1. To assign events to HTML elements we can use event attribute <button onClick= “func()”>
2. HTML DOM allows to assign events to HTML elements using java script

document.getElementById(“demo”).onClick= displayDay();

onClick is actually used in the HTML tag like

<button onClick= “displayDay()”>click</button>

1. onmouseover, onmousedown, onmouseup, onclick

**From here on wards we are using exp1.js and exp1.html:**

Normal programs are synchronous. If there is a task which is gonna take time, it blocks the rest of the code. Hence its known as blocking code.

Java Script by its nature is a synchronous language. It can execute only one statement at a time from top to bottom. It is a single threaded language. Synchronous code is blocking code.

Asynchronous is start something now and finish it later.

Say

Statement 1,2,3,4,5

2 is asynchronous.

Javascript takes 2 out of the scope of the thread and executes it in another part of the browser. It then continues through 3,4,5. Once the rest of the functions are executed, it is allowed to call the callback function and finish the original function 2. Asynchronous is non-blocking.

API Endpoints: urls a particular api or server exposes to us to access resources.

Asynchronous use call back functions. These are the functions which are executed as soon as the asynchronous function comes back with result. Call back functions tell the asynchronous functions what to do after they are done processing. While processing the rest of the code is executed.

**XMLHttpRequest**

https://jsonplaceholder.typicode.com/

this website helps you experiment with http requests. Scroll down. You will see an example. There is a url which ends with /*todos/*1. When you call it, it returns JSON data.

So we will keep using that url in our code in XMLHttpRequest();

This is the url:

[https://jsonplaceholder.typicode.com/todos/](https://jsonplaceholder.typicode.com/)

You can browse this URL in search bar and see the data.

const request=new XMLHttpRequest(); //this returns a request object

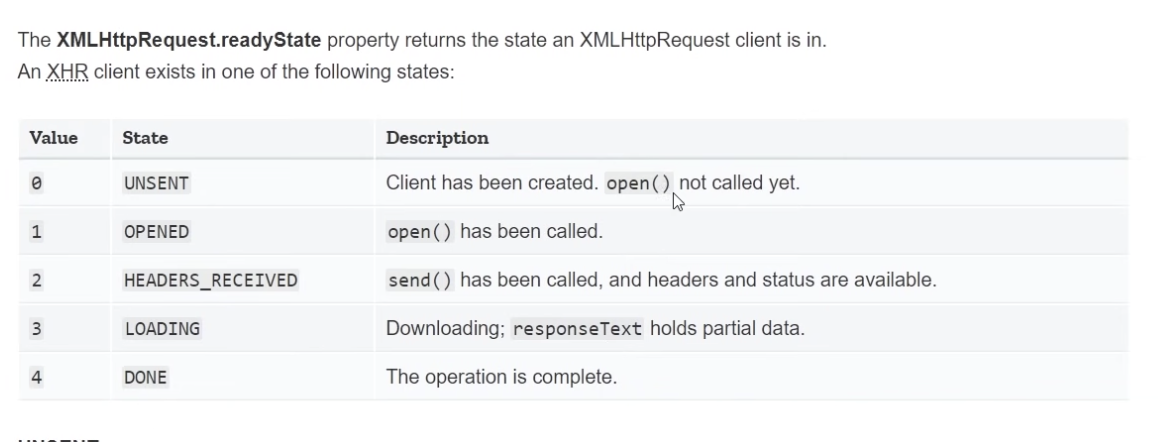
request.addEventListener(‘event\_name’,callback\_function\_if\_event\_happens);

request.open(‘GET/POST’, ‘url to the resource’);

request.send();

readystatechange is an event which takes place whenever the state of the request changes

There are 4 different states a request goes through:



const request=new XMLHttpRequest();

request.addEventListener(‘readystatechange’,()=>{

if(readState===4)

console.log(request.responseTex);

}

reqeust.open(‘type\_of\_the\_request’, ‘api end point to access resoures’);

request.send();

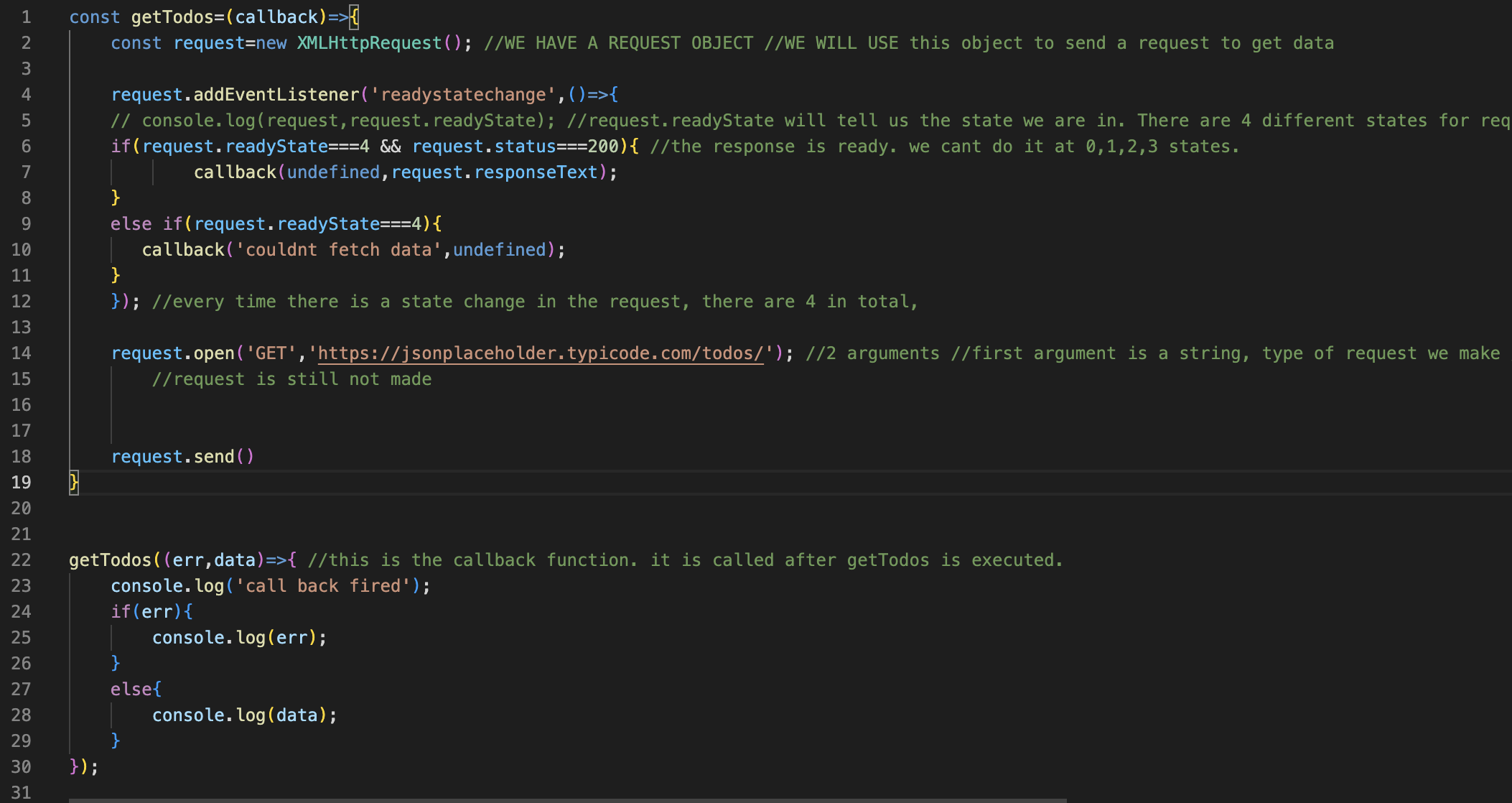
when readyState===4 it means we received the response. But that alone is not enough because there might be an error and the request state or the state of the request might still be 4. All though the request reaches state 4, it comes back with error code. So we need to check the status additionally.

200 status: everything is fine and the response came back with data.

300: for redirections

404: cant find the resource. client errors.

500: when there is server error.



In the above code getTodos is a function.

That function takes one argument=callback;

In that function we send a http requeest and check for readstate and status codes.

In those if else blocks we call the callback function which is argument of getTodos.

The call back function can be uniquely defined and implemented each time we call getTodos.

As shown above, when we call the getTodos function we define the callback function with 2 arguments (ERROR,DATA).

if(err) statement is true if error is a string, if(err) statement is false if error is undefined.

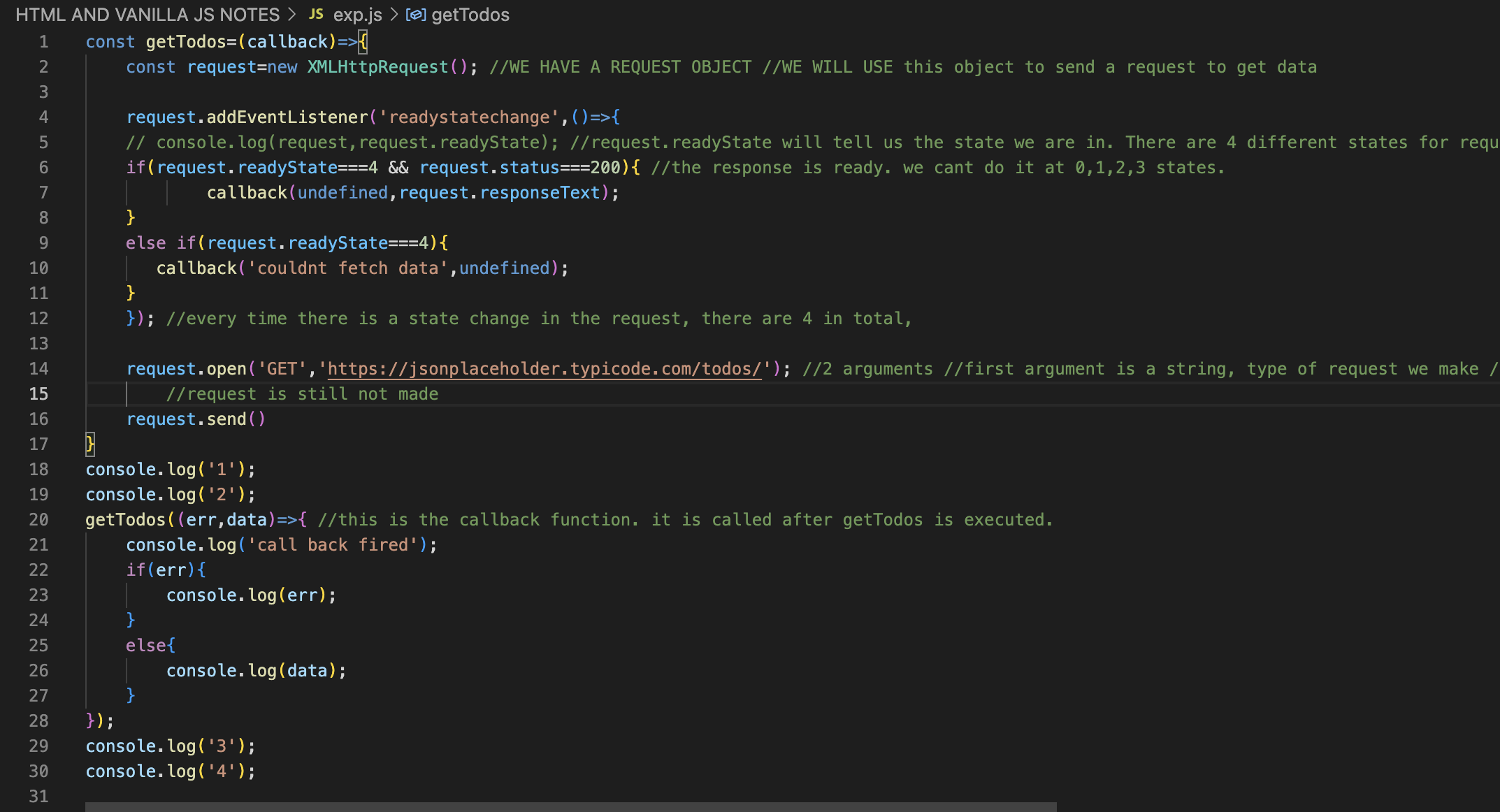
Error is defined when request.status!=200. For this to happen change the url.

https://jsonplaceholder.typicode.com/todosssssxyz/

Now there is no resource in the above url. It is a client side error. So you get status 404. Hence the else if statement is executed. In that the error is ‘couldnt fetch data’

This is more reusable.

So lets now experiment if it works asnychrnonously by putting the console.log() functions before and after the calling of getTodos function.



In the above code the output is:

1

2

3

4

callback fired.

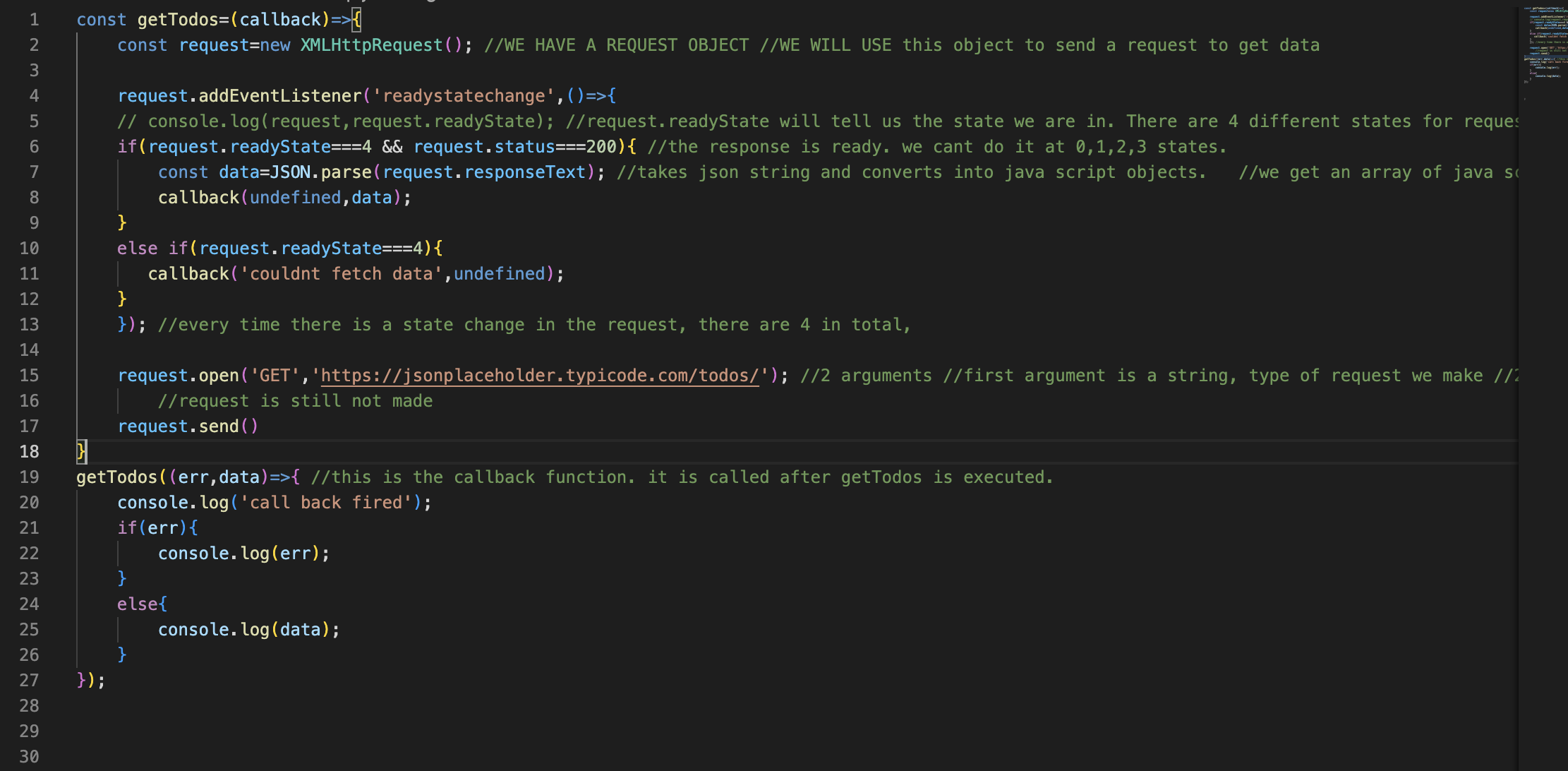
all the data.

**JSON DATA**

The data looks like java script objects. JSON is strings that look like java script objects. We are going to take the json strings and convert it into java script objects. There is an object built into the java script language which we can use.

We will convert the request.responseText into Java script object.

Now lets edit the previous program in such a manner that it shows the json data by converting them into java script array of objects.



JSON.parse() will take the json strings and convert into java script objects. JSON is just in the java script object notation. They arent java script objects. JSON is used in communication. And network communication takes place in text format. So JSON is text. purely strings.

We can also create our own todos.json file.

[

{“key”: “value”},

{“key”: “value”},

{“key”:1234}

]

In JSON, all our keys are in double quotes, string as a value are also in double quotes. If we want to use numbers as values, there is no need to use double quotes.

Keep it on a http server.

$npm install -g http-server //install server

$http-server //launch server

Now exp1.html which calls the exp1.js script are in the same folder in which the http server is launched.The todos.json file is in the same directory. Replace the request.open(); with that url.

request.open('GET','todos.json');

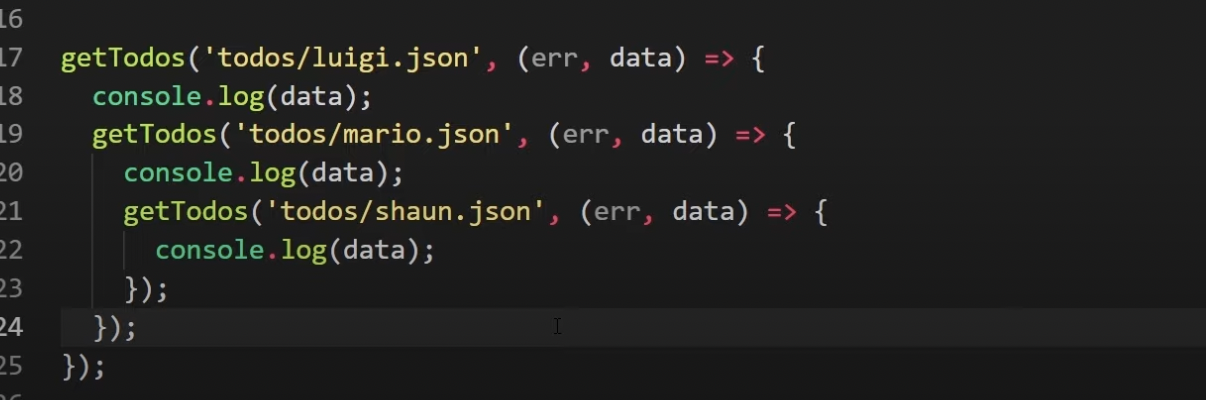
You will get the

To summarize:

When accessing a server, it sends json data. We parse the json data in javascript and use it.

**FROM HERE ONWARDS WE WILL USE EXP2.JS and EXP2.HTML AND TODOS FOLDER WHICH WILL HAVE JSON FILES**

Now what if we want to access many json falls one after the other?



Our code starts looking like the above. We are nesting the requests inside the callback function. This is known as callback hell.

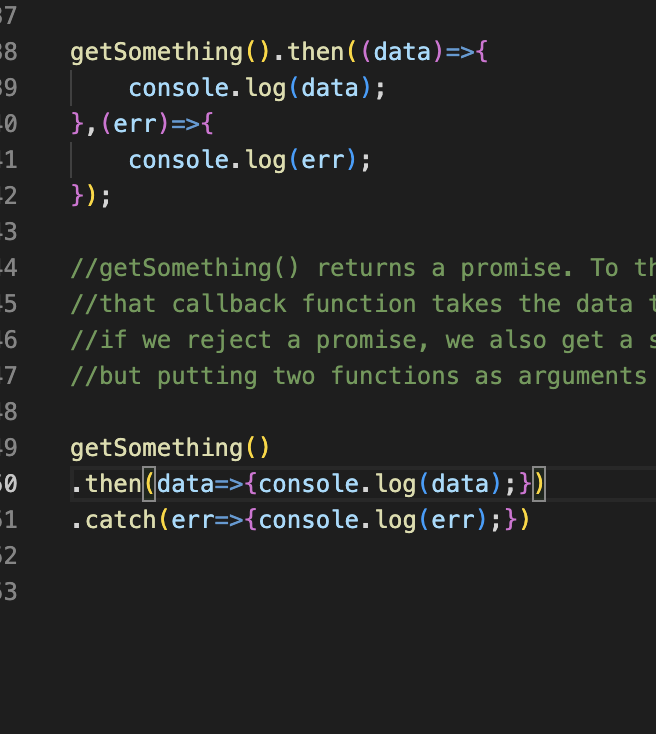
So we use promises.

Now use exp3.js and exp3.html to learn basics of promise

**PROMISE**

A promise is something which takes some time to do.

2 outcomes of promise: 1)resolved 2)rejected



//getSomething() returns a promise. To that we can attach .then method. When we resolve in a promise, it fires a callback function in the then method.

//that callback function takes the data that is passed from the resolve function

//if we reject a promise, we also get a second call back function as second method in the then method.So we will use the catch method insteaD

promise().then(callback\_function\_called\_when\_resolved).catch(callback\_function\_called\_in\_error);

From here onwards use exp4.html and exp4.js

Here we took the exp2.js and modified it using the promises.

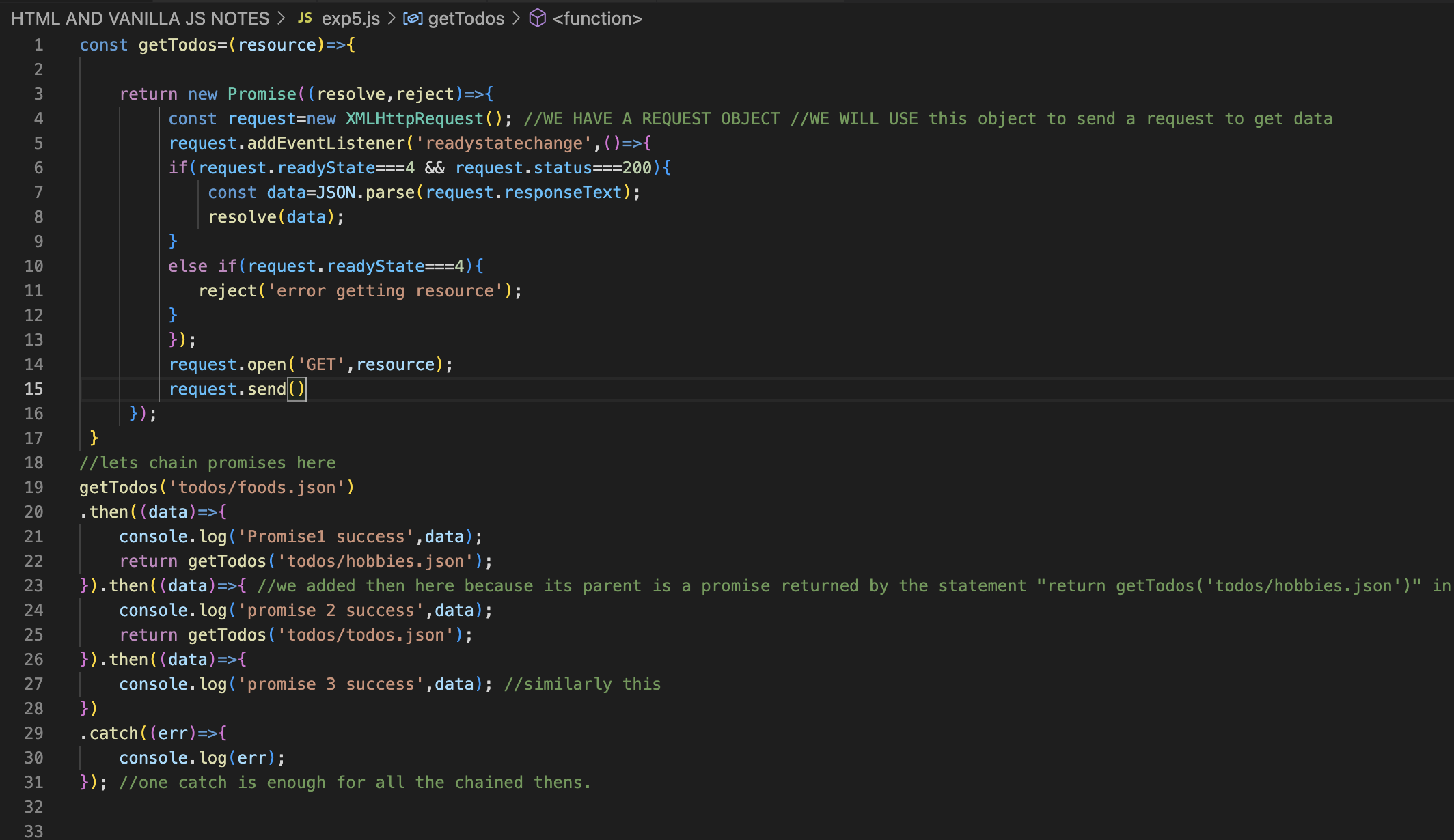
getTodos function will return a promise.

This promise will call back resolve or reject.

If it calls resolve(data) which sends the data, the callback function within the “then” of getTodos().then() will get executed. This callback function will receive the data as the parameter.

else the callback function within the catch of of getTodos.then().catch() will get executed and this callback function will take the error as the parameter.

lets use exp5.html and epx5.js to chain promises.



**FETCH API**

Till now we used XMLHttpRequest. We will use exp6.js and exp6.html.

fetch() will take one argument which is the address of the resource we try to access.

1)fetch() the data

2) take the response and return response.JSON(). That returns a promise

3)So attach .then() again.

**ASYNC AND AWAIT**

Use exp8.html and exp8.js (exp7.js and exp7.html are of no use)

When we start to chain more promises, it starts to look messy.

Async and await, will help us to section off all of them into a function- an asynchronous function.

**THROWING ERRORS**