mouredev pro

ENUNCIADOS EJERCICIOS CURSO DE JAVASCRIPT INTERMEDIO

Funciones avanzadas

1. Crea una función que retorne a otra función.

2. Implementa una función currificada que multiplique 3 números.

3. Desarrolla una función recursiva que calcule la potencia de un número elevado a un exponente.

4. Crea una función createCounter() que reciba un valor inicial y retorne un objeto con métodos para increment(), decrement() y getValue(), utilizando un closure para mantener el estado.

5. Crea una función sumManyTimes(multiplier, ...numbers) que primero sume todos los números (usando parámetros Rest) y luego multiplique el resultado por multiplier.

6. Crea un Callback que se invoque con el resultado de la suma de todos los números que se le pasan a una función.

7. Desarrolla una función parcial.

8. Implementa un ejemplo que haga uso de Spread.

9. Implementa un retorno implícito.

10. Haz uso del this léxico.

Estructuras avanzadas

1. Utiliza map, filter y reduce para crear un ejemplo diferente al de la lección.

2. Dado un array de números, crea uno nuevo con dichos números elevados al cubo y filtra sólo los números pares.

3. Utiliza flat y flatMap para crear un ejemplo diferente al de la lección.

4. Ordena un array de números de mayor a menor.

5. Dados dos sets, encuentra la unión, intersección y diferencia de ellos.

6. Itera los resultados del ejercicio anterior.

7. Crea un mapa que almacene información se usuarios (nombre, edad y email) e itera los datos.

8. Dado el mapa anterior, crea un array con los nombres.

9. Dado el mapa anterior, obtén un array con los email de los usuarios mayores de edad y transfórmalo a un set.

10. Transforma el mapa en un objeto, a continuación, transforma el objeto en un mapa con clave el email de cada usuario y como valor todos los datos del usuario.

Objetos y clases avanzados

1. Agregega una función al prototipo de un objeto.
2. Crea un objeto que herede de otro.
3. Define un método de instancia en un objeto.
4. Haz uso de get y set en un objeto.
5. Utiliza la operación assign en un objeto.
6. Crea una clase abstracta.
7. Utiliza polimorfismo en dos clases diferentes.
8. Implementa un Mixin.
9. Crea un Singleton.
10. Desarrolla un Proxy.

Asincronía

1. Crea una función para saludar que reciba un nombre y un callback.
   El callback debe ejecutarse después de 2 segundos y mostrar en consola "Hola, [nombre]".

2. Crea tres funciones task1(callback), task2(callback) y task3(callback).
   Cada función debe tardar 1 segundo en ejecutarse y luego llamar al callback.

3. Crea una función para verificar un número que retorne una Promesa.
   Si el número es par, la promesa se resuelve con el mensaje "Número par".
   Si el número es impar, la promesa se rechaza con el mensaje "Número impar".

4. Crea tres funciones que devuelvan promesas:
   firstTask(): tarda 1s y muestra "Primera tarea completada".
   secondTask(): tarda 2s y muestra "Segunda tarea completada".
   thirdTask(): tarda 1.5s y muestra "Tercera tarea completada".

5. Transforma el ejercicio anterior de Promesas en una función async/await llamada executeTasks().

6. Crea una función getUser(id) que devuelva una promesa y simule una llamada a una API (que se demore 2s).
   Si el id es menor a 5, la promesa se resuelve con { id, nombre: "Usuario " + id }.
   Si el id es 5 o mayor, la promesa se rechaza con el mensaje "Usuario no encontrado".
   Usa async/await para llamar a getUser(id) y maneja los errores con try/catch.

7. Intenta predecir el resultado de este código antes de ejecutarlo en la consola:
   ```
   console.log("Inicio")
   setTimeout(() => console.log("setTimeout ejecutado"), 0)
   Promise.resolve().then(() => console.log("Promesa resuelta"))
   console.log("Fin")
   ```

8. Crea tres funciones que devuelvan promesas con tiempos de espera distintos.
   A continuación, usa Promise.all() para ejecutarlas todas al mismo tiempo y mostrar "Todas las promesas resueltas" cuando terminen.

9. Crea una función waitSeconds(segundos) que use setTimeout dentro de una Promesa para esperar la cantidad de segundos indicada.
   A continuación, usa async/await para que se espere 3 segundos antes de mostrar "Tiempo finalizado" en consola.

10. Crea una simulación de un cajero automático usando asincronía.
    - La función checkBalance() tarda 1s y devuelve un saldo de 500$.
    - La función withdrawMoney(amount) tarda 2s y retira dinero si hay suficiente saldo, o devuelve un error si no hay fondos.
    - Usa async/await para hacer que el usuario intente retirar 300$ y luego 300$ más.

    Posible salida esperada:
    Saldo disponible: 500$
    Retirando 300$...
    Operación exitosa, saldo restante: 200$
    Retirando 300$...
    Error: Fondos insuficientes

APIs

1. Realiza una petición GET con fetch() a JSONPlaceholder y muestra en la consola la lista de publicaciones.

2. Modifica el ejercicio anterior para que verifique si la respuesta es correcta usando response.ok. Si no lo es, lanza y muestra un error.

3. Reescribe el ejercicio 1 usando la sintaxis async/await en lugar de promesas.

4. Realiza una petición POST a JSONPlaceholder para crear una nueva publicación. Envía un objeto con propiedades como title o body.

5. Utiliza el método PUT para actualizar completamente un recurso (por ejemplo, modificar una publicación) en JSONPlaceholder.

6. Realiza una petición PATCH para modificar únicamente uno o dos campos de un recurso existente.

7. Envía una solicitud DELETE a la API para borrar un recurso (por ejemplo, una publicación) y verifica la respuesta.

8. Crea una función que realice una solicitud GET (la que quieras) a OpenWeatherMap.

9. Utiliza la PokéAPI para obtener los datos de un Pokémon concreto, a continuación los detalles de la especie y, finalmente, la cadena evolutiva a partir de la especie.

10. Utiliza una herramienta como Postman o Thunder Client para probar diferentes endpoint de una API.

DOM

1. Crea un elemento (por ejemplo, un <h1 id="title">) y cambia su contenido a "¡Hola Mundo!"" al cargar la página.

2. Inserta una imagen con id="myImage" y cambia su atributo src a otra URL.

3. Crea un sin clases y agrega la clase resaltado cuando se cargue la página.

4. Crea un párrafo con id="paragraph" y cambia su color de texto a azul.

5. Agrega un botón que, al hacer clic, cree un nuevo elemento <li> con el texto "Nuevo elemento y lo agregue a una lista <ul id="list">.

6. Crea un párrafo con id="deleteParagraph" y un botón. Al hacer clic en el botón, elimina el párrafo del DOM

7. Crea un con algún texto y reemplaza su contenido por un <h2> con el mensaje "Nuevo Contenido".

8. Crea un botón con id="greetBtn" y añade un evento que muestre una alerta con el mensaje "¡Hola!" al hacer clic.

9. Crea un <input id="textInput"> y un . Al escribir en el input, el se debe actualizarse mostrando lo que se escribe.

10. Crea un botón con id="backgroundBtn" y, al hacer clic, cambia el color de fondo del <body> a un color diferente.

Depuración

1. Crea un código con un error lógico y usa VS Code para encontrarlo.

2. Experimenta con breakpoints y observa cómo cambia el flujo de ejecución.

Regex

1. Crea una RegEx que valide correos electrónicos.

2. Crea una RegEx obtenga Hashtags de un Texto.

3. Crea una RegEx que valide contraseñas seguras (mínimo 8 caracteres, al menos una letra y un número).

NOTA: Aplícalas utilizando diferentes operaciones.

Testing

1. Crea una función isEven(number) que devuelva true si el número es par y false si es impar.

2. Escribe una prueba en Jest para verificar que la función funciona correctamente.

3. Verifica que la prueba se ejecuta satisfactoriamente.


{m}pro