Callbacks y APIs

Manejo de errores

{desafío} latam_

Utilizar elementos de programación asíncrona para resolver un problema simple distinguiendo los diversos mecanismos para su implementación acorde al lenguaje Javascript.

Utilizar el objeto XHR y la API Fetch para el consumo de una API externa y su procesamiento acorde al lenguaje Javascript.

{desafío} latam_

- Unidad 1:
  ES6+ y POO
- Unidad 2:
  Herencia
- Unidad 3:
  Callbacks y APIs

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Utiliza generación y captura de errores personalizados para resolver un problema simple de programación asíncrona acorde al lenguaje Javascript.*

{desafío}
latam_

¿Hay algo que quieras consultar sobre la clase anterior?

{desafío} latam_

/* Excepciones */

Excepciones

- Las excepciones son imprevistos que ocurren durante la ejecución de una aplicación y que provocan que estas no funcionen de la manera que se espera.

- Para el caso de una promesa, cuando esta se rechaza, automáticamente se vuelve un error y debemos continuar la ejecución con **.catch**.

- En otros casos de excepción vamos a crear y poner nuestro código dentro del bloque **try/catch/finally** para que nuestras aplicaciones no interrumpan su flujo normal.

/* Tipos de errores */

Tipos de errores

Error: Permite establecer un mensaje de error personalizado.

```
try {
    throw new Error("Ups! Ha ocurrido un error");
} catch (e) {
    console.log(e.name + ": " + e.message);
}
```

RangeError: Ocurre cuando un número está fuera del rango permitido por el lenguaje.

```
let a = []; a.length = a.length - 1;
```

Tipos de errores

ReferenceError: Ocurre cuando se hace referencia a variables no declaradas.

```
const x = y;
```

SyntaxError: Ocurre cuando hay un error de sintaxis en nuestro código.

```
function getValue(){
    return 2;
};
getValue();
```

Tipos de errores

TypeError: Ocurre cuando un valor no es del tipo esperado.

```
const x = {};
x.y();
```

URIError: Ocurre cuando se codifica o decodifica una URL utilizando las funciones encodeURI, decodeURI, encodeURIComponent o decodeURIComponent.

```
decodeURIComponent("http://%google.com");
```

Tipos de errores

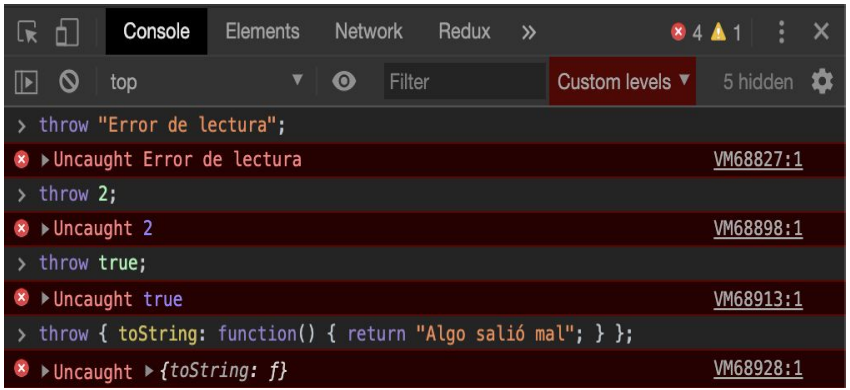
EvalError: Ocurre cuando hay un error al evaluar una expresión con la función eval. Hay que mencionar que evaluar expresiones con esta función es una mala práctica, por lo que no se debería usar.

/* Throw */

Throw

Este comando permite enviar al navegador una excepción tal cual como si de una real se tratase.

```
throw "Error de lectura";   // Tipo string
throw 2;          // Tipo número
throw true;        // Tipo booleano
throw { toString: function() { return "Algo salió mal"; } }; // tipo objeto
```



{desafío} latam_

Ejercicio guiado: Excepciones personalizadas con Throw

{desafío}
latam_

Ejercicio guiado

Excepciones personalizadas con Throw

Simular un error levantando una excepción con “Throw” que indique que “Ha ocurrido un error” dentro del bloque try para ser atrapada con el bloque catch. Para ello, sigue los siguientes pasos:

Paso 1: Abre tu navegador web preferido y ve a la consola. Seguidamente, sobre ella, vamos a copiar un código, en este caso, como vamos a simular un error y probar la estructura Try...Catch, crea una constante con el nombre de `showError`, sobre ella una función sin parámetros mediante ES6. Dentro de la función, establecemos el primer bloque “try” para ejecutar y forzar un error personalizable mediante la sentencia throw con un mensaje: `'Ha ocurrido un error'`.

```
const showError = () => {
    try {
      throw 'Ha ocurrido un error';
    }
}
```

{desafío} latam_

Ejercicio guiado

Excepciones personalizadas con Throw

Paso 2: Ahora agregamos el bloque “catch” para atrapar el error personalizado creado en el bloque try. Recibiendo como parámetro con la letra “e” y mostrando en consola el parámetro.

```
const showError = () => {
    try {
      throw 'Ha ocurrido un error';
    }
    catch(e) {
        console.log(e)
    }
}
showError();
```

Paso 3: Finalmente hacemos el llamado a la función showError(), y observamos el resultado con un correcto manejo del error.

```
Ha ocurrido un error
```

{desafío} latam_

¿Te quedó alguna duda sobre el ejercicio?

{desafío} latam_

/* Método catch */

Método catch

- Para el manejo de errores en las promesas utilizaremos el método catch, en el que viene como argumento el detalle de la excepción retornada por esta, es decir, el detalle del error ocurrido durante el proceso.

- Recordar que debemos devolver la promesa rechazada (reject) para que podamos utilizar el método de manejo de error.

Ejercicio guiado: Método catch

{desafío} latam_

Ejercicio guiado

Método catch

Crear una función que retorne una promesa, pero en este caso forzaremos el error, rechazando la promesa y creando un error.

Ahora sigamos los pasos a continuación para realizar este ejemplo.

Paso 1: Abre tu navegador web preferido y ve a la consola. Seguidamente, vamos a copiar un código directamente sobre la consola del navegador web, en este caso, como vamos a rechazar una promesa y probar la estructura Try...Catch, crea una función con el nombre de showError, sobre ella retornamos primeramente una nueva promesa, utilizando como parámetros (resolve y reject), luego dentro de la promesa utiliza el reject para retornar un nuevo error con el mensaje "Ha ocurrido un error!".

```
function showError () {
    return new Promise((resolve, reject) => {
      reject(new Error("Ha ocurrido un error!"));
    })
}
```

{desafío} latam_

Ejercicio guiado

Método catch

Paso 2: Finalmente, se debe llamar a la showError y como esta función retorna una promesa, debemos prepararnos para recibir la respuesta, ya sea correcta (resolve) o regrese un error por cualquier situación (reject). En el caso de retornar una respuesta satisfactoria el then la recibiría y mostraría por la consola. Pero, como ya sabemos que estamos rechazando la promesa, es decir, está retornando un error, entonces el catch se encargará de atraparla y mostrar el mensaje que trae el objeto.

```
showError()
.then(resolve =>{
    console.log(resolve)
})
.catch(err => {
    console.log(err.message)
})
```

Paso 3: Ejecutar el código anterior en la consola del navegador web, el resultado mostrado sería el siguiente:

```
Ha ocurrido un error!
Promise { <state>: "pending" }
```

/* Try, Catch, Finally */

Try, Catch, Finally

Es un bloque de control en el que podemos controlar las excepciones imprevistas y en la que podemos determinar qué realizar al momento de un error y si debemos continuar con la ejecución de nuestra aplicación o debemos detener el flujo.

```
try {

} catch (error) {

} finally {

}
```

Try, Catch, Finally

El bloque **try** se encarga de ejecutar todas las instrucciones que posea y evalúa en cada una si se ejecutó de forma correcta.

El bloque **catch** permite obtener cuál fue el error que se produjo en el bloque try, en el que podemos acceder a la información del error mediante una variable.

El bloque **finally** permite que siempre se puedan ejecutar sentencias posterior a try y catch, habiendo evaluado de forma correcta el try o si es que hubiera algún error en catch, sí o sí ejecutará lo que está en este bloque.

/* Buenas prácticas en el manejo de errores */

Buenas prácticas en el manejo de errores

- Utilizar el bloque try...catch...finally en el que podemos capturar errores lógicos y de ejecución.
- El uso de throw en combinación con try..catch para generar excepciones propias y controlar la aplicación de mejor manera.
- El método onerror del objeto window permite capturar los errores en tiempo de ejecución.
- Se puede utilizar onerror en el elemento img de html para capturar el error cuando no existe la imagen cargada.

Ejercicio guiado: Depurando errores

{desafío} latam_

Ejercicio guiado

Depurando errores

Acceder al stack desde el bloque catch, mediante la construcción de tres funciones, donde la primera función pase valores a una segunda función que no los reciba y a su vez llame a una tercera función que tenga la estructura try...catch, forzando un error desde el try para acceder al stack de ese error en el catch.

Paso 1: Abre tu navegador web preferido y ve a la consola. Seguidamente, vamos a declarar crear la primera función llamada trace(), que contenga el bloque try...catch. Para generar y atrapar el error.

```
function trace(){
    try {
      throw new Error('miErrorpersonalizado');
    }
    catch(e) {
        console.log(e.stack);
    }
}
```

{desafío} latam_

Ejercicio guiado

Depurando errores

Paso 2: Seguidamente, creamos la segunda función que no recibirá ningún parámetro y llamará a la función trace para ejecutar la estructura try...catch.

```
function b(){
   trace();
}
```

Paso 3: Luego, creamos la función que llamará a la función creada anteriormente b(), pasando algunos parámetros, creando así la cadena de llamados.

```
function a(){
   b(1,'texto',undefined,{});
}
```

Ejercicio guiado

Depurando errores

Paso 4:: Al ejecutar todo este código en la consola del navegador web, el resultado obtenido será:

```
a();
```

Paso 5: Queda por agregar al final el llamado a esta primera función denominada a(), para que se ejecuten todas las funciones en cadena y se genere el resultado en la consola del navegador web.

```
Error: miErrorPersonalizado                                    VM15889:5
    at trace (<anonymous>:3:15)
    at b (<anonymous>:10:5)
    at a (<anonymous>:14:5)
    at <anonymous>:17:1
```

Consideraciones para controlar errores en funciones que se ejecutan de manera asíncrona:

- Es conveniente realizarlo con promesas, ya que en caso de haber error, se puede rechazar la promesa al finalizar el proceso.

- Funciona de la misma forma en una “función asíncrona” (declarada con async) y maneja los errores con el método catch de la promesa retornada.

- Lo que no se puede realizar, es manejar los errores con try..catch en funciones asíncronas, ya que éste es síncrono y por lo tanto jamás capturará el error, puede parecer como que no ocurriese nada.

Comenten con sus palabras:

¿Para qué nos sirve el manejo de errores?

Resumiendo

- Las excepciones son imprevistos que ocurren durante la ejecución de una aplicación y que provocan que estas no funcionen de la manera que se espera.
- Para el caso de una promesa, cuando esta se rechaza, automáticamente se vuelve un error y debemos continuar la ejecución con .catch.
- Try CatchEs un bloque de control en el que podemos controlar las excepciones imprevistas y en la que podemos determinar qué realizar al momento de un error y si debemos continuar con la ejecución de nuestra aplicación o debemos detener el flujo.

Próxima sesión...

- *Desafío guiado - Datos de usuarios*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales