JAVA SCRIPT PART 3:

Synchronous programming lang vs asynchronous programming lang :

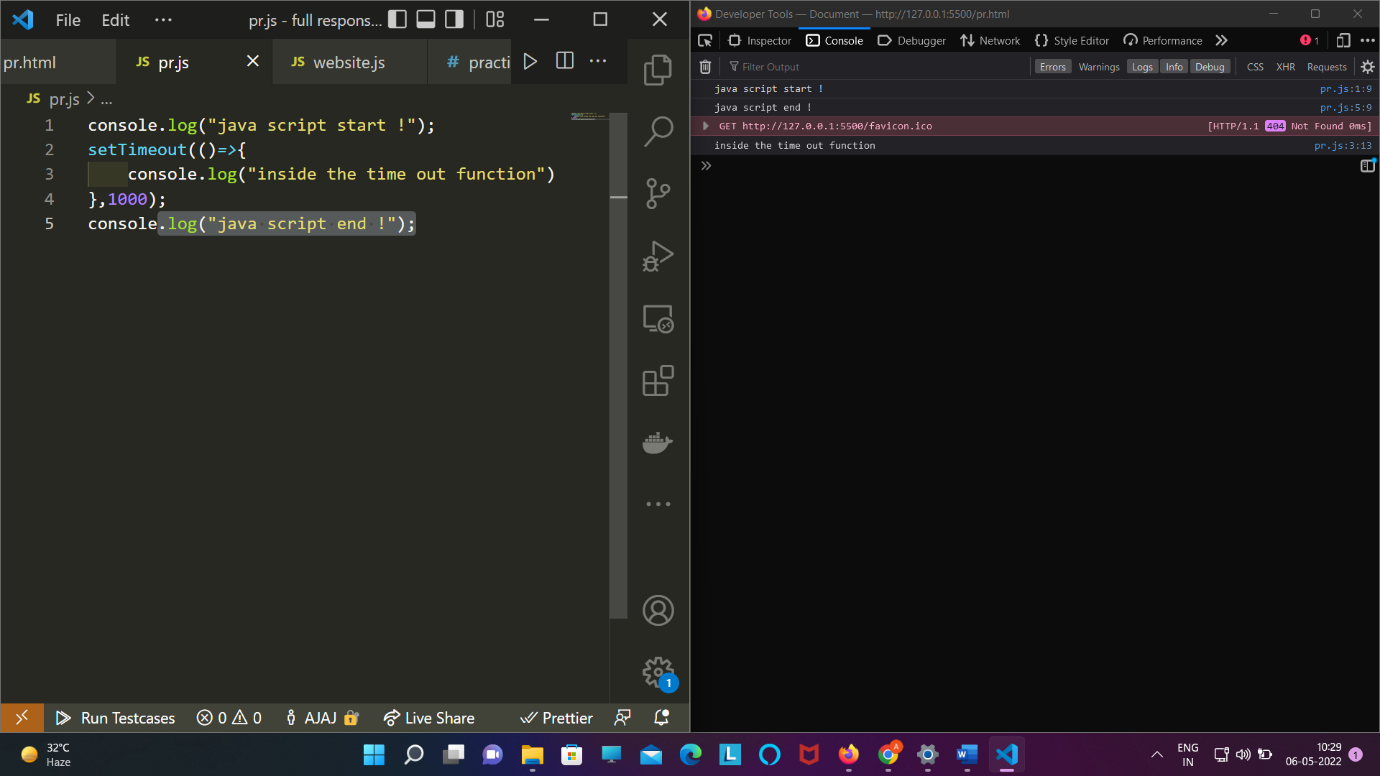
Note: JavaScript is single thread asynchronous programming lang .

Set timeout function : it will going to take function and time in milli second as input and specify the minimum time after which that function will going to run .

Note : set time out is browser functionality it will going to run by browser

ii ) function inside the set time out execute only after the whole js code execute and it will be executed by browser not by js .

iii) function inside the set time out will after its compliation of its time it will going to store in callback queue .



iv) 1000msecond = 1second

v ) if we try to store set time out in a variable the it will going to store as 1 ,2 ,3 …….

Eg : const var1=settimeout(function(),1000);

Console.log(var1) // output 1 ;

Clear time out : this will going to take input as a variable which we store set time out and it forced the set time out not to run;

console.log("java script start !");

*const* id=setTimeout(()*=>*{

    console.log("inside the time out function")

},0);

console.log("java script end !");

clearTimeout(id);

SetInterval :

console.log("start js.....");

setInterval(()*=>*{

    console.log(Math.random());

},1000);

console.log("end js");

it will going to keep doing the same work for infinite no of times .

Clearinterval :

console.log("start js.....");

*const* id=setInterval(()*=>*{

    console.log(Math.random());

},1000);

console.log("end js");

clearInterval(id);

it will going to stop the setinterval to work .

Application of set time out and function callback :

We want to change text content and color at ever second :

Html :

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <script src="pr.js" defer></script>

</head>

<body>

    <div>

        <h1 class="h1">heading1</h1>

        <h1 class="h2">heading2</h1>

        <h1 class="h3">heading3</h1>

        <h1 class="h4">heading4</h1>

        <h1 class="h5">heading5</h1>

        <h1 class="h6">heading6</h1>

    </div>

</body>

</html>

Js : (Call back hell )

setTimeout(() *=>* {

*const* h1 = document.querySelector(".h1");

  h1.textContent = "one";

  h1.style.color = "violet";

  setTimeout(() *=>* {

*const* h2 = document.querySelector(".h2");

    h2.textContent = "two";

    h2.style.color = "purple";

    setTimeout(() *=>* {

*const* h3 = document.querySelector(".h3");

      h3.textContent = "three";

      h3.style.color = "red";

      setTimeout(() *=>* {

*const* h4 = document.querySelector(".h4");

        h4.textContent = "four";

        h4.style.color = "pink";

        setTimeout(() *=>* {

*const* h5 = document.querySelector(".h5");

          h5.textContent = "five";

          h5.style.color = "green";

          setTimeout(()*=>*{

*const* h6=document.querySelector(".h6");

                h6.textContent="six";

                h6.style.color="blue";

            },1000)

        }, 1000);

      }, 1000);

    }, 1000);

  }, 1000);

}, 1000);

🡺 // callbacks , callback hell, pyramid of doom

// asynchronous programming

*const* heading1 = document.querySelector(".heading1");

*const* heading2 = document.querySelector(".heading2");

*const* heading3 = document.querySelector(".heading3");

*const* heading4 = document.querySelector(".heading4");

*const* heading5 = document.querySelector(".heading5");

*const* heading6 = document.querySelector(".heading6");

*const* heading7 = document.querySelector(".heading7");

*const* heading8 = document.querySelector(".heading8");

*const* heading9 = document.querySelector(".heading9");

*const* heading10 = document.querySelector(".heading10");

// Text       Delay   Color

// one        1s      Violet

// two        2s      purple

// three      2s      red

// four       1s      Pink

// five       2s      green

// six        3s      blue

// seven      1s      brown

// callback hell

// setTimeout(()=>{

//   heading1.textContent = "one";

//   heading1.style.color = "violet";

//   setTimeout(()=>{

//     heading2.textContent = "two";

//     heading2.style.color = "purple";

//     setTimeout(()=>{

//       heading3.textContent = "three";

//       heading3.style.color = "red";

//       setTimeout(()=>{

//         heading4.textContent = "four";

//         heading4.style.color = "pink";

//         setTimeout(()=>{

//           heading5.textContent = "five";

//           heading5.style.color = "green";

//         },2000)

//       },1000)

//     },2000)

//   },2000)

// },1000)

*function* changeText(*element*, *text*, *color*, *time*, *onSuccessCallback*, *onFailureCallback*) {

  setTimeout(()*=>*{

    if(*element*){

*element*.textContent = *text*;

*element*.style.color = color;

      if(*onSuccessCallback*){

        onSuccessCallback();

      }

    }else{

      if(*onFailureCallback*){

        onFailureCallback();

      }

    }

  },*time*)

}

// pyramid of doom

changeText(heading1, "one","violet",1000,()*=>*{

  changeText(heading2, "two","purple",2000,()*=>*{

    changeText(heading3, "three","red",1000,()*=>*{

      changeText(heading4, "four","pink",1000,()*=>*{

        changeText(heading5, "five","green",2000,()*=>*{

          changeText(heading6, "six","blue",1000,()*=>*{

            changeText(heading7, "seven","brown",1000,()*=>*{

              changeText(heading8, "eight","cyan",1000,()*=>*{

                changeText(heading9, "nine","#cda562",1000,()*=>*{

                  changeText(heading10, "ten","dca652",1000,()*=>*{

                  },()*=>*{console.log("Heading10 does not exist")})

                },()*=>*{console.log("Heading9 does not exist")})

              },()*=>*{console.log("Heading8 does not exist")})

            },()*=>*{console.log("Heading7 does not exist")})

          },()*=>*{console.log("Heading6 does not exist")})

        },()*=>*{console.log("Heading5 does not exist")})

      },()*=>*{console.log("Heading4 does not exist")})

    },()*=>*{console.log("Heading3 does not exist")})

  },()*=>*{console.log("Heading2 does not exist")})

},()*=>*{console.log("Heading1 does not exist")})

🡺 Promise

🡪 promise hold the future result .

🡪 define : taking a call back function

*const* friedRice=new *Promise*((resolve,reject)*=>*{

})

🡪 To use promise we need to do :

i ) taking two input callback function one for resolve and one for reject

friedRice.then((*data*)*=>*{

},(*error*)*=>*{

})

ii) use promise.then(callback\_function\_resolve).catch(callback\_function\_of\_reject)

friedRice.then((*data*)*=>*{

}).catch((*error*)*=>*{

})

iii) like settimeout() ,setinterval() promise also work on control by browser so it will excute only after execution of all stack call .

Example of promise :

Promised to make fried rice .

*const* bucket = ["coffee", "chips", "vegitable", "rice", "salt"];

*const* fried\_rice = new *Promise*((resolve, reject) *=>* {

  if (

    bucket.includes("vegitable") &&

    bucket.includes("salt") &&

    bucket.includes("rice")

  ) {

    resolve("Fried rice"); // it can be anything number,string,object ,arrays, map etx;

  } else {

    reject("could'nt do it");

  }

});

//method1

fried\_rice.then(

  (*data*) *=>* {

    console.log("I am able to fullfile my promise o ", *data*);

  },

  (*error*) *=>* {

    console.log(*error*);

  }

);

//method 2

fried\_rice

  .then((*data*) *=>* {

    console.log("I am able to fullfile my promise o ", *data*);

  })

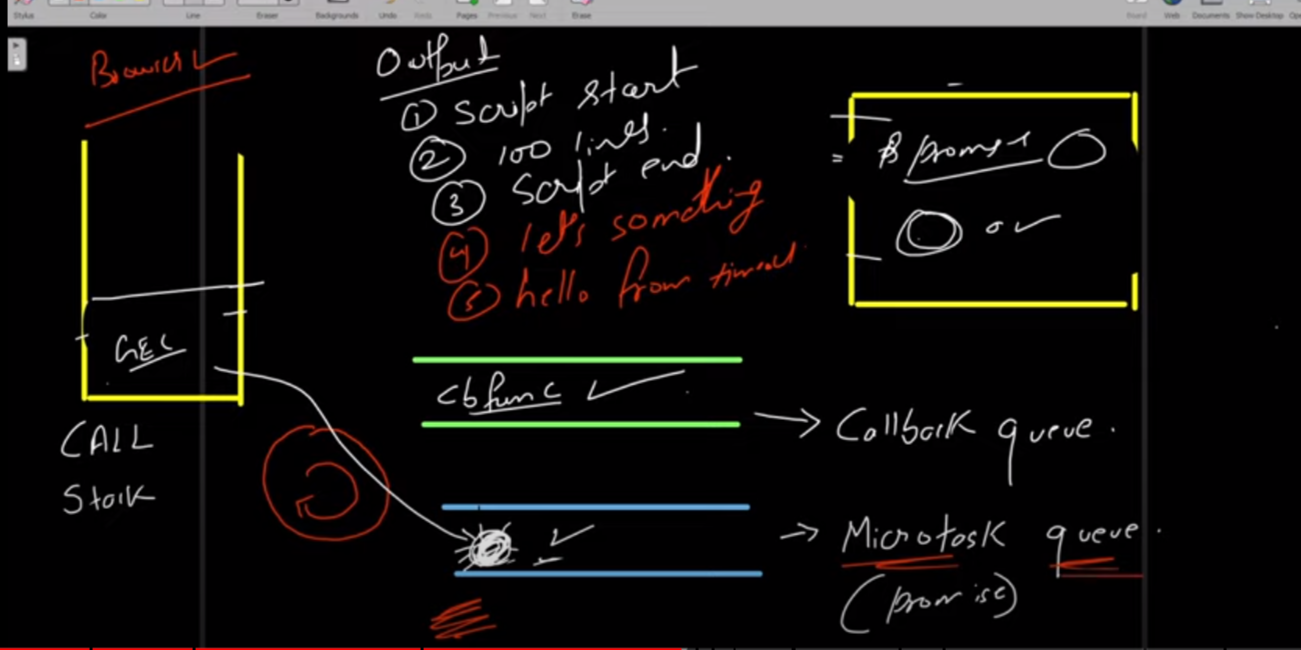
  .catch((*error*) *=>* {

    console.log(*error*);

  });

Note : from web api promise comes in microtask queue .

ii ) priority of microtask queue is more than the call back queue .



ex : running a code containing set timeout function and promise function .

console.log("start of programm ....");

*const* bucket = ["coffee", "chips", "vegitable", "rice", "salt"];

*const* fried\_rice = new *Promise*((resolve, reject) *=>* {

  if (

    bucket.includes("vegitable") &&

    bucket.includes("salt") &&

    bucket.includes("rice")

  ) {

    resolve("Fried rice"); // it can be anything number,string,object ,arrays, map etx;

  } else {

    reject("could'nt do it");

  }

});

//method 2

fried\_rice

  .then((*data*) *=>* {

    console.log("I am able to fullfile my promise o ", *data*);

  })

  .catch((*error*) *=>* {

    console.log(*error*);

  });

  setTimeout(()*=>*{

      console.log("calling settimeout function .")

  },2000);

  console.log("End of programm");

output :

start of programm ....

End of programm

I am able to fullfile my promise o Fried rice

calling settimeout function .

🡺 returning promise:

console.log("start of programm ....");

*function* fried\_rice() {

*const* bucket = ["coffee", "chips", "vegitable", "rice", "salt"];

  return new *Promise*((resolve, reject) *=>* {

    if (

      bucket.includes("vegitable") &&

      bucket.includes("salt") &&

      bucket.includes("rice")

    ) {

      resolve("Fried rice"); // it can be anything number,string,object ,arrays, map etx;

    } else {

      reject("could'nt do it");

    }

  });

}

//method 2

fried\_rice()

  .then((*data*) *=>* {

    console.log("I am able to fullfile my promise o ", *data*);

  })

  .catch((*error*) *=>* {

    console.log(*error*);

  });

Promise.resolve() method :

*const* name=*Promise*.resolve("Ajaj");

name.then((*value*)*=>*{

    console.log(*value*);

})

Note : if we return anything in then function it will be return as promise.resolve(value) :

Eg :

*function* name(){

    return new *Promise*((reslove,reject)*=>*{

        reslove("Ajaj");

    })

}

*const* fullname=name().then((*value*)*=>*{

    console.log(*value*);

    return *value*+=" Alam";  // it is like returning promise.resolve("Ajaj Alam");

})

fullname.then((*value*)*=>*{

    console.log(*value*);

})

Promise chaining :

name()

  .then((*value*) *=>* {

    console.log(*value*);

    return (*value* += " Alam");

  })

  .then((*value*) *=>* {

    console.log(*value*);

    return (*value* += " From Bettiah");

  })

  .then((*value*) *=>* {

    console.log(*value*);

  })

  .then((*value*) *=>* {

    console.log(*value*);

  });

Note : if we not returning any thing but still do Channing it will going to return ‘undefine’ .

Output :

Ajaj

Ajaj Alam

Ajaj Alam From Bettiah

Undefined

🡺 flattening callback hell :

*const* heading1 = document.querySelector(".heading1");

*const* heading2 = document.querySelector(".heading");

*const* heading3 = document.querySelector(".heading3");

*const* heading4 = document.querySelector(".heading4");

*const* heading5 = document.querySelector(".heading5");

*const* heading6 = document.querySelector(".heading6");

*const* heading7 = document.querySelector(".heading7");

*const* heading8 = document.querySelector(".heading8");

*const* heading9 = document.querySelector(".heading9");

*const* heading10 = document.querySelector(".heading10");

*function* changeText(*element*, *text*, *color*, *time*) {

    return new *Promise*((resolve, reject) *=>* {

        setTimeout(()*=>*{

            if(*element*){

*element*.textContent = *text*;

*element*.style.color = *color*;

              resolve();

            }else{

              reject("element not found");

            }

          },*time*)

    })

  }

changeText(heading1, "one", "red", 1000)

  .then(()*=>*changeText(heading2, "two", "purple", 1000))

  .then(()*=>*changeText(heading3, "three", "green", 1000))

  .then(()*=>*changeText(heading4, "four", "orange", 1000))

  .then(()*=>*changeText(heading5, "four", "orange", 1000))

  .then(()*=>*changeText(heading6, "four", "orange", 1000))

  .then(()*=>*changeText(heading7, "four", "orange", 1000))

  .then(()*=>*changeText(heading8, "four", "orange", 1000))

  .then(()*=>*changeText(heading9, "four", "orange", 1000))

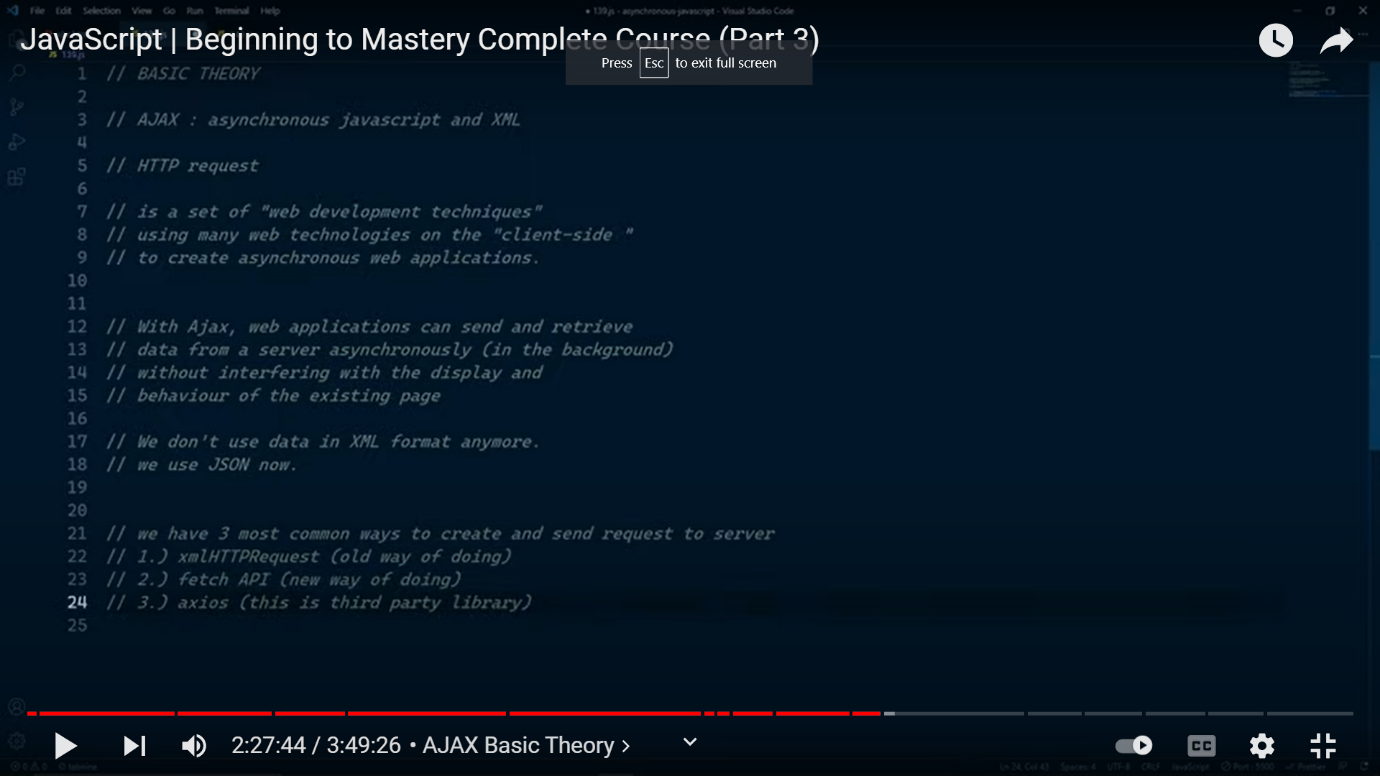
  .then(()*=>*changeText(heading10, "four", "orange", 1000))

  .catch((*error*)*=>*{

      alert(*error*);

  })

Ajax : asynchronous JavaScript and xml



Full form of http : hyper text transfer protocol .

Way to inspect network in console :

Open inspect ->click network->click all if present->if not then click filter icon then click all -> then refresh the page .

Link of dumy API :

<https://jsonplaceholder.typicode.com/>

🡺 Creating and sending resquest to server :

Method 1:xmlHTTPRequest (xhr):

a)Readystate

**XMLHttpRequest.readyState**

The **XMLHttpRequest.readyState** property returns the state an XMLHttpRequest client is in. An XHR client exists in one of the following states:

| **Value** | **State** | **Description** |
| --- | --- | --- |
| 0 | UNSENT | Client has been created. open() not called yet. |
| 1 | OPENED | open() has been called. |
| 2 | HEADERS\_RECEIVED | send() has been called, and headers and status are available. |
| 3 | LOADING | Downloading; responseText holds partial data. |
| 4 | DONE | The operation is complete. |

Step1 :

*const* url="https://jsonplaceholder.typicode.com/posts";

*const* xhr=new *XMLHttpRequest*();

// console.log(xhr);

xhr.open("GET",url);

 xhr.onreadystatechange=()*=>*{

    console.log(xhr.readyState);

 }

xhr.send();

note :it will going to give me information about the different -different readystate . we can get information about the ready state change by xhr.onreadystatechange

step 2 :in this step we will fetch the data from the api .

*const* url="https://jsonplaceholder.typicode.com/posts";

*const* xhr=new *XMLHttpRequest*();

// console.log(xhr);

xhr.open("GET",url);

 xhr.onreadystatechange=()*=>*{

   if(xhr.readyState==4){

*const* result=xhr.response;

*const* data=JSON.parse(result);

   }

 }

xhr.send();

other method instead of : xhr.onreadstatechange is xhr.onload

note : we gernrally going to use xhr.load instead of xhr . onreadystagecange .

*const* url="https://jsonplaceholder.typicode.com/posts";

*const* xhr=new *XMLHttpRequest*();

// console.log(xhr);

xhr.open("GET",url);

 xhr.onload=()*=>*{

    if(xhr.readyState==4){

*const* result=xhr.response;

*const* data=JSON.parse(result);

     console.log(data);

    }

  }

xhr.send();

to know and use status : xhr.status

:Error heandeling : xhr.onerror

Xhr callback hell :

Suppose we want to get vaule at id : 3 ;

*const* url="https://jsonplaceholder.typicode.com/posts";

*const* xhr=new *XMLHttpRequest*();

// console.log(xhr);

xhr.open("GET",url);

 xhr.onload=()*=>*{

    if(xhr.status>=200 && xhr.status<300){

*const* result=xhr.response;

*const* data=JSON.parse(result);

        //connecting with the id=3

*let* url2=`${url}/${data[2].id}`;

*let* xhr2=new *XMLHttpRequest*();

        xhr2.open("GET",url2);

        xhr2.onload=()*=>*{

            if(xhr.status>=200 && xhr2.status<300){

*let* data2=JSON.parse(xhr2.response);

                console.log(data2);

            }

        }

        xhr2.onerror=()*=>*{

            console.log("Some error has occured !");

        }

        xhr2.send();

    }

  }

xhr.onerror=()*=>*{

    console.log("some error occur");

}

xhr.send();

removing xhr callback hell by promise :

*function* connect(*method*,*url*){

    return new *Promise*((resolve,reject)*=>*{

*let* xhr=new *XMLHttpRequest*();

        xhr.open(*method*,*url*);

        xhr.onload=()*=>*{

            if(xhr.status>=200 && xhr.status<300){

                resolve(xhr.response);

            }

            else{

                reject("erro in connecting");

            }

        }

        xhr.onerror=()*=>*{

            reject("Error in onnection");

        }

        xhr.send();

    })

};

*const* url="https://jsonplaceholder.typicode.com/posts";

connect("GET",url).then((*data*)*=>*{

*let* result=JSON.parse(*data*);

    return connect("GET",`${url}/${result[2].id}`);

}).then((*data*)*=>*{

    console.log(*data*);

}).catch((*error*)*=>*{

    console.log(*error*);

})

Method 2 of AJX: fetch

Step 1 :

*const* url="https://jsonplaceholder.typicode.com/posts";

fetch(url).then((*Response*)*=>*{

    console.log(*Response*);

})

Step2 :

*const* url="https://jsonplaceholder.typicode.com/posts";

fetch(url).then((*Response*)*=>*{

    return *Response*.json();

}).then((*Response*)*=>*{

    console.log(*Response*);

})

Final :

*const* url="https://jsonplaceholder.typicode.com/posts";

fetch(url).then((*Response*)*=>*{

   if(*Response*.ok){ // response.ok check the error

    return *Response*.json();

   }

   else{

    throw new *Error*("something went wrong");

   }

}).then((*Response*)*=>*{

    console.log(*Response*);

}).catch((*error*)*=>*{

    console.log("inside the catch");

    console.log(*error*);

});

🡺

Creating a resource or api :

fetch('https://jsonplaceholder.typicode.com/posts', {

  method: 'POST',

  body: JSON.stringify({

    title: 'foo',

    body: 'bar',

    userId: 1,

  }),

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  },

})

  .then((*response*) *=>* *response*.json())

  .then((*json*) *=>* console.log(*json*));

Output:

{

  id: 101,

  title: 'foo',

  body: 'bar',

  userId: 1

}

Note : very important link must open link :

<https://jsonplaceholder.typicode.com/guide/>

it contains information about :

1)get

2)post

3)put

4)delete

5)patch

6)filter

🡺 asny and await

We can remove then from the promise changing .

*const* url="https://jsonplaceholder.typicode.com/posts";

*let* connect=async ()*=>*{

*let* response=await fetch(url);

*let* data=await response.json();

    console.log(data);

}

connect();

for error handeling :

*const* url="https://jsonplaceholder.typicode.com/post";

*let* connect=async ()*=>*{

    try {

*let* response=await fetch(url);

        if(response.ok){

*let* data=await response.json();

            console.log(data);

        }

        else{

            throw new error();

        }

    }

     catch (error) {

        console.log(error);

    }

}

connect();

note : we can print : also -> error.name or error.message

exprort and import :

we can export one function from one file of java script to another file with the help of export and import .

remember to use : type=”module” in script file after that no need to use defer .

<script src="pr.js" type="module"></script>

type 1 :

export file :

 export default   *function* sum(*a*,*b*){

    return *a*+*b*;

}

  Export default *function* sub(*a*,*b*){

    return *a*-*b*;

}

 export  *function* multiply(*a*,*b*){

    return *a*\**b*;

}

  export *function* divison(*a*,*b*){

    return *a*/*b*;

};

Import file :

import {sum,divison,sub,multiply} from "./file1.js";

console.log(sum(1,2));

type 2 : export default :

i)after using export default we do not need to use {} eg :

export default :

 export default   *function* sum(*a*,*b*){

    return *a*+*b*;

}

import sum from "./file1.js";

console.log(sum(1,2));

note : we can only we export default only once in file .

eg : it will give error :

 export default   *function* sum(*a*,*b*){

    return *a*+*b*;

}

  Export default *function* sub(*a*,*b*){

    return *a*-*b*;

}