{desafío}
latam_

Callbacks y APIs _

Parte II

Trabajar con APIs

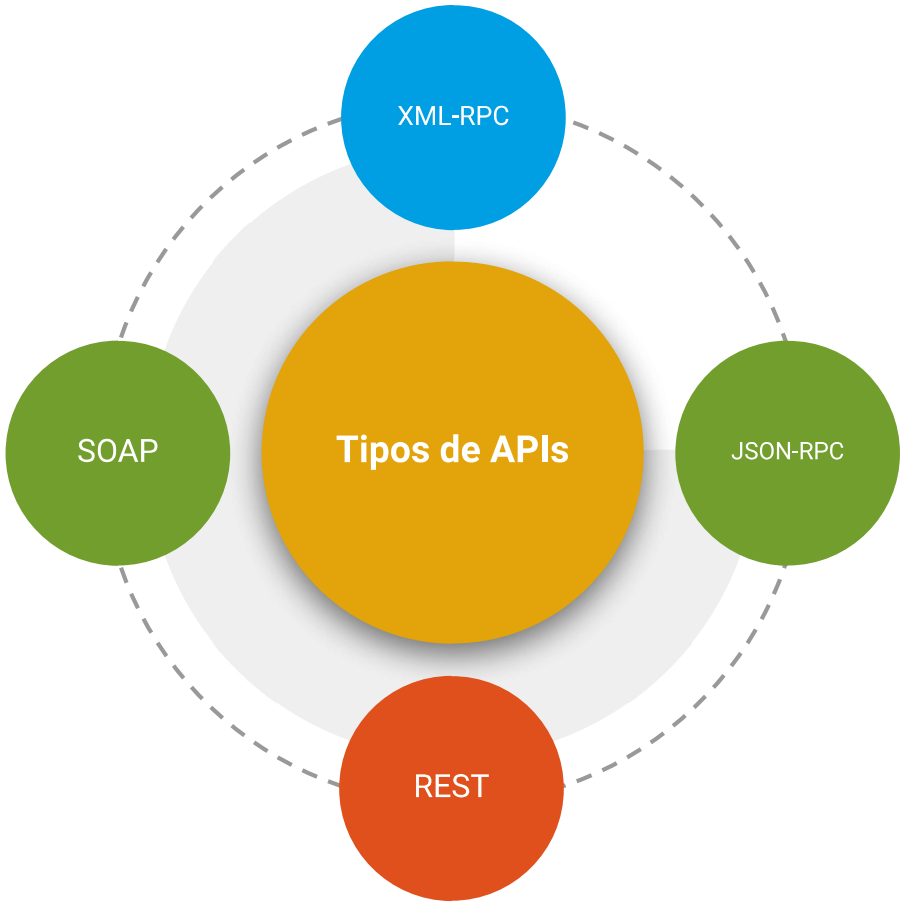
{desafío}
latam_

Competencias

- Consultar múltiples API utilizando Ajax y promesas para controlar el orden de los llamados.
- Determinar cuáles métodos utilizar para manejar el flujo al trabajar con Promesas

¿Qué es una API?

- **API**: Interfaz de Programación de Aplicaciones (Application Programming Interface).
- Es una especificación formal sobre cómo un módulo de un software se comunica o interactúa con otro.
- Una de las principales funciones de las API es poder facilitar el trabajo a los desarrolladores, ahorrar tiempo y dinero.


XML-RPC
SOAP
Tipos de APIs
JSON-RPC
REST

SOAP (Simple Object Access Protocol)

- Sirve para que dos procesos puedan comunicarse intercambiando datos XML.
- Se usa para exponer servicios web a través del protocolo HTTP.
- Soporta sólo XML como formato de datos.

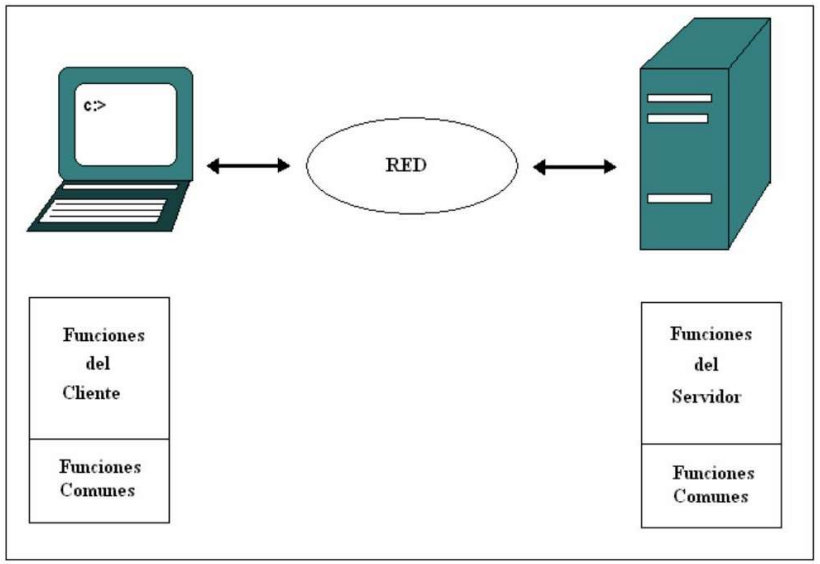
```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-enc
oding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>.
```

REST (Representational State Transfer)

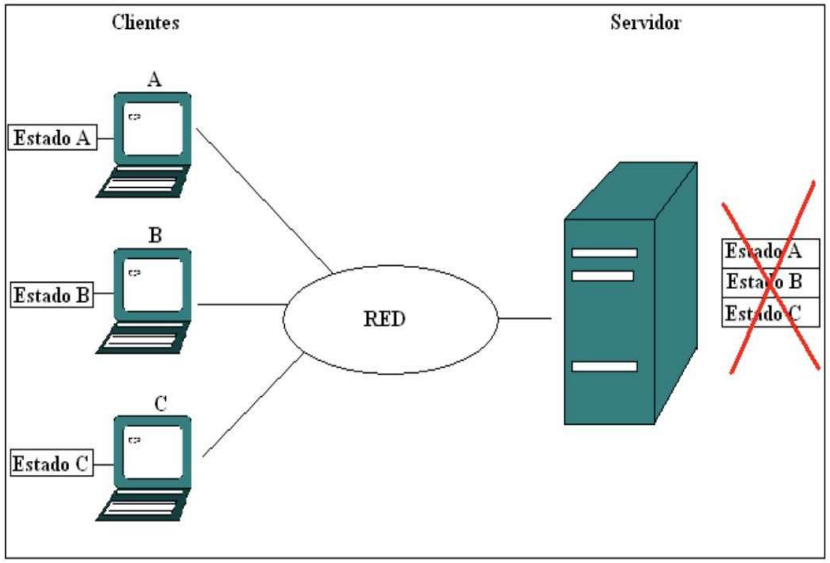
- Interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

- La arquitectura REST define algunas restricciones o características para ser utilizada, como el caso de: Cliente/Servidor, Sin estado, Caché, Sistema de capas, Interfaz uniforme y Código bajo demanda.

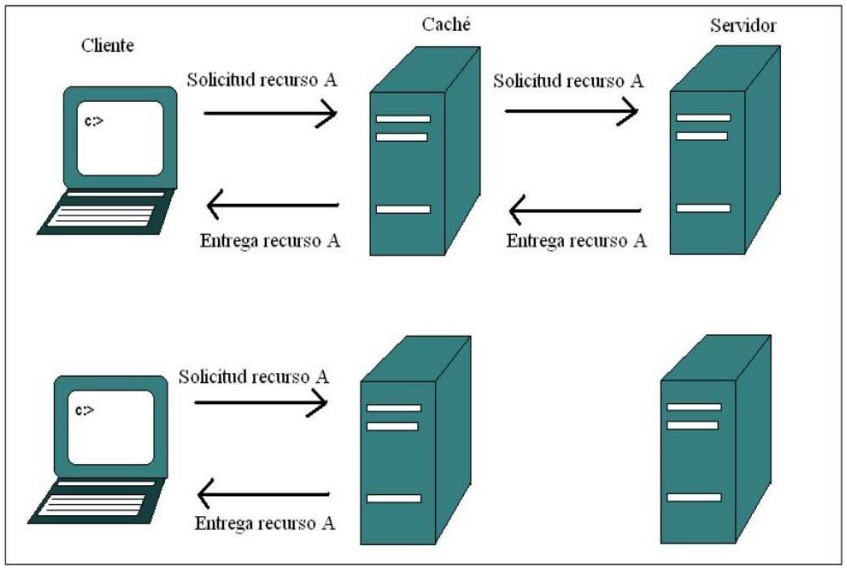
Cliente/Servidor



Sin estado



Caché

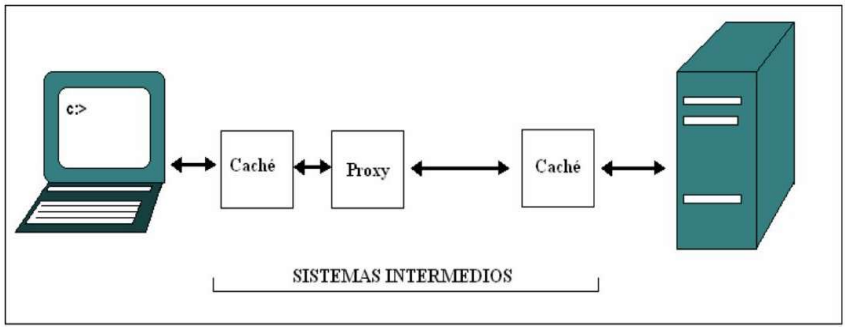


Interfaz uniforme

Para este estándar se establecen algunas restricciones:

- Identificación de recursos.
- Manipulación de recursos a través de sus representaciones.
- Mensajes auto-descriptivos.
- Hipermedios como el motor del estado de la aplicación.

Sistema de capas



Código bajo demanda

Este ítem es opcional y permite a los clientes descargar el código y ejecutarlo en forma de scripts. Esto le da al cliente la posibilidad de extender la cantidad de funcionalidades de la aplicación.

API REST

- El término API REST significa utilizar una API para acceder a aplicaciones backend, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura REST.

- Para realizar el envío y recepción de datos se utilizan las especificaciones más importantes del protocolo HTTP que son GET, POST, PUT, DELETE.

- Todas las solicitudes cuentan con una URL, el tipo y los datos que se requiere enviar.

Tipos de solicitudes

GET

Se utiliza para obtener información. Se le puede pasar parámetros a través de la misma URL.

POST

Se usa para agregar nuevos datos. Los datos deben enviarse a través del Body de la solicitud.

PUT

Sirve para actualizar datos y éstos deben enviarse en el body de la solicitud.

DELETE

Nos indica que queremos eliminar algo. Los datos pueden enviarse en la URL o en el body de la solicitud.

Consumir APIs

- Las APIs se encargan de manejar nuestras peticiones desde el browser y devolvernos los datos que les solicitamos para que podamos mostrarlos en la web.
- Para consultar y enviar datos utilizaremos la página JSONPlaceholder , que nos permite extraer y enviar datos, además de utilizarlo como API REST para hacer pruebas o prototipos.

Resources

JSONPlaceholder comes with a set of 6 common resources:

/posts	100 posts
/comments	500 comments
/albums	100 albums
/photos	5000 photos
/todos	200 todos
/users	10 users

Note: resources have relations. For example: **posts** have many **comments**, **albums** have many **photos**, ... see below for routes examples.

Ejercicio guiado: Solicitud a una API

Conectarse a la sección de usuarios (/users) de la JSONPlaceholder y solicitar los datos que contenga esa dirección, mediante el método fetch y empleando promesas, traer la respuesta y mostrarla en la consola del navegador web.

Ejercicio propuesto

Utilizando el método fetch y promesas, muestra en la consola del navegador web la información suministrada por la API

Solicitud de múltiples APIs en secuencia

- Existen ocasiones que para solicitar datos a una API, dependemos de algunos valores que nos retornan otras APIs y para esto necesitamos solicitar datos a múltiples APIs en forma secuencial.

- Implementar el método fetch pero esta vez trabajado de forma asincrónica con Async / Await.

Ejercicio guiado: Múltiples solicitudes

Se solicita mostrar la información de cada usuario que realizó o escribió comentarios en un Blog. Por ende, se debe traer de la siguiente API una lista de comentarios (/posts) para luego utilizar de la lista de posts el ID del usuario que realizó el comentario, el atributo donde se encuentra el ID lleva por nombre "**userId**". Después de extraer este dato en particular, se debe solicitar toda la información de cada uno de los usuarios que escribieron o realizaron uno de los 100 posts que entrega la API de JSONPlaceholder.

Ejercicio propuesto

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).

Ejercicio guiado: Publicaciones de un usuario

Se solicita mostrar la información de un usuario en específico que haya realizado publicaciones en el blog. Por ende, debe traer primeramente de una API la lista de publicaciones (/posts?userId=id) realizados por ese usuario en particular, así como la información exclusiva del usuario que entrega la API de JSONPlaceholder. Por consiguiente, en este ejercicio vamos a realizar una llamada múltiple, a los datos y post del usuario con id “1”.

Ejercicio propuesto

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.

Ejercicio guiado: Publicaciones de un usuario, utilizando la información

Partiendo del ejercicio “Publicaciones de un usuario”, donde se hace la solicitud en paralelo a la API con dos promesas, para luego mostrar en la consola del navegador los datos del usuario y los post realizados por el mismo, en este ejercicio debemos mostrar esos datos de forma ordenada e individual.

Ejercicio propuesto

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

Manejo de errores

{desafío}
latam_

Competencias

- Detallar los distintos tipos de excepciones que ocurren al ejecutar un código en JavaScript para ubicar errores rápidamente.
- Implementar excepciones con Throw / Reject para generar errores sin restricción del tipo de excepción.
- Capturar errores con catch para mostrar mensajes personalizados de errores

Excepciones

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

- Para el caso de una promesa, cuando ésta 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

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.

```
funcction 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.

Ejercicio propuesto

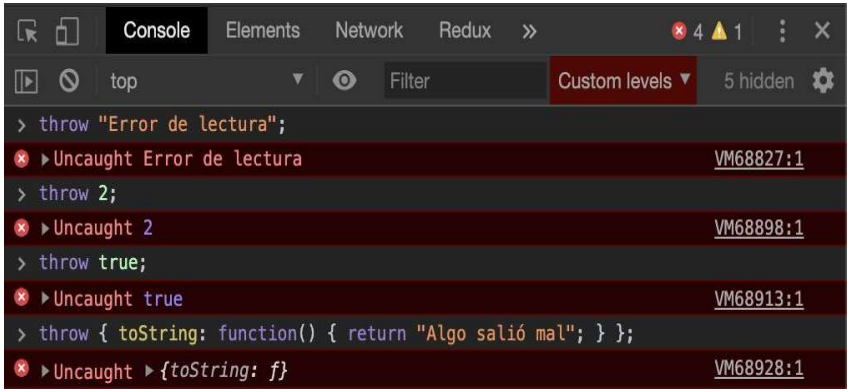
Del siguiente código y sin ejecutar en el navegador, ¿Cuál crees que sería el error que se mostrará en la consola? Verifica ahora tu respuesta ejecutando el código directamente en la consola de navegador web.

```
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));
```

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
```



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.

Ejercicio propuesto

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”.

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 ésta, 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

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

{desafío} latam_

Ejercicio propuesto

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.

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.

Ejercicio guiado: 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.

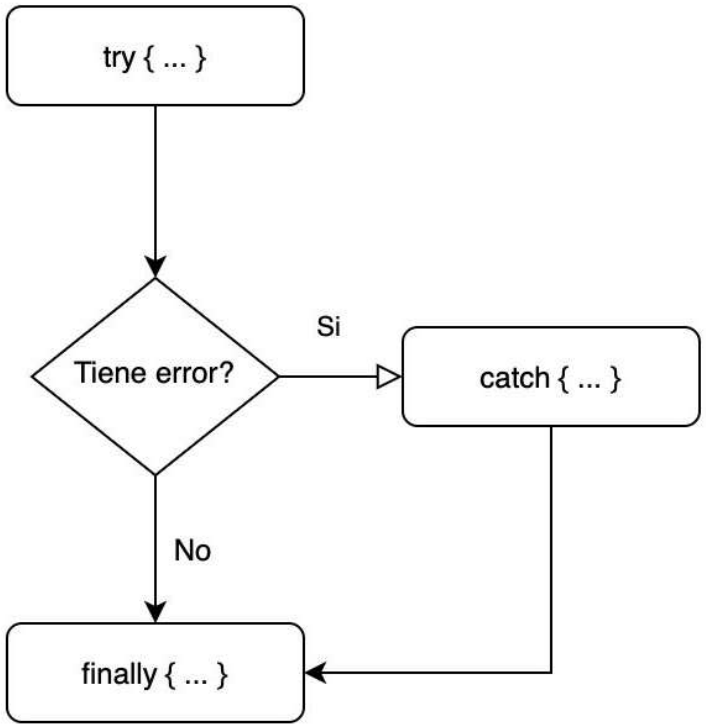


Diagrama de flujo de try, catch, finally

Ejercicio propuesto

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."
  }]
}
```

Buenas prácticas

- 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

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.

{desafío} latam_

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.

Ejercicio propuesto

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.

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com