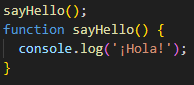


***JavaScript***

Sí, en JavaScript es posible definir una función después de llamarla. Esto se debe a que JavaScript utiliza un concepto llamado "hoisting" (elevación) que implica que las declaraciones de funciones y variables se mueven automáticamente al principio del ámbito en el que se encuentran, antes de que se ejecute el código. Ejemplo;



Identar= Ordenar el codigo.

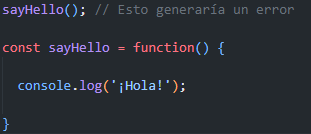
En este ejemplo, la función sayHello() se llama antes de su declaración. Sin embargo, debido al hoisting, la declaración de la función se mueve al principio del ámbito y la función se puede ejecutar correctamente.

Es importante tener en cuenta que el hoisting solo ocurre con las declaraciones de funciones (usando la palabra clave function) y a las variables declaradas con var.

No con las expresiones de funciones (como las funciones flecha, let, const).

Las expresiones de funciones deben declararse antes de su uso.

Ejemplo que muestra cómo una expresión de función no se puede invocar antes de su declaración:



Los errores son nkeos

En este caso, intentar llamar a la expresión de función sayHello() antes de su declaración generaría un error, ya que las expresiones de funciones no se ven afectadas por el hoisting.

En resumen, en JavaScript puedes llamar a una función antes de su definición debido al hoisting. Sin embargo, este comportamiento solo se aplica a las declaraciones de funciones y no a las expresiones de funciones.

Los eventos del clic no podremos darle un valor a funciones asociadas button.addEventListener("click",()=>{

alert("pedro");

})

La forma correcta de hacerlo es:

button.addEventListener("click",saludar);

Funcion= saludar(){

alert("pedro");

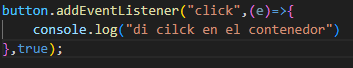
};

El único parámetro que podemos pasar a los eventos el es “event”

PAra acceder al metodo de un objeto utilizamos la funcion (.this)

EventBuubling vienen por defecto

EventCapturing son eventos específicos que les damos estos tienen mayor prioridad.

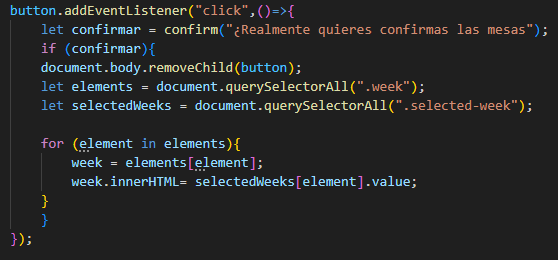


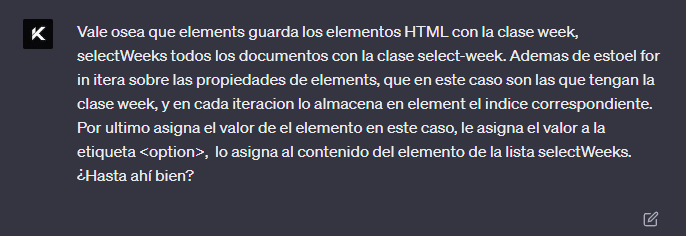
Si le damos true a un evento le dará mayor prioridad, aunque no la tenga

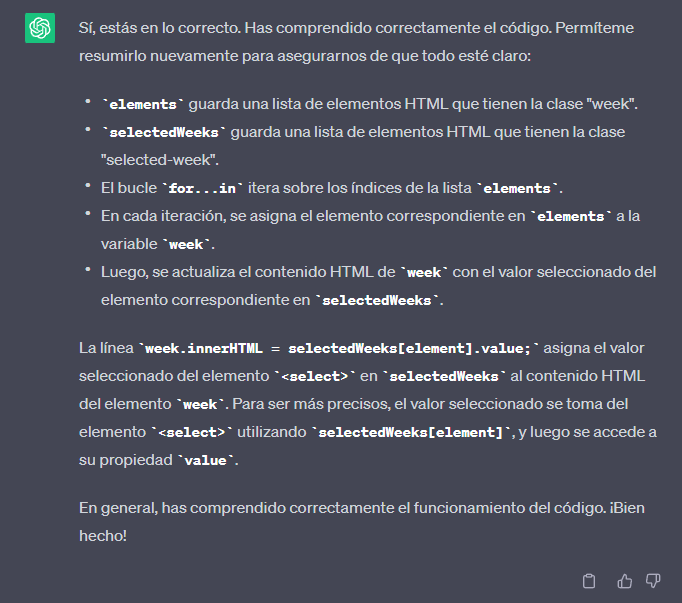
e.stopPropagation() para frenar la propagación

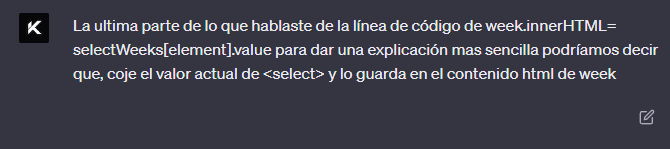
Ejemplo de explicación del código:

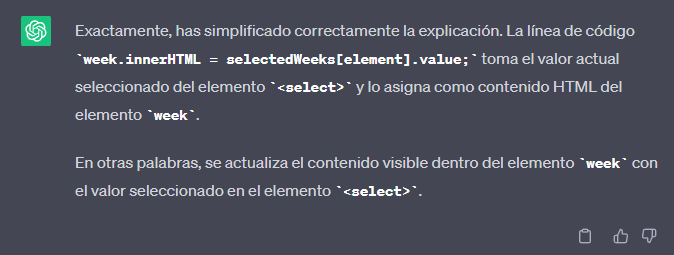






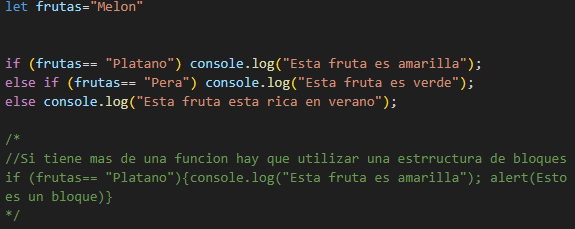




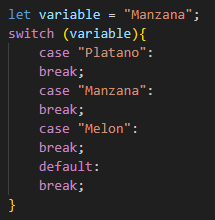


Las sentencias de control de flujo

Son; if, else, else if, while, switch. Estas sentencias funcionan en bloques.



**Estructura switch**



default= en el caso de que no sea ninguna de las anteriores

Las sentencias de manejo de excepciones

***- Excepciones y tipos***

* Objet
* Function
* Bolean
* Synbol
* Error
* EvalError
* InternqalError
* RangeError
* ReferenceError
* SyntaxError
* TypeError
* URIEError

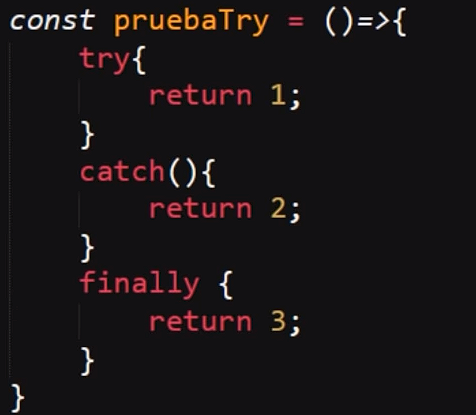
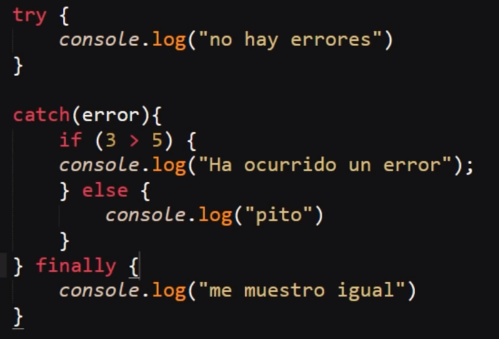
Try... Catch

El try no puede ejecutarse si no hay un catch

Typeof nos dice de que tipo es (boleano, numero, string, obejto)...

Sentencias de bloques

Try catch, con una condición dentro es condicional, si no incondicional.

finaly, pase lo que pase se muestra.

throw lanza errores throw, puede almacenar bloques {}, o arrays [];.

En chatch se suele poner e, o error por motivos de legibilidad.

En catch guarda sus parámetros en función de lo que le demos a throw, normalmente en try.

***Documentación de JavaScript***

<https://www.w3schools.com/>

<https://developer.mozilla.org/es/docs/Web/JavaScript>

***Callbak***

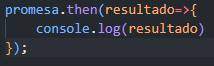
Un callbak es una funcion dentro de otra fucnion

***Promise***

Las promesas son datos encapsulado a los que no se pueden acceder

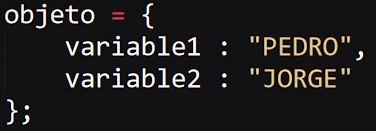
resolve o rejet.

Para acceder a las promesas debemos acceder a él con then. Es un metodo que recibe un callback.

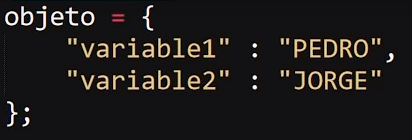


***Petcionones HTTP***

JSON (javascript objet notation)



Esto es una array asociativa, si le añadimos comillas a las propiedades se transforma en un JSON.



Cuando decimos que esta serializado es una cadena de texto., eso seria un JSON serializado



**stringfy= serializa el código para poder enviarlo a el servidor.**

**.parse= lo desrealiza para poder usarlo. Si no lo hacemos he intentamos acceder a el objeto nos dará undefined ya que es un string y con parse lo transformamos en objeto.**

**Ejemplo JSON.parse**



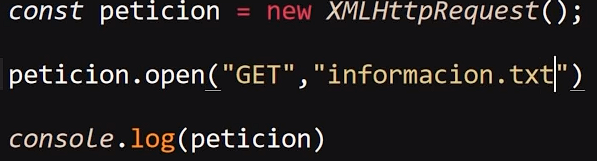
AYAX (asincronic java)

***HTML Request***

.open es un método para abrir una petición con “GET” o “POST” y “URL” o archivo.

.send para enviar

Para acceder a un dato el código de respuesta debe ser 3 o 4 y el status en 200.



Readystatechange tiene 4 estados. (Solo nos indica si lo recibió correctamente y si nos envió una respuesta, pero esa respuesta perfectamente podría ser un error, no nos indica que aya encontrado el recurso)

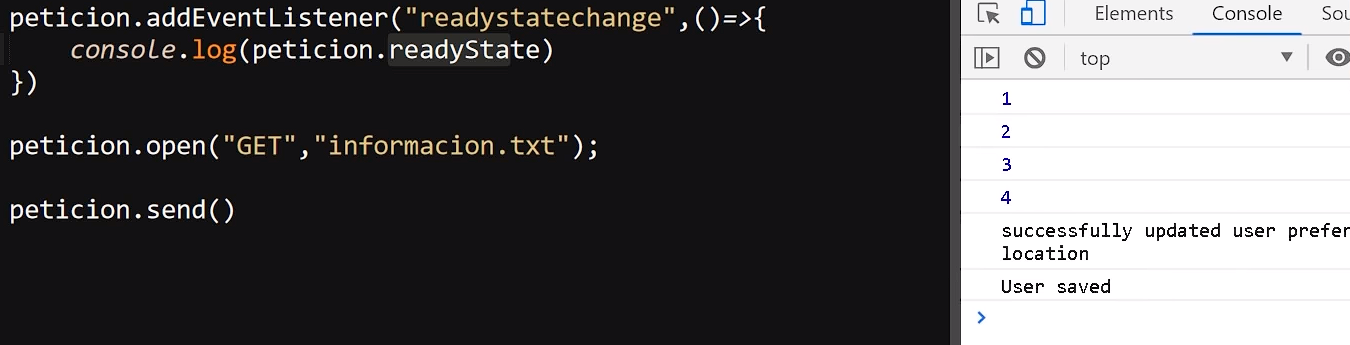
1 Se creo correctamente.

2 Solicitud se recibió correctamente.

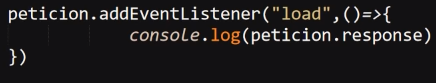
3 Procesando petición

4 Todo listo nos dio ya una respuesta y ya se puede utilizar.

El estado lo vemos con el método .status si todo salió correctamente el status es de 200.



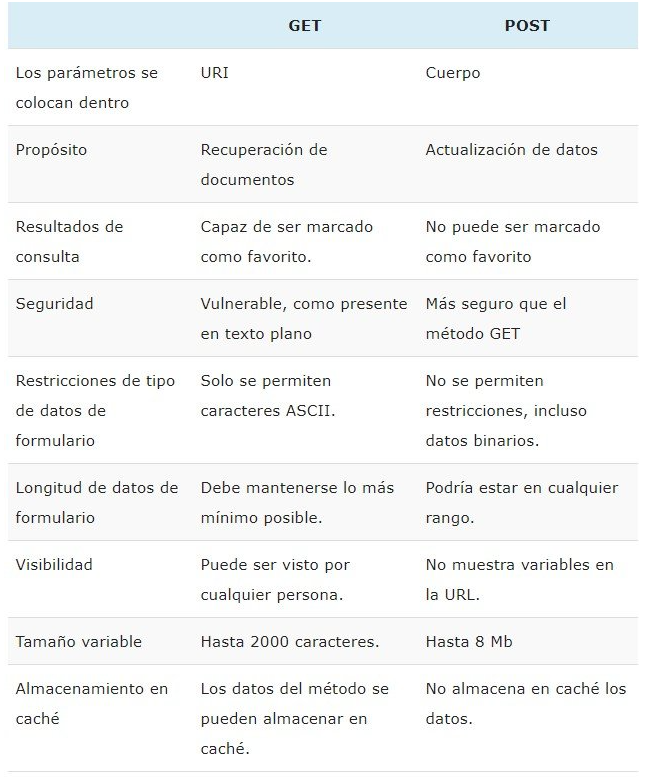
Para validar la información ya no se utiliza readystatechange o status actualmente se utiliza el parametro load.



AYAX no es soportando por todos los exploradores

Peticiones Post:

Devuelve el .status 201

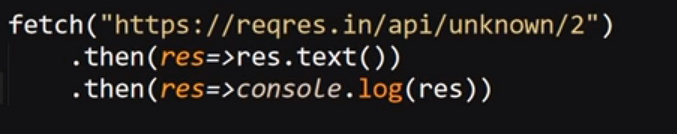


***Fetch***

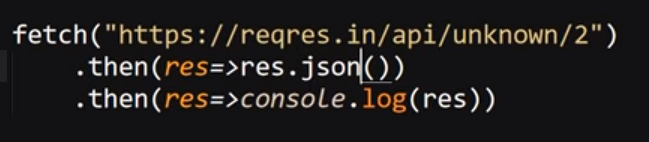
Fetch a sustituido a ayax en los últimos tiempos, nos devuelve una promesa encapsulada.

Json.stringyfy(res) = res.text()

Pero con fetch utilizamos res.txt() ya que no podremos serializarlo o de serializarlo ya que e es una promesa.



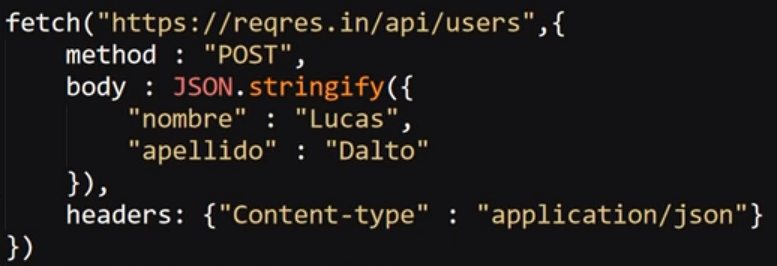
Mostraria el string



Mostraría el texto encapsulado en promesas en un objeto JSON

Jsonstringify(res), Esto deserializa o transforma a texto el resultado. Utiliza una estructuira json.

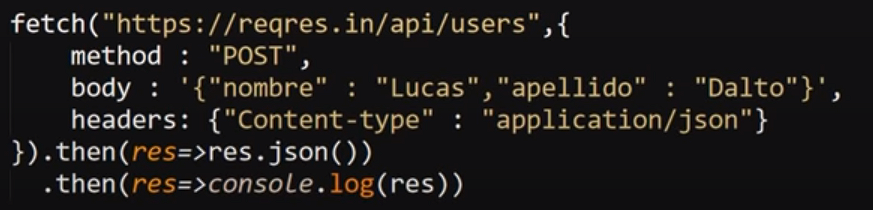
JSON, Esto serializa o transforma el texto en un objeto JSON.

***Peticiones POST***

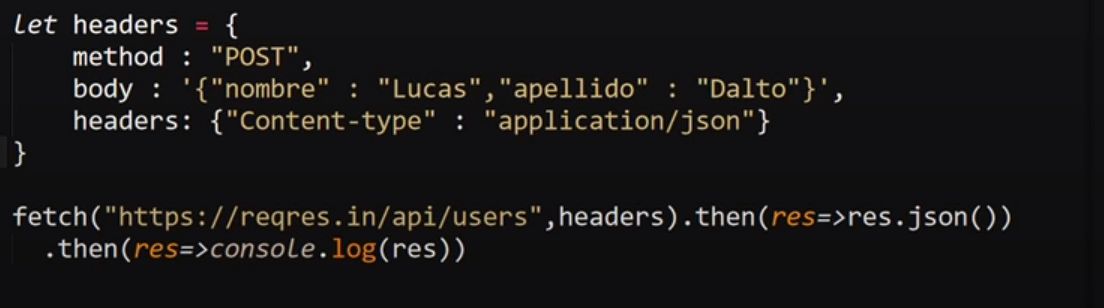
En esta imagen falta añadir las desencapsulaciones con .then

Deberemos crear objetos con fetch(“URL”,{ method :, body : JSON.stringify ({ parametro: , parametro:}), headers: { “Tipo de contenido”: “application/json”}});

Podemos cambiar JSON.stringify por backtics en el caso de que sea corto ‘’.

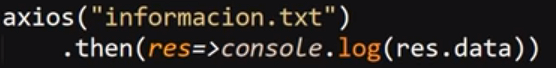


Para hacerlo mas eficiente podremos utilizar una variable que ocupe los objetos y luego llmar a fetch.



Dependiendo del metodo que elijamosdebemops elegir un headrer y valor de la header distinto distinto.

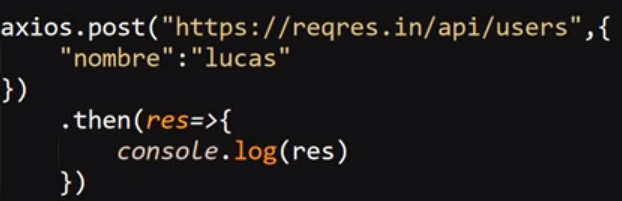
Axios



Estas dos líneas de código hacen lo mismo que las peticiones anteriores de post, fetch, ayax, y XMLHTTPRequest anteriores.

Con axios.get tenernos lo mismo, pero con el método get = axios, get viene por defecto

Podremos utilizaraxios. Post para utilizar el metdo post, el segundo parametro es lo que vamosa a enviar, en este cado un objeto nombre con la propiedad lucas.



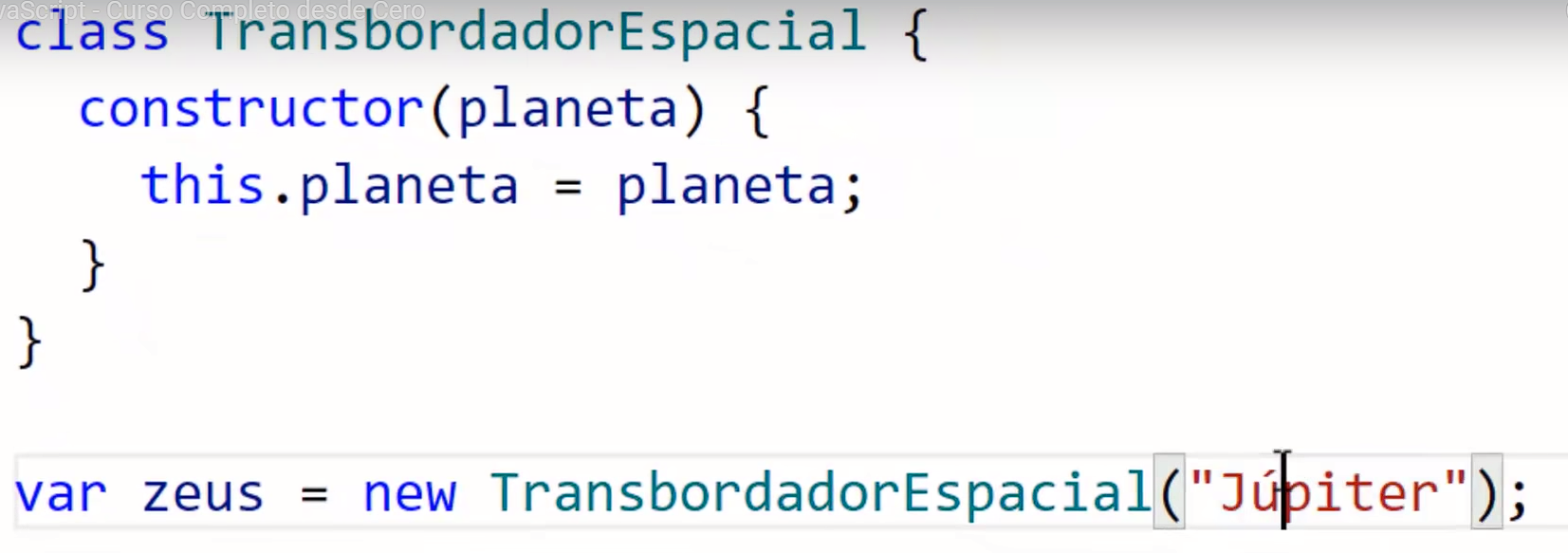
Implementación.

SI el valor de una variable es una función la llamamos método.

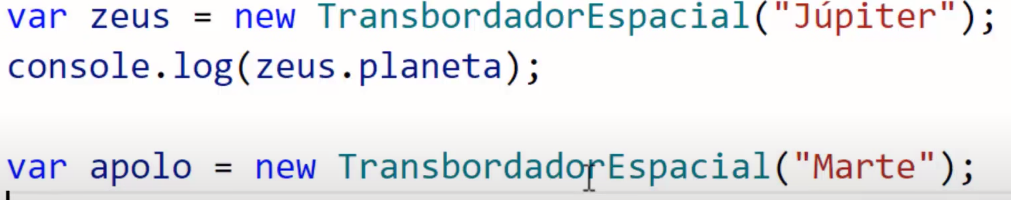
Este ejemplo sirve para crear clases con diferentes paramstros

Estamos creando una clase que contiene un constructor que se llama automáticamente al llamar a la clase, en el constructor podremos asignar un parámetro, después le decimos que ese parámetro se debe almacenar en un objeto llamado planeta.

Una vez hecho esto creamos una variable y le asignamos el valor de una nueva clase que constara del parámetro planeta y le podremos asignar el valor para que cree la clase



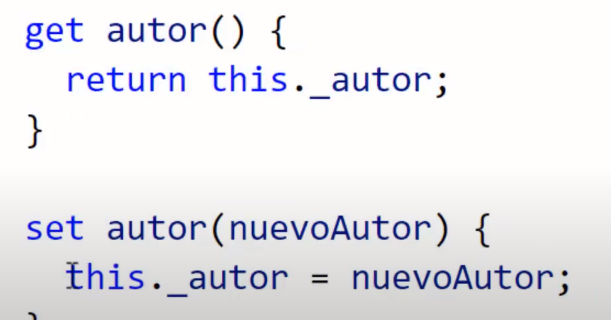
De esa manera podremos crear nuevas clases



\_ se utiliza como señar para decirles a otros desabolladores que no se debería cambiara las propiedades de ese objeto.



TEnemos getter y setter, geter para acceder y ver la propiedad y setter para actualizarla a una nueva y ver la actual propiedad usando condicionales.



? Operador ternario

Funciona como in if else