Guía de ejercicios - Callbacks y APIs

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada N°1: Implementación de Callbacks

Crear una función que devuelva el nombre "`John Doe`", el cual tiene que mostrarse en la consola del navegador y luego de ser obtenido, se debe implementar funciones con callback. Realizar el ejercicio con ES6 y ES5 para visualizar mejor el callback y su funcionamiento. Sigamos los siguientes pasos:

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea un archivo script.js. Luego, implementando ES6 realizar la función que retorne el nombre, esto lo podemos lograr creando primeramente una constante con el nombre getName para tener la función callback retornando el string "John Doe"..

```
const getName = callback => callback('John Doe')
```

- ***Paso 2***: Llamar a la función y pasarle como argumento la función anónima con el argumento "name" que será la respuesta que se obtenga de la función getName, luego mostramos en consola el string.

```
const getName = callback => callback('John Doe')
getName(name => console.log(name)) // output: John Doe
```

- ***Paso 3***: Al ejecutar el archivo script.js con ayuda de Node directamente desde la terminal con el comando: `node script.js`, el resultado encontrado seria:

```
John Doe
```

Veamos ahora el mismo ejemplo, pero escrito en ES5 para visualizar mejor el callback y su funcionamiento, partiendo de un nuevo archivo para no modificar el actual.

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea un archivo script.js. Luego, Implementando ES5 realicemos ahora la función con el nombre getName, la cual recibe un parámetro que será a su vez una función, y esta nueva función denominada "callback" llevará como argumento el string "John Doe".

```
function getName(callback) {
      callback('John Doe');
}
```

- ***Paso 2***: Posteriormente, se tiene que definir la función callback con el argumento "name", que será la respuesta de la función "getName", imprimiendo con un console.log el parámetro de la función, es decir, el string "John Doe".

```
function callback(name){
      console.log(name);
}
```

- ***Paso 3***: Luego, hacemos el llamado a la función "`getName`", pasando como argumento la función "`callback`".

```
getName(callback);
```

- **Paso 4**: Al ejecutar el archivo script.js con ayuda de Node directamente desde la terminal con el comando: `node script.js`, el resultado encontrado seria:

```
John Doe
```

Realicemos otro ejemplo, se solicita crear funciones con callbacks para obtener el cuadrado de un número. En este caso, utilizaremos la terminal y Node para ejecutar el archivo script creado. Por lo tanto:

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea un archivo llamado script.js. Seguidamente, en el archivo creado en el paso anterior declaramos la función `getSquare`, a la cual se le pasan dos argumentos, uno denominado "number", quien será el valor del cual queremos obtener el cuadrado multiplicando el número por sí mismo, y otro denominado "callback" que será la función que retorna el cuadrado del número enviado.

```
const getSquare = (number, callback) => callback(number * number);
```

- ***Paso 2***: Ahora hacemos el llamado a la función `getSquare`, pasando como argumento el número del cual calcularemos el cuadrado y el nombre que le daremos a la función callback, para recibir el valor y mostrarlo por la consola. Implementamos el llamado a la función varias veces para obtener distintos resultados.

```
const getSquare = (number, callback) => callback(number * number);
getSquare(4, result => console.log(result));
getSquare(2, result => console.log(result));
getSquare(5, result => console.log(result));
getSquare(12, result => console.log(result));
```

- ***Paso 3***: Ejecutamos el archivo script.js en la terminal con ayuda de Node mediante la instrucción: `node script.js`, obteniendo como resultado:

```
16
4
25
144
```

¡Manos a la obra! - Ejercicios propuestos I

- Ejecuta el código mostrado a continuación en la página web Loupe, y describe el comportamiento con tus propias palabras.

```
var A = [1, 2], B = [];

for (var i = 0; i < A.length; i++) {
   B.push(sumarDos(A[i]));
};
function sumarDos(x) {
   return x + 2;
};
console.log(B)
```

- Desarrollar un programa con JavaScript implementando funciones con callback, que al pasarle los puntos obtenidos en dos ejercicios de un examen, sume ambos puntos e indique si hemos superado la prueba. El examen tiene una ponderación total de 10 puntos. Cada ejercicio tiene una nota máxima de 5 puntos.
- Crear la función "separar", donde se pasan dos (2) argumentos, un arreglo de números y un callback. La función deberá devolver un objeto con dos (2) arreglos, uno con los pares y otro con los impares. Ejemplo: Si se tiene el arreglo [3,12,7,1,2,9,18]. La función debe retornar: pares: [12,2,18], impares: [3,7,1,9].

- Crear cuatro (4) funciones con las operaciones básicas matemáticas, sumar, restar, multiplicar y dividir, a las cuales se les debe enviar dos valores, el primer valor siempre será más grande que el segundo. Este segundo valor, no puede ser 0. Los valores deben ser devueltos en callbacks.

Actividad guiada N°2: Callbacks anidados

Crear tres funciones que se utilizarán con un solo llamado, dos de ellas se ejecutarán dentro de la primera, y devolverán el resultado en la función callback. La primera función debe retornar el doble del argumento pasado, mientras que la segunda función debe retornar el cuadrado del argumento y por último la tercera función debe retornar el 25% del número. Para ello, ejecutamos los siguientes pasos:

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crear un archivo con el nombre de script.js. Ahora, en el archivo script.js comenzamos por definir las funciones principal, multiply y calculate25Percent como constantes. La función principal multiplica el número pasado por argumento por dos (2). Mientras que la función multiply recibe como argumento el resultado de la multiplicación y la función calculate25Percent le pasamos por argumento el resultado de multiply.

```
const principal = (number) => {
   const double = number * 2;
}
const multiply = (number, callback) => callback();
const calculate25Percent = (number, callback) => callback();
```

- ***Paso 2***: En la función principal, se debe ejecutar el resto del proceso, el cual corresponde a las llamadas de las otras dos funciones y mostrar el resultado en un console.log en el último llamado a la función, es decir, a la función calculate25Percent.

```
const principal = (number) => {
      const double = number * 2;
// llamamos a multiply y le pasamos el resultado anterior
      multiply(double, square => {
// llamamos a calculate25Percent y le pasamos el resultado de B
          calculate25Percent(square, percent => {
// enviamos a la consola el resultado obtenido de calculate25Percent
              console.log(percent)
          })
      })
}

const multiply = (number, callback) => callback();
const calculate25Percent = (number, callback) => callback();
```

- ***Paso 3***: La función multiply recibe como argumento el resultado de la multiplicación y a su vez se encarga de devolver el cuadrado del número recibido, mientras que la función calculate25Percent, a la que pasamos por argumento el resultado de multiply, devuelve el 25% del resultado recibido. Finalmente hacemos el llamado de la función principal enviando el número 5.

```
const principal = (number) => {
      const double = number * 2
      multiply(double, square => {
          calculate25Percent(square, percent => {
               console.log(percent)
          })
      })
}

const multiply = (number, callback) => callback(number * number);
const calculate25Percent = (number, callback) => callback(number * 25 /
100);
principal(5);
```

- ***Paso 4***: Ejecutamos el archivo script.js en la terminal con ayuda de Node mediante la instrucción: `node script.js`, obteniendo como resultado:

```
25
```

Ahora, en pocas palabras al ejecutar el código anterior enviando como parámetro el número cinco (5), lo que ocurrió fue: comienza la ejecución de la función principal y se le pasa como argumento el número 5; esta función multiplica el 5 * 2 obteniendo 10. Luego, la función principal ejecuta la función multiply pasándole como argumento este resultado y que multiplica por sí mismo y devuelve como resultado 100. Posteriormente, terminada la ejecución de multiply pasa a ejecutar calculate25Percent enviándole el argumento 100 que obtuvo de multiply, y en el que se obtiene el 25% de éste, o sea, el 25% de 100 y nos da como resultado 25 que es lo que se mostraría en el console.log(percent).

¡Lo lograste! / ¡Felicitaciones!

¡Manos a la obra! - Ejercicios propuestos II

- Crear un programa con JavaScript mediante el uso de funciones con Callback y el método setTimeout, que permita mostrar los datos de un usuario de acuerdo al nombre o el número de identificación. Los datos serán los siguientes:

```
usuario1 = {id: 2356256,name: 'Juan',lastName: 'Duran',age: 35}
usuario2 = {id: 27564512,name: 'Manuel',lastName: 'Perez',age: 31}
usuario3 = {id: 17658624,name: 'Jocelyn',lastName: 'Rodriguez',age: 30}
usuario4 = {id: 12345678,name: 'Maria',lastName: 'Garrido',age: 30}
```

- Crear un programa con JavaScript utilizando promesas que calcule un número aleatorio entre el "1" y el "100", pero que muestre el número aleatorio si y sólo si este número está comprendido entre "20" y "60", ambos valores incluidos.

- Realizar un programa en JavaScript que calcule la suma o la resta o la multiplicación de dos números. Para ello se debe implementar una promesa por cada operación matemática y solo se debe mostrar el resultado de la promesa ganadora. Utiliza tiempos aleatorios para los setTimeout que poseen cada promesa con su operación matemática. Igualmente implementa Promise.race para mostrar sola la primera promesa que retorne el resultado de de la operación matemática ganadora.

Actividad guiada N°3: Combinando promesas y eventos en el DOM

Solicitan que al hacer un clic sobre un botón, se ejecute una promesa que retorne y muestre por pantalla tres mensajes:

- Primero: "Solicitando Autorización"
- Segundo: "Esperando la información"
- Tercero: "El usuario en línea es: Juan".

Estos mensajes se deben insertar en el documento utilizando el DOM dentro de una función externa.

Para generar el primer mensaje se debe utilizar una función que contenga una promesa y tarde 2.5 segundos en indicar que todo está correcto, es decir, que tiene autorización

retornando “true”, mientras que el segundo y tercer mensaje se origina en otra función que contiene una promesa que muestra primeramente el segundo mensaje y al cabo de 2,5 segundos muestra el tercer y último mensaje.
Ahora bien, las promesas se deben activar cuando el usuario haga un click sobre el botón denominado “Ejecutar”, y mientras la respuesta sea verdadera “true” se deben mostrar los dos últimos mensajes. Para ello, ejecutamos los sigamos pasos:

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js. Seguidamente, en el index.html debes escribir la estructura básica de un documento HTML como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Promesas</title>
</head>
<body>
   <h4>Usando ES6 - Promesas</h4>
   <section id="contenido"></section>
   <button id="boton">Ejecutar</button>
   <script src="script.js"></script>
</body>
</html>
```

- ***Paso 2***: En el archivo script.js, lo primero es agregar un escucha al botón, para que cuando el usuario haga un click sobre él, se activen los llamados a la funciones que contienen las promesas. La primera función que debe llamar será la que retorna el mensaje de autorización con el nombre de `getData`, es decir, si el usuario está o no autorizado, en este caso y para este ejemplo, el retorno deberá ser “true”. A ella, se le concatena un “*then*” para esperar la respuesta de la promesa, si la promesa retorna true, entonces se debe regresar el llamado a la función que permita mostrar los datos (mensajes dos y tres), esta función llevará por nombre `mostrarData()`. Ahora, esta última función también contiene una promesa, por lo tanto se debe concatenar el “*then*” para así poder mandar a mostrar por pantalla el último mensaje con el nombre del usuario.

```
boton.addEventListener('click',()=>{
      getData().then(autorizacion => {
         if(autorizacion){
            return mostrarData();
```

```
        };
    }).then(usuario => {
        setTexto(`El usuario en línea es: ${usuario.nombre}`);
    });
});
```

- ***Paso 3***: Agregamos ahora la lógica interna de la función `getData()`. Ya que es la primera en ser invocada. En esta función se retorna una nueva promesa y dentro de la promesa se debe llamar a la función encargada de mostrar el contenido en el documento mediante el DOM `'Solicitando Autorización'`. Luego de transcurrir los 2.5 segundos del método setTimeout, se resuelve con true la promesa.

```
const getData = () => {
    return new Promise((resolve,reject)=>{
        setTexto('Solicitando Autorización');
        setTimeout(()=>{
            resolve(true);
        },2500);
    });
};
```

- ***Paso 4***: Debemos trabajar ahora con la segunda función, esta función es convocada si y sólo si la primera promesa retorna "true". Dentro de ella se retorna una nueva promesa, en donde se llamará nuevamente a la función `setTexto` para que muestre el mensaje `'Esperando la información'`, luego de transcurrir 2.5 segundos se debe resolver la promesa con el dato: `{nombre:"El usuario en línea es: Juan"}`. Que será parte del último mensaje a mostrar por pantalla llamando a la función `setTexto` pero al retornar la promesa.

```
const mostrarData = () => {
    return new Promise((resolve,reject)=> {
        setTexto('Esperando la información');
        setTimeout(()=>{
            resolve({nombre:"Juan"});
        },2500);
    });
};
```

- ***Paso 5***: Ya solo queda crear la función que incrustará los mensajes en el documento, específicamente en el elemento con el "id" igual a "contenido", mediante la propiedad `textContent` le igualamos el valor que traen el parámetro.

```
const setTexto = datos => {
      contenido.textContent = datos;
};
```

- ***Paso 6***: Finalmente, ejecutamos el archivo index.html con nuestro navegador web, y hacemos un click sobre el botón para obtener la secuencia de resultados como se muestra en la imagen:

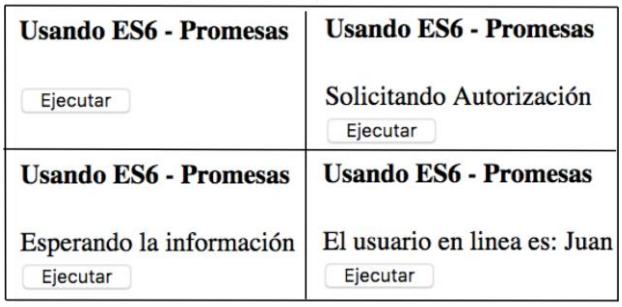

Imagen1. Secuencia de respuestas en el documento web.
Fuente: Desafío Latam

¡Manos a la obra! - Ejercicios propuestos III

- Convertir el siguiente código para que se pueda usar promesas.

```
const log = (text, callback) => {
   setTimeout(() => {
       console.log(`${text}`);
       callback()
   }, 1000)
}
log('uno', () => {
   log('dos', () => {
       log('tres', () => { })
   })
})
```

- Utilizando promesas, crea la función **verify** al que se le pasen dos argumentos **a** y **b**, (deben ser de tipo numérico), que los compare y resuelva la promesa si el primer número es mayor que el segundo calculando la potencia del número “a” elevado al

número “b” , y rechace la promesa en caso contrario indicando que no se puede realizar la operación, muestre los datos en la consola.

- Crea una función que permita traer y mostrar por la consola del navegador web la información de la siguiente dirección web: https://www.feriadosapp.com/api/holidays.json. Implementa Async...Await

Actividad guiada N°4: Promise.race, la respuesta más rápida

En un colegio, la profesora decide dar incentivos a quien responda primero las preguntas que hace. El que lo haga obtendrá décimas para la próxima evaluación. Selecciona a Carlos, María y Cristian para que respondan. Los estudiantes piensan y dan su respuesta casi al mismo tiempo, ¿quién da la respuesta más rápido?

Para realizar este ejemplo, se deben seguir los siguientes pasos:

- ***Paso 1***: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea un archivo script.js. Luego, en el archivo script.js, creamos primeramente la función llamada `randomNumber` que retorna el número aleatorio, recibiendo dos valores (uno mínimo y uno máximo), para ayudar al método Math.random() a generar el número entre la diferencia de los dos números recibidos por la función. Por consiguiente, usaremos esta función como tiempo en que se demoran los niños en responder.

```
const randomNumber = (min, max) => {
   return parseInt(Math.random() * (max - min) + min);
}
```

- ***Paso 2***: Definir la función `responder`, que se encarga de esperar el tiempo aleatorio que tardan en responder y devuelve el nombre del niño que respondió. El tiempo será establecido entre 500 y 700 ms.

```
const responder = alumno => new Promise((resolve, reject) => {
   setTimeout(() => { resolve(alumno) }, randomNumber(500,700))
})
```

- ***Paso 3***: Definir una nueva función, la cual, tendrá el `Promise.race` y se encargará de mostrar la promesa que retorne más rápido una respuesta, indicando el mensaje en la consola del navegador web: `'La estrella es para:'.`

```
const pregunta = (alumnos) => {
   Promise.race(alumnos)
   .then(response => console.log('La estrella es para:', response))
}
```

- ***Paso 4***: Ya solo queda iniciar una variable por cada alumno llamando a la función `responder` creada anteriormente y pasando como parámetro el nombre del alumno.

```
const Carlos = responder('Carlos')
const Maria = responder('Maria')
const Cristian = responder('Cristian')
const alumnos = [Carlos, Maria, Cristian];
pregunta(alumnos)
```

- ***Paso 5***: Al final ejecutamos las promesas de forma simultánea y esperamos a que llegue la respuesta con el niño que respondió más rápido.

```
La estrella es para: Cristian
```

En este caso el alumno que respondió más rápido fue Cristian.

¡Manos a la obra! - Ejercicios propuestos IV

- Utilizando el método fetch y promesas, muestra en la consola del navegador web la información suministrada por la API: https://swapi.dev/api/people/

- Realizar un programa en JavaScript que permita mostrar todos los post realizados en una página web y luego, de acuerdo al autor de cada post, mostrar la información por individual de cada usuario. Toda la información debe mostrarse en la consola del navegador web. Los post los puedes encontrar en: http://demo.wp-api.org/wp-json/wp/v2/posts, y la identificación del autor se encuentra en el atributo con el nombre author. La información de cada usuario la puedes encontrar en: https://demo.wp-api.org/wp-json/wp/v2/users/ (id).

- Realizar un programa en JavaScript que permita mostrar la información exclusiva del personaje de la serie animada Rick and Morty llamado “Rick Sánchez”, además de todos los residentes disponibles en la misma locación o ubicación del personaje. Toda la información debe mostrarse en la consola del navegador web. El personaje lo puedes encontrar en: https://rickandmortyapi.com/api/character/1, mientras que la

ubicación con todos los residentes del lugar que habita el personaje la puedes ubicar en: https://rickandmortyapi.com/api/location/1. El parámetro común en este caso es el id correspondiente al número 1.

- Realizar un programa con JavaScript que utiliza la API mindicador para obtener todos los indicadores económicos. Para ello, debes crear las funciones para obtener los valores de: Dólar, Euro UF, UTM, IPC, agregándole al final de la url el código que quieres obtener, estos son: dolar, euro, uf, utm e ipc. Por consiguiente, se debe listar la fecha y los valores de la propiedad "serie" y utilizar promesas para obtener todos los valores a la vez. Ejemplo de la URL: https://mindicador.cl/api/dolar, https://mindicador.cl/api/ipc

¡Lo lograste! / ¡Felicitaciones!

Actividad guiada N°5: Try… catch.. finally

Mostrar tres valores de una variable que contiene distintos tipos de datos. Los dos primeros valores a mostrar deben estar en los datos de la variable indicada, mientras que el tercero no debe existir. Se debe utilizar los bloques de control try...catch...finally.

- ***Paso 1***: Abre tu navegador web preferido y ve a la consola. Luego, vamos a declarar una constante denominada "json" con los datos de un post ficticio.

```
const json = {
  "articles": [{
    "id": "1",
    "title": "Mi primer artículo",
    "body": "Nunca había escrito un artículo."
  }, {
    "id": "2",
    "title": "Mi segundo artículo",
    "body": "Este es mi segundo artículo."
  }]
}
```

- ***Paso 2***: Seguidamente sobre la misma consola del navegador web, trabajamos con la expresión `try` para mostrar con console.log los títulos de los artículos que se encuentran disponibles en la posición "0, 1 y 2" del arreglo `"articles"`. En este caso estaremos generando un error voluntariamente ya que la posición dos [2]:

`json.articles[2].title` del arreglo no existe. Este error nos servirá para trabajar con la estructura `catch`..

```
try {
    console.log(json.articles[0].title);
    console.log(json.articles[1].title);
    console.log(json.articles[2].title);
}
```

- ***Paso 3***: Ahora para poder atrapar los errores, vamos a crear la estructura `catch`, dentro de ella mostraremos en la consola el nombre del error y el mensaje que este genera. Recuerda que el error es atrapado como un objeto al cual podemos acceder a sus propiedades.

```
catch(e) {
    console.log('Errores');
    console.log(e.name, e.message);
}
```

- ***Paso 4***: Finalmente, agregamos el `finally` para mostrar el último mensaje por defecto cuando todo se cumpla.

```
finally {
    console.log('Finalizado el try...catch. Este mensaje aparece
siempre.');
}
```

- ***Paso 5***: Ejecutar el código en la consola del navegador web, encontraremos el siguiente resultado.

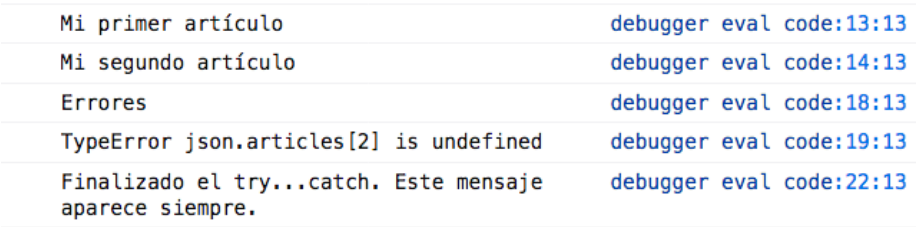

Imagen 2. Resultado de la consola del navegador.
Fuente: Desafío Latam

La forma en que se ejecutan los bloques lo podemos visualizar en el siguiente diagrama de flujo.

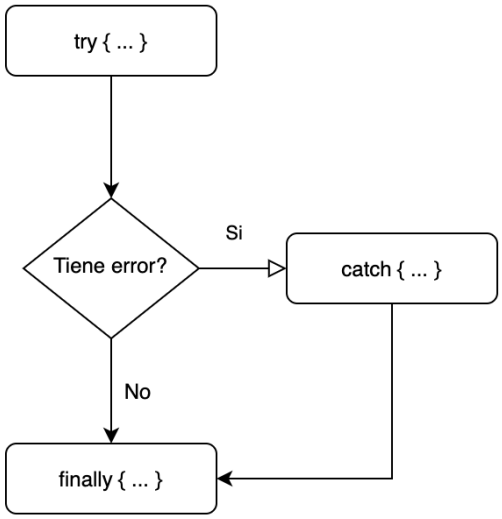

Imagen 3. Diagrama de flujo de try, catch, finally.
Fuente: Desafío Latam

¡Manos a la obra! - Ejercicios propuestos V

- Del siguiente código y sin ejecutar en el navegador, ¿Cuál crees que sería el error que se mostrará en la consola? Verifica ahora tu respuesta ejecutando el código directamente en la consola de navegador web.

```
var a = new Array(4294967295);
var b = new Array(-1);

var num = 2.555555;
document.writeln(num.toExponential(4));
document.writeln(num.toExponential(-2));

num = 2.9999;
document.writeln(num.toFixed(2));
document.writeln(num.toFixed(105));

num = 2.3456;
document.writeln(num.toPrecision(1));
document.writeln(num.toPrecision(0));
```

- Realizar una función que genere un error en el bloque del “try” indicando que las variables no están declaradas. Para atrapar el error se debe implementar el “catch”.

- Desarrollar un programa con JavaScript que permita indicar a un alumno, si su respuesta de verdadero o falso estuvo acertada o no. Utiliza promesas, then...catch y funciones para generar el código.

- De los datos indicados a continuación. Muestra por consola los valores del nombre y apellido de cada persona, incluyendo una tercera persona ficticia, utilizando la estructura try...catch...finally. Indicando el nombre del error y el mensaje en una ventana con la instrucción alert.

```
const json = {
  "persona": [{
    "id": "1",
    "nombre": "Juan",
    "apellido": "Romero."
  }, {
    "id": "2",
    "nombre": "Jocelyn",
    "apellido": "Perez."
  }]
}
```

- Crear la función **whereToSee** a la que se pueda pasar el argumento **direction**, que tiene 2 valores posibles, **left** y **right**. La función debe retornar siempre lo opuesto a lo que se envía. Se deben manejar los errores en caso de que se envíe a la función otro valor que no sean los posibles, retornando el error con la clase Error.

- Crea una función con el nombre de ask y luego crea dos (2) variables denominadas “num1 y num2”, a ambas variables se le deben asignar la función prompt para solicitar un número cualquiera. Posteriormente se debe validar que realmente se ingresaron los dos números y devolver la suma de ambos. Pero, si no se puede realizar la suma porque el usuario no ingresó ambos números se debe devolver un error mediante el levantamiento de una excepción con throw.

Preguntas de proceso

Reflexiona:

- ¿Qué he aprendido hasta ahora?
- ¿Hay algo que me está dificultando mucho?
- ¿Por qué es importante entender el concepto de callback para desarrollar aplicaciones web?
- ¿Qué sucede si se omite la llamada a la función callback en JavaScript?
- ¿Cuál es la ventaja de utilizar una función con callback en lugar de una función normal en JavaScript?
- ¿Por qué es importante utilizar la estructura try...catch...finally en el manejo de errores en JavaScript?

¡Continúa aprendiendo y practicando!