UNIVERSIDAD MAYOR DE SAN ANDRÉS

**FACULTAD DE CIENCIAS PURAS Y NATURALES CARRERA DE INFORMÁTICA**



**PRACTICA Nº1**

**NOMBRE:** JHOSELIN DANIELA QUISPE ALVAREZ

**CI:** 6967761 **CARRERA:** INFORMATICA

**MATERIA:** PROGRAMACION WEB III

LA PAZ – BOLIVIA

24 de Septiembre de 2025

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

function contVocal(cadena){

const x = { a: 0, e: 0, i: 0, o: 0, u: 0 };

cadena = cadena.toLowerCase();

for (let i = 0; i < cadena.length; i++) {

let letra = cadena[i];

if (letra === "a") {

x.a++;

} else if (letra === "e") {

x.e++;

} else if (letra === "i") {

x.i++;

} else if (letra === "o") {

x.o++;

} else if (letra === "u") {

x.u++;

}

}

return x;

}

const cadena = "euforia";

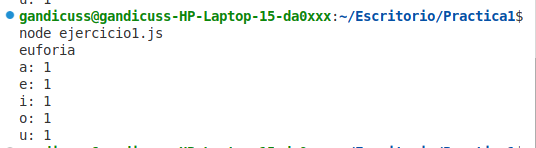
console.log(cadena);

const x = contVocal(cadena);

for (let vocal in x) {

console.log(`${vocal}: ${x[vocal]}`);

}



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

function miFuncion(cadena) {

let invertida = "";

for (let i = cadena.length - 1; i >= 0; i--) {

invertida = invertida + cadena[i];

}

return invertida;

}

let cad = miFuncion("abcd");

console.log(cad);



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

function miFuncion(numeros) {

let r = { pares: [], impares: [] };

let p = 0;

let imp = 0;

for (let i = 0; i < numeros.length; i++) {

let num = numeros[i];

if (num % 2 === 0) {

r.pares[p] = num;

p++;

} else {

r.impares[imp] = num;

imp++;

}

}

return r;

}

let obj = miFuncion([1, 2, 3, 4, 5]);

console.log(obj);



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

function miFuncion(numeros) {

let menor = numeros[0];

let mayor = numeros[0];

for (let i = 1; i < numeros.length; i++) {

let num = numeros[i];

if (num < menor) {

menor = num;

}

if (num > mayor) {

mayor = num;

}

}

return { mayor: mayor, menor: menor };

}

let obj = miFuncion([3, 1, 5, 4, 2]);

console.log(obj);



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

function miFuncion(cadena) {

let invertida = "";

for (let i = cadena.length - 1; i >= 0; i--) {

invertida = invertida + cadena[i];

}

if (cadena === invertida) {

return true;

} else {

return false;

}

}

let band = miFuncion("oruro");

console.log(band);

band = miFuncion("hola");

console.log(band);



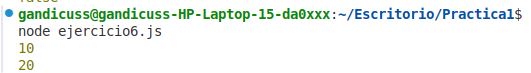
**6. Tomar los dos primeros elementos de un arreglo y almacenarlos en dos variables mediante desestructuración.**

let numeros = [10, 20, 30, 40];

let [a, b] = numeros;

console.log(a);

console.log(b);



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

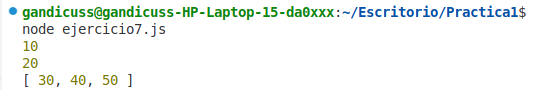
let numeros = [10, 20, 30, 40, 50];

let [a, b, ...resto] = numeros;

console.log(a);

console.log(b);

console.log(resto);



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

function ejecutarDespuesDe2Segundos(callback) {

setTimeout(function() {

callback();

}, 2000);

}

function saludo() {

console.log("Hola, Se mostro luego de dos segundos");

}

ejecutarDespuesDe2Segundos(saludo);



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

const miPromesa = new Promise((resolve) => {

setTimeout(() => {

resolve("Jhoselin");

}, 3000);

});

miPromesa.then((x) => {

console.log(x);

});



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

CALLBACKS

Callback en JavaScript es una función que se pasa como argumento a otra función y es llamada por ella en un momento determinado de la operación, por ejemplo, después de obtener datos de la base de datos. De esa manera el código de llamada puede extender la función llamada, proporcionando un comportamiento adicional, y la función llamada se puede escribir de una manera más general, sin conocer el contexto en el que se llama.

PROMESAS

Una Promesa en JavaScript es un objeto que contiene información sobre el estado actual y resultado de una operación asincrónica. Hay tres estados diferentes: pending, fulfilledy rejected. El pendingEstado es el estado inicial para cada Promesa y significa que la operación aún no se ha completado. Cuando lo es fulfilled, significa que la operación se completa con éxito, y su valor está disponible. Si la operación falla, la Promise estará en el rejectedestado y como resultado recibiremos un error.

**11. Proporcione un ejemplo concreto de encadenamiento de promesas.**

function irALaUniversidad() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("Estoy yendo a clases");

}, 1000);

});

}

function pasarClases() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("Estoy pasando clases");

}, 2000);

});

}

function volverACasa() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("Regrese a casa");

}, 3000);

});

}

irALaUniversidad()

.then((mensaje1) => {

console.log(mensaje1);

return pasarClases();

})

.then((mensaje2) => {

console.log(mensaje2);

return volverACasa();

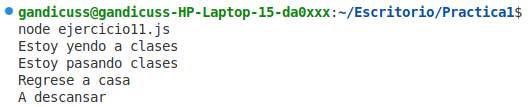
})

.then((mensaje3) => {

console.log(mensaje3);

console.log("A descansar");

});



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

function hacerTareaCallback(callback) {

setTimeout(() => {

console.log("Hice mi tarea");

callback();

}, 1000);

}

function cenarCallback(callback) {

setTimeout(() => {

console.log("Cené");

callback();

}, 1000);

}

function dormirCallback(callback) {

setTimeout(() => {

console.log("Me fui a dormir");

callback();

}, 1000);

}

console.log("---------- Versión con Callbacks ----------");

hacerTareaCallback(() => {

cenarCallback(() => {

dormirCallback(() => {

console.log("El dia acabo");

});

});

});

function hacerTarea() {

return new Promise((resolve) => {

setTimeout(() => resolve("Hago mi tarea"), 1000);

});

}

function cenar() {

return new Promise((resolve) => {

setTimeout(() => resolve("Estoy cenando"), 1000);

});

}

function dormir() {

return new Promise((resolve) => {

setTimeout(() => resolve("Es hora de dormir"), 1000);

});

}

async function rutina() {

console.log(await hacerTarea());

console.log(await cenar());

console.log(await dormir());

console.log("Acabo el dia");

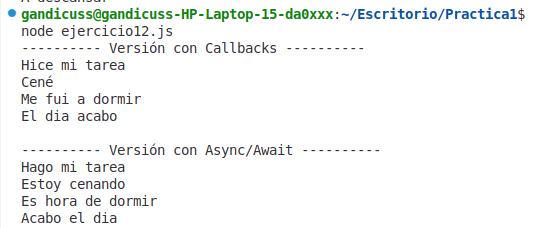
}

setTimeout(() => {

console.log("\n---------- Versión con Async/Await ----------");

rutina();

}, 4000);



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

function hacerTareaPromesa() {

return new Promise((resolve) => {

setTimeout(() => resolve("Hice mi tarea"), 1000);

});

}

function cenarPromesa() {

return new Promise((resolve) => {

setTimeout(() => resolve("Cenare mi comida"), 1000);

});

}

function dormirPromesa() {

return new Promise((resolve) => {

setTimeout(() => resolve("Es hora de dormir"), 1000);

});

}

console.log("---------- Versión con Promesas Anidadas ----------");

hacerTareaPromesa().then((mensaje1) => {

console.log(mensaje1);

cenarPromesa().then((mensaje2) => {

console.log(mensaje2);

dormirPromesa().then((mensaje3) => {

console.log(mensaje3);

console.log("El dia termino");

});

});

});

async function rutina() {

const tarea = await hacerTareaPromesa();

console.log(tarea);

const cena = await cenarPromesa();

console.log(cena);

const dormir = await dormirPromesa();

console.log(dormir);

console.log("El dia termino");

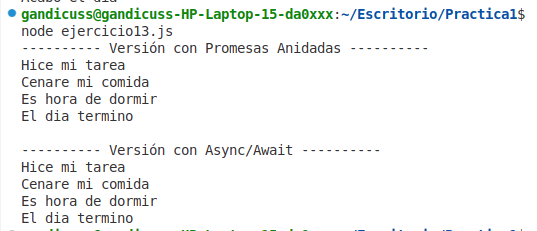
}

setTimeout(() => {

console.log("\n---------- Versión con Async/Await ----------");

rutina();

}, 4000);



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

function mensajeCallback(callback) {

setTimeout(() => {

callback("Mi nombre es Jhoselin");

}, 1000);

}

function mensajePromesa() {

return new Promise((resolve) => {

mensajeCallback((mensaje) => {

resolve(mensaje);

});

});

}

mensajePromesa().then((mensaje) => {

console.log(mensaje);

});



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

function nombreCallback(callback) {

setTimeout(() => {

callback("Mi nombre es Jhoselin");

}, 1000);

}

function nombrePromesa() {

return new Promise((resolve) => {

nombreCallback((mensaje) => {

resolve(mensaje);

});

});

}

nombrePromesa().then((mensaje) => {

console.log("Resultado con promesa:", mensaje);

});



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

function obtenerNombre() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("Soy Jhoselin");

}, 1000);

});

}

async function obtenerNombreAsync() {

const mensaje = await obtenerNombre();

console.log(mensaje);

}

obtenerNombreAsync();

