HACER UN FETCH

```
fetch("https://jsonplaceholder.typicode.com/users")
    .then((respuesta) => {
      // Si detecta algun problema lanza un nuevo error con el status
del HTTP retornado
      if (!respuesta.ok) {
        throw new Error("Error HTTP: ", respuesta.status);
      }
      return respuesta.json();
    })
    .then((datos) => {
      datos.forEach((usuario) => {
        console.log("Name: ", usuario.name);
        console.log("Email: ", usuario.email);
        console.log("Phone number: ", usuario.phone);
        console.log("----");
      });
    })
    .catch((error) => {
      console.log("Ocurrió un error: ", error.message);
    });

//Estructura JSON
/*
"id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
```

```
      "bs": "harness real-time e-markets"
    }
*/
```

```
// a) Completa el código para convertir el objeto datosEstudiante a una
cadena JSON
// estudianteJSON y enviarla a la consola:
const datosEstudiante = {
  nombre: "María",
  notas: [8, 9, 7.5],
  asignaturas: {
    matematicas: "A",
    programacion: "A+",
    historia: "B"
  }
};

// Tu código aquí
const estudianteJSON = JSON.stringify(datosEstudiante);
console.log(estudianteJSON);

/*b) Escribe código para convertir la siguiente cadena JSON en un
objeto JavaScript
productoObjeto que cumpla con los ejemplos de uso:*/
const datosJSON =
'{"producto":"Monitor","precio":249.99,"enStock":true}';

// Tu código aquí
const productoObjeto = JSON.parse(datosJSON);

// Ejemplos de uso (NO MODIFICAR):
console.log(productoObjeto.producto); // "Monitor"
console.log(productoObjeto.precio); // 249.99
console.log(productoObjeto.enStock); // true

//c) Escribe el objeto datosEstudiante del apartado a) en su
representación YAML.
// Tu YAML aquí
/*
nombre: María
notas:
  - 8
  - 9
  - 7.5
asignaturas:
  matematicas: "A"
  programacion: "A+"
```

```
  historia: "B"
*/
```

EXAMEN ANTERIOR

```
// Ejercicio 1: Manipulación de JSON y XML (20%)

// a) Completa el código para convertir el objeto `datosCurso` a una
cadena JSON formateada con 2 espacios de indentación y enviarla a la
consola:

const datosCurso = {
  nombre: "JavaScript Avanzado",
  estudiantes: [
    { id: 1, nombre: "Ana", calificacion: 9.5 },
    { id: 2, nombre: "Carlos", calificacion: 8.7 },
    { id: 3, nombre: "Elena", calificacion: 9.2 }
  ],
  profesor: {
    nombre: "Miguel",
    experiencia: 8,
    especialidades: ["Frontend", "Node.js", "Arquitectura"]
  },
  fechaInicio: "2025-09-15"
};

// Tu código aquí
const datosCursoJSON = JSON.stringify(datosCurso, null, 2);
console.log(datosCursoJSON);



// b) Escribe código para convertir la siguiente cadena JSON en un
objeto JavaScript `productosPedido` y calcula `precioTotal` que la suma
de todos los productos incluidos en el pedido:

const pedidoJSON = '{"cliente":"Juan
García","productos":[{"nombre":"Teclado","precio":45.99,"cantidad":1},{
"nombre":"Monitor","precio":199.50,"cantidad":2},{"nombre":"Ratón","pre
cio":25.75,"cantidad":1}],"urgente":true}';

// Tu código aquí
const productosPedido = JSON.parse(pedidoJSON);
let precioTotal = 0;

productosPedido.productos.forEach(producto => {
  precioTotal += producto.precio * producto.cantidad;
});
```

```
// Ejemplo de uso (NO MODIFICAR):
console.log("Precio total:", precioTotal); // Debe mostrar el precio
total del pedido
console.log(productosPedido.cliente); // "Juan García"
console.log(productosPedido.urgente); // true


// c) Escribe el objeto `datosCurso` del apartado a) en su
representación XML.

<curso>
    <nombre>JavaScript Avanzado</nombre>
    <estudiantes>
        <estudiante>
            <id>1</id>
            <nombre>Ana</nombre>
            <calificacion>9.5</calificacion>
        </estudiante>
        <estudiante>
            <id>2</id>
            <nombre>Carlos</nombre>
            <calificacion>8.7</calificacion>
        </estudiante>
        <estudiante>
            <id>3</id>
            <nombre>Elena</nombre>
            <calificacion>9.2</calificacion>
        </estudiante>
    </estudiantes>
    <profesor>
        <nombre>Miguel</nombre>
        <experiencia>8</experiencia>
        <especialidades>
            <especialidad>Frontend</especialidad>
            <especialidad>Node.js</especialidad>
            <especialidad>Arquitectura</especialidad>
        </especialidades>
    </profesor>
    <fechaInicio>2025-09-15</fechaInicio>
</curso>
```

```
// Ejercicio 2: Temporizadores y Promesas (20%)

// a) Escribe código que implemente una secuencia temporizada de
mensajes para un proceso de carga:
// Tu código aquí

console.log("Iniciando carga de datos...");

setTimeout(() => {
  console.log("Progreso: 25%");
}, 1000);

setTimeout(() => {
  console.log("Progreso: 50%");
}, 2000);

setTimeout(() => {
  console.log("Progreso: 75%");
}, 3000);

setTimeout(() => {
  console.log("Carga completada!");
}, 4000);

// b) Crea una función `esperarTiempo(segundos)` que devuelva una
promesa que se resuelva después del número especificado de segundos.

function esperarTiempo(segundos) {
  // Tu código aquí
  return new Promise((resolve) => {
    setTimeout(() => resolve(), segundos * 1000);
  });
}

// Ejemplo de uso (NO MODIFICAR):
console.log("Iniciando secuencia...");
esperarTiempo(2)
  .then(() => {
    console.log("Paso 1 completado");
    return esperarTiempo(1);
  })
  .then(() => {
    console.log("Paso 2 completado");
```

```
  return esperarTiempo(1.5);
})
.then(() => {
  console.log("Secuencia finalizada");
});
```

```
// Ejercicio 3: Almacenamiento en Cliente (20%)

// a) Crea código en JavaScript para guardar las preferencias de un
usuario (idioma, tema y notificaciones) en `localStorage` de manera que
persista entre sesiones.

const preferenciasUsuario = {
  idioma: "es",
  tema: "oscuro",
  notificaciones: true,
};

// Tu código aquí
// (Este código debe ejecutarse en la consola del navegador)
localStorage.setItem(
  "preferenciasUsuario",
  JSON.stringify(preferenciasUsuario)
);
console.log("Preferencias guardadas en localStorage.");

// b) Escribe una función `obtenerPreferencias()` que recupere la
configuración guardada en el paso anterior. Si no existe configuración
previa, debe devolver los valores por defecto: idioma "en", tema
"claro", notificaciones false.

function obtenerPreferencias() {
  // Tu código aquí
  // (Este código debe ejecutarse en la consola del navegador)
  const preferenciasGuardadas =
localStorage.getItem("preferenciasUsuario");
  if (preferenciasGuardadas) {
    return JSON.parse(preferenciasGuardadas);
  } else {
    return {
      idioma: "en",
      tema: "claro",
      notificaciones: false,
    };
  }
}

// Ejemplo de uso (NO MODIFICAR):
console.log(obtenerPreferencias());
```

```
// c) Escribe una función `gestionarHistorial(nuevaPagina)` que:

/* Mantenga un historial de las últimas 5 páginas visitadas por el
usuario.
 * Al añadir una nueva página, debe colocarla al principio del
historial.
 * Si el historial ya tiene 5 páginas, debe eliminar la más antigua.
 * Si la página ya está en el historial, debe moverla al principio.
 * La función debe devolver el array de páginas actual.*/

function gestionarHistorial(nuevaPagina) {
  // Tu código aquí
  // (Este código debe ejecutarse en la consola del navegador)
  let historial = JSON.parse(localStorage.getItem("historialVisitas"))
|| [];

  // Si la página ya está en el historial, eliminarla para moverla al
principio
  const indiceExistente = historial.indexOf(nuevaPagina);
  if (indiceExistente > -1) {
    historial.splice(indiceExistente, 1); // Eliminar la página
existente
  }

  // Añadir la nueva página al principio del array
  historial.unshift(nuevaPagina);

  // Mantener solo las últimas 5 páginas
  if (historial.length > 5) {
    historial.pop();
  }

  // Guardar el historial actualizado
  localStorage.setItem("historialVisitas", JSON.stringify(historial));

  return historial;
}

// Ejemplo de uso (NO MODIFICAR):
// (Este código se ejecuta mejor en la consola del navegador)
// Para probar, es mejor limpiar el localStorage antes o comentar las
líneas de abajo
```

```
// localStorage.removeItem('historialVisitas'); // Descomentar para
resetear en cada ejecución de prueba
console.log(gestionarHistorial("/inicio")); // ["/inicio"]
console.log(gestionarHistorial("/productos")); // ["/productos",
"/inicio"]
console.log(gestionarHistorial("/contacto")); // ["/contacto",
"/productos", "/inicio"]
console.log(gestionarHistorial("/productos")); // ["/productos",
"/contacto", "/inicio"]
console.log(gestionarHistorial("/usuarios")); //
["/usuarios","/productos", "/contacto", "/inicio"]
console.log(gestionarHistorial("/posts")); //
["/posts","/usuarios","/productos", "/contacto", "/inicio"]
console.log(gestionarHistorial("/servicios")); //
["/servicios","/posts","/usuarios","/productos", "/contacto"]
```

```
// Ejercicio 4: DOM e Interacción con API (40%)

// JavaScript (a completar):

// NO MODIFICAR LAS VARIABLES GLOBALES
// API Base URL
const API_URL = 'https://restcountries.com/v3.1';
// Elementos del DOM
const regionSelect = document.getElementById('region');
const sortNameRadio = document.getElementById('sort-name');
const sortPopulationRadio = document.getElementById('sort-population');
const countriesList = document.getElementById('countries-list');
const spinner = document.getElementById('spinner');
const errorAlert = document.getElementById('error-alert');
// Array para almacenar los países
let currentCountries = [];

// 1. Función para mostrar/ocultar elementos (ya implementada, NO
MODIFICAR)
function toggleElement(element, show) {
  if (show) {
    element.classList.remove('d-none');
  } else {
    element.classList.add('d-none');
  }
}

// 2. Función para mostrar errores (ya implementada, NO MODIFICAR)
function showError(message) {
  errorAlert.textContent = message;
  toggleElement(errorAlert, true);
  setTimeout(() => toggleElement(errorAlert, false), 5000);
}

// 3. Función para crear una tarjeta de país (ya implementada, NO
MODIFICAR)
function createCountryCard(country) {
  const card = document.createElement('div');
  card.className = 'col-md-3 col-sm-6 mb-3'; // Using updated Bootstrap
grid classes
  card.innerHTML = `
<div class="card h-100">
```

```
<img src="${country.flags.png}" class="card-img-top"
alt="${country.name.common} flag" style="height: 120px; object-fit:
cover;">
<div class="card-body p-2">
<h6 class="card-title">${country.name.common}</h6>
<p class="card-text small"><strong>Capital:</strong> ${country.capital
? country.capital[0] : 'N/A'}</p>
<p class="card-text small"><strong>Población:</strong>
${country.population.toLocaleString()}</p>
</div>
</div>
`;
  return card;
}

// 4. Añadir event listeners para los elementos interactivos (ya
implementada, NO MODIFICAR)
function setupEventListeners() {
  regionSelect.addEventListener('change', (event) => {
    loadCountriesByRegion(event.target.value);
  });
  sortNameRadio.addEventListener('change', sortCountries);
  sortPopulationRadio.addEventListener('change', sortCountries);
}

// 5. Inicializar la aplicación (ya implementada, NO MODIFICAR)
function initApp() {
  setupEventListeners();
  toggleElement(spinner, false);
  toggleElement(errorAlert, false);
  toggleElement(countriesList, true);
  sortNameRadio.disabled = true;
  sortPopulationRadio.disabled = true;
}

// 6. Función para ordenar países (ya implementada, NO MODIFICAR)
function sortCountries() {
  if (currentCountries.length === 0) {
    return;
  }
  if (sortNameRadio.checked) {
    currentCountries.sort((a, b) =>
a.name.common.localeCompare(b.name.common));
```

```
  } else if (sortPopulationRadio.checked) {
    currentCountries.sort((a, b) => b.population - a.population);
  }
  displayCountries();
}

// 7. Función para cargar países por región (COMPLETAR)
async function loadCountriesByRegion(region) {
  // A. Limpia la lista anterior de países (currentCountries) y el
contenido del contenedor (countriesList)
  currentCountries = [];
  countriesList.innerHTML = '';

  // B. Oculta la alerta de error (errorAlert) y el contenedor de
países (countriesList) mientras carga
  toggleElement(errorAlert, false);
  // No ocultar countriesList aquí, queremos que sea visible incluso si
está vacío o cargando

  // C. Deshabilita los botones de ordenación (sortNameRadio,
sortPopulationRadio) durante la carga y si no se ha seleccionado región
  sortNameRadio.disabled = true;
  sortPopulationRadio.disabled = true;

  // D. Si no se ha seleccionado ninguna región (el valor de 'region'
está vacío),
  // oculta el spinner, muestra el contenedor de la lista (vacío) y sal
de la función.
  if (!region) {
    toggleElement(spinner, false);
    toggleElement(countriesList, true); // Asegura que el contenedor
esté visible (aunque vacío)
    return;
  }

  // E. Muestra el spinner de carga (spinner)
  toggleElement(spinner, true);

  // F. Define la URL completa para la petición fetch usando API_URL y
la 'region' seleccionada
  const url = `${API_URL}/region/${region}`;
```

```
  // G. Inicia un bloque try...catch para manejar la petición asíncrona y los posibles errores
  try {
    // a. Realiza la llamada fetch a la URL
    const response = await fetch(url);

    // b. Verifica si la respuesta HTTP fue exitosa (response.ok). Si no, lanza un nuevo Error.
    if (!response.ok) {
      throw new Error(`Error HTTP: ${response.status} - ${response.statusText}`);
    }

    // c. Parsea la respuesta a JSON
    const data = await response.json();

    // d. Verifica si los datos recibidos son un array y no está vacío
    if (Array.isArray(data) && data.length > 0) {
      // i. Guarda el array de países en la variable global currentCountries
      currentCountries = data;
      // ii. Llama a la función sortCountries() para aplicar el ordenación inicial (por nombre, por defecto)
      //     (sortCountries también llama a displayCountries)
      sortNameRadio.checked = true; // Por defecto ordenar por nombre
      sortCountries();
      // iii. Habilita los radio buttons de ordenación (sortNameRadio, sortPopulationRadio)
      sortNameRadio.disabled = false;
      sortPopulationRadio.disabled = false;
    } else {
      // Si los datos no son un array válido o está vacío:
      // i. Llama a showError() indicando que no se encontraron países en esa región
      showError(`No se encontraron países en la región: ${region}`);
      // ii. Muestra el contenedor de la lista (countriesList) que ahora estará vacío
      displayCountries(); // Limpia y muestra lista vacía
    }
  } catch (error) {
    // Dentro del catch (manejo de errores de fetch o errores lanzados):
    // a. Llama a showError() con el mensaje del error
```

```
    showError(`Error al cargar los países: ${error.message}`);
    // b. Opcional: Muestra el error completo en la consola del
navegador para depuración
    console.error("Error completo al cargar países:", error);
    // c. Muestra el contenedor de la lista (countriesList) que ahora
estará vacío
    displayCountries(); // Limpia y muestra lista vacía
  } finally {
    // Dentro de un bloque finally (se ejecuta siempre al final del
try/catch):
    // a. Oculta el spinner de carga (spinner)
    toggleElement(spinner, false);
    // Asegurar que la lista de países (aunque esté vacía o con error)
sea visible
    toggleElement(countriesList, true);
  }
}

// 8. Función para mostrar la lista de países (COMPLETAR)
function displayCountries() {
  // A. Limpia todo el contenido HTML dentro del contenedor
countriesList
  countriesList.innerHTML = '';

  // B. Verifica si el array currentCountries tiene elementos. Si tiene
elementos:
  if (currentCountries && currentCountries.length > 0) {
    // Recorre el array currentCountries (puedes usar forEach o un
bucle for).
    currentCountries.forEach(country => {
      // i. Llama a createCountryCard() pasando el objeto del país
actual para obtener el elemento HTML de la tarjeta.
      const countryCard = createCountryCard(country);
      // ii. Añade (append) el elemento de la tarjeta recién creado al
contenedor countriesList.
      countriesList.appendChild(countryCard);
    });
  }
  // C. Asegúrate de que el contenedor countriesList sea visible
(aunque esté vacío después de limpiar si no hay países)
  toggleElement(countriesList, true);
}
```

```
// Iniciar la aplicación cuando el DOM esté cargado (NO MODIFICAR)
document.addEventListener('DOMContentLoaded', initApp);
```