---

5. Verificar si una cadena es palíndromo

```
function miFuncion(t) {
  let r = t.split("").reverse().join("");
  return t === r;
}

let band = miFuncion("oruro");
console.log(band); // true

band = miFuncion("hola");
console.log(band); // false
```

---

6. Desestructurar los dos primeros elementos de un arreglo

```
let a = [10, 20, 30];
let [x, y] = a;

console.log(x); // 10
console.log(y); // 20
```

---

7. Almacenar el resto de los elementos de un arreglo

```
let a = [10, 20, 30, 40];
let [x, y, ...z] = a;

console.log(z); // [30, 40]
```

---

8. Ejecutar una función callback después de 2 segundos

```
function miFuncion(cb) {
  setTimeout(cb, 2000);
}

miFuncion(function() {
  console.log("Ejecutado después de 2 segundos");
});
```

---

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

```
function miFuncion() {
  return new Promise(function(r) {
    setTimeout(() => r("Éxito después de 3 segundos"), 3000);
  });
}

miFuncion().then(function(m) {
  console.log(m);
});
```

---

10. ¿Cuándo usar callback y cuándo promesa?

- **Callback**: Para tareas simples, rápidas o eventos (como `setTimeout`, `onclick`).
- **Promesa**: Para tareas asíncronas complejas, encadenamiento, manejo de errores (`fetch`, `async/await`).

---

11. Encadenamiento de promesas

```
function paso1() {
  return new Promise(r => setTimeout(() => {
    console.log("Paso 1");
    r("Resultado 1");
  }, 1000));
}

function paso2(valor) {
  return new Promise(r => setTimeout(() => {
    console.log("Paso 2 con:", valor);
    r("Resultado 2");
  }, 1000));
}

function paso3(valor) {
  return new Promise(r => setTimeout(() => {
    console.log("Paso 3 con:", valor);
    r("Finalizado");
  }, 1000));
}

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

---

12. Reescribir anidamiento de callbacks con async/await

```
function paso1() {
  return new Promise(r => setTimeout(() => {
    console.log("Paso 1");
    r();
  }, 1000));
}

function paso2() {
```

```
  return new Promise(r => setTimeout(() => {
    console.log("Paso 2");
    r();
  }, 1000));
}

async function ejecutar() {
  await paso1();
  await paso2();
  console.log("Completado");
}

ejecutar();
```

---

13. Reescribir anidamiento de promesas con async/await

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

function paso2() {
  return new Promise(r => setTimeout(() => r("Paso 2 completado"), 1000));
}

function paso3() {
  return new Promise(r => setTimeout(() => r("Paso 3 completado"), 1000));
}

async function ejecutar() {
  let res1 = await paso1();
  console.log(res1);

  let res2 = await paso2();
  console.log(res2);

  let res3 = await paso3();
  console.log(res3);

  console.log("Proceso finalizado");
}

ejecutar();
```

---

14. Convertir una promesa en callback

```
function obtenerDato() {
  return new Promise(r => setTimeout(() => r("Dato recibido"), 1000));
}

function obtenerDatoConCallback(cb) {
  obtenerDato()
    .then(res => cb(null, res))
    .catch(err => cb(err));
}

obtenerDatoConCallback(function(err, res) {
  if (err) console.error(err);
  else console.log(res);
});
```

---

15. Convertir un callback en promesa

```
function obtenerDato(cb) {
  setTimeout(() => cb(null, "Dato recibido"), 1000);
}

function obtenerDatoPromesa() {
  return new Promise(function(r, e) {
    obtenerDato(function(err, res) {
      if (err) e(err);
      else r(res);
    });
  });
}

obtenerDatoPromesa().then(res => console.log(res));
```

---

16. Migrar una función con promesas a async/await

```
function obtenerDato() {
  return new Promise(r => setTimeout(() => r("Dato recibido"), 1000));
}

async function mostrarDato() {
  let res = await obtenerDato();
  console.log(res);
}

mostrarDato();
```