***Concurrency / Asynchronous javascript:***

***Concurrency In javascript :***

Whenever two things are happening at the same time. Means in parallel more than one function is running at the same time.

If your application is not asynchronous you cant listen music while writing a code or using your coded application.

We can do a call over API and not wait for its reply instead of it do something useful in the meantime.

The concept is called concurrency.

In order to do concurrency we applied it using following things :

**Async , await , callbacks , promises**

***Synchronous javascript:***

Javascript is syncrhounous by default , means we can do one task at the same time.no more than one task can run in parallel at the same time. Once a function is invoked and working until it is finished nonetheless other functions will start working until current function does his assigned work.

Every statement in a code executed one by one , similarly every function executed in sequence.

***Asynchronous javascript:***

Multiple tasks happening at the same time immediately , in a bigger picture where hundreds of functions present in the same js file , it is very much delaying with synchornouse default working of javascript.we can use

**Set time out , set interval , async / await there.**

To make it asynchronous , and overcome delaying problem.

***Callbacks :***

It just take another function as an argument , while assigning callback into mainfunction alongwith passing argument in a **callback();** function of first function we want to be call backed.

So there were two functions works out at the same time. Bingo ! concurrency works.

Example 01 :

function doSomething(callback) {

    callback();

   }

   function sayHi() {

    console.log("Hi!");

   }

   doSomething(sayHi);  /

Example 02 :

function hey(ret){

    ret="yes"

  console.log("return works!"+ret);

}

function there(a,b,callback){

    let ret;

    addup=a+b;

    console.log("another function : "+addup);

    callback(ret);

}

there(3,4,hey); // passing another function in a function as a refrence using callback called out other function

example 03 :

function judge(grade) {

    switch (true) {

    case grade == "A":

    console.log("You got an", grade, ": amazing!");

    break;

    case grade == "B":

    console.log("You got a", grade, ": well done!");

    break;

    case grade == "C":

    console.log("You got a", grade, ": alright.");

    break;

    case grade == "D":

    console.log("You got a", grade, ": hmmm...");

    break;

    default:

    console.log("An", grade, "! What?!");

    }

   }

   function getGrade(score, callback) {

    let grade;

    switch (true) {

    case score >= 90:

    grade = "A";

    break;

    case score >= 80:

    console.log(score);

    grade = "B";

    break;

    case score >= 70:

    grade = "C";

    break;

    case score >= 60:

    grade = "D";

    break;

    default:

    grade = "F";

    }

    callback(grade);

   }

   getGrade(85, judge);

main objective is to callout apis from third party database.

***Set timeout & set interval :***

Some js builtin functions works in callback principle, for example settimeout and settimeinterval functions.

For set time out the function assigned to them is executed under certain amount of time.

Every certain amount of time is specified for interval.

1 second = 1k mili second

Example of set timeout:

setTimeout(myfunction,300); // the function works out after 300 milisecond crossed

function myfunction(sumup){

   var sumup = 6+6;

   return console.log(sumup);

}

Example of set timeinterval:

setInterval(encourage, 5000); // time delay 500 ms (mili seconds)  ,  encourage passed as an argument into callback form

function encourage() {

 console.log("You're doing great, keep going!");

 console.log("You're doing great, stoping!");

}// the function keeps work after 5 thausand mili second delay and second time it works again after 5k ms delay

Example 02 :

setInterval(myFunction2, 1000);// with interval of every second time works out.

function myFunction2() {

  let d = new Date();

console.log(d.getHours() + ":" +

d.getMinutes() + ":" +

d.getSeconds());

}

Note ! the function automatically callout when builtin function set time interval or timout writtern.

***Promises:***

A promise is the javascript object that links producing code and consuming code.

Prod code: take take some time.

Consuming code : that code must wait for the result of prod code to be appear.

A promise is a special objects that connects code that needs to produce a result and the code that needs to use this result.

*Syntax of creating a promise :*

1. Creates an object as new promise. // let myPromise = new Promise
2. Passing function // (function
3. Function arguments usually name them resolve and reject // (myResolve, myReject)
4. If resolve is called promise.then’s block body success is return  // myResolve(); when successful
5. If reject is called promise.then’s block body failure is return // myReject();  when error
6. With object instance then function is called , which contains function arguments either defined values are successfully worked or not. //myPromise.then(  
     function(value) { /\* code if successful \*/ },  
     function(error) { /\* code if some error \*/ }  
   );

Connecting all above syntax breakdown we got promise complete structure :

let myPromise = new Promise(function(myResolve, myReject) {  
// "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
  function(value) { /\* code if successful \*/ },  
  function(error) { /\* code if some error \*/ }  
);

Example of a promise:

// object created named obj , promise inbuilt function called !

let obj=new Promise(function(resolve,reject){ // arugments rescolve , reject

// "Producing Code" (May take some time)

let x = 3;

    if(x>1){

        resolve(x);

    }

    else(

        reject("too low")

    )

});

// "Consuming Code" (Must wait for a fulfilled Promise)

obj.then( // instacne of object used with then where then is function itself used to check if function worked out as required or not

function(value){

    console.log("Success ! ",value);

},

function(value){

console.log("Failure ! ",value);

}

);

Example 02 :

let promise = new Promise((work,notworked) => {

   var valued="1";

   //var checktype=typeof(valued);

 var checkout=true;

   if(isNaN(valued)){

    notworked("not a number");

   }

   else{

    work(valued);

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

   }

})

.then(value => {

    console.log(value);

    return 1;

})

.then(value => {

    console.log(value);

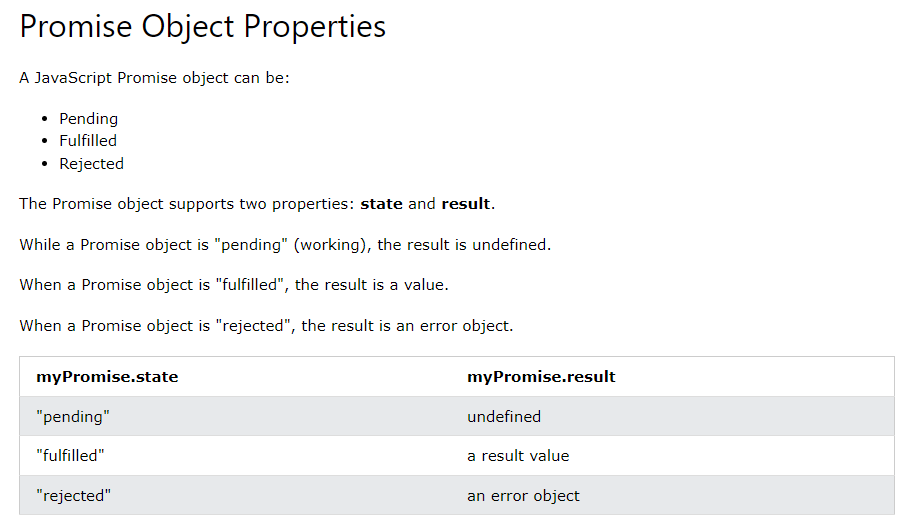
    return 2;

})

.catch(value => {

console.log(value);

})



**Promise.then()** takes two arguments, a callback for success and another for failure.

Both are optional, so you can add a callback for success or failure only.

Example 03 :

function myDisplayer(some) {

    document.getElementById("demo").innerHTML = some;}

let mypromise = new Promise (function (resolve,reject) {

    let req = new XMLHttpRequest();

    req.open('GET', "index.html");

    req.onload = function() {

      if (req.status == 200) {

        resolve(req.responseText);

      } else {

        reject("Error: " + req.status);

      }

    }

    req.send();

  });

  mypromise.then(

    function(value) {myDisplayer(value);},

    function(error) {myDisplayer(error);}

  );

Example 04 :

  let myPromise = new Promise(function(myResolve, myReject) {

        setTimeout(function() { myResolve("I love You !!"); }, 3000); // message showed with the delay of 3 seconds

  });

  myPromise.then(function(value) {

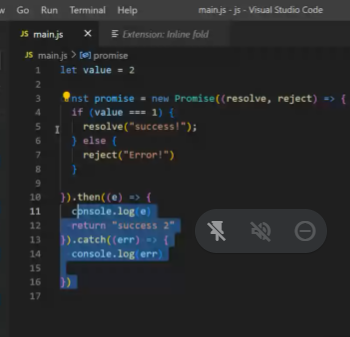
    document.getElementById("demo").innerHTML = value;

  },

  function(error) {console.log(error);}

  );

Used for api fetching from database in the json format.



We pass api address to the resolve function as an argument to use the fetched data.

Example another:

const passtypes =["431","312","adx"];

function checkout(pass){

    return passtypes.includes(pass);

}

function login(password){

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

  if(checkout(password)){

    resolve({

status : true ,

show : "Logged in sucessfully"

    });

  }

  else{

    reject({

        status : false

            });

  }

});

}

function checkout2(pass){

    login(pass)

    .then(token => {

    console.log("Approve:");

    console.log(token);

    })

    .catch(value => {

    console.log("Reject:");

    console.log(value);

    })

}

var ask =prompt("enter password :");

checkout2(ask);

***Async & Await:***

Async and await makes promises easier to write. Async make the function return promise , await makes function wait for it turn to be execute.

Aysnc before function wrote makes function return the promise !

**Syntax :**

async function myFunction() {  
  return "Hello";  
}

*is same as :*

function myFunction() {  
  return Promise.resolve("Hello");  
}

complete async shot :

async function myFunction() {  
  return "Hello";  
}  
myFunction().then(  
  function(value) {myDisplayer(value);}  
);

**Syntax (await):**

let value = await promise;

await keyword can only be used inside async function only.

*Sample async await*

async function myDisplay() {

    let myPromise = new Promise(function(resolve, reject) {

      resolve("I love You !!");

    });

    document.getElementById("demo").innerHTML = await myPromise;

  }

  myDisplay();

*with timeout:*

async function myDisplay() {

    let myPromise = new Promise(function(resolve) {

        setTimeout(() => {

            resolve("I love You !!");

        }, 3000);

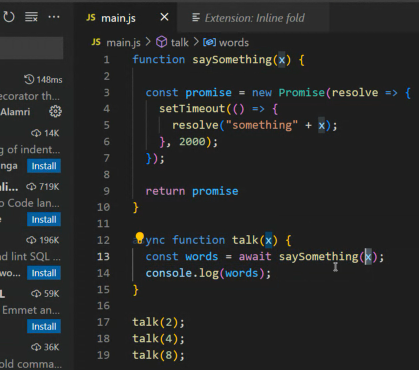
    });

    document.getElementById("demo").innerHTML = await myPromise;

  }

  myDisplay();

Example 01 :

******

If from above picture…talk function isn’t asynchronouse , then obviously we cant use await as well in the function. This results main funtion’s promise goes in pending state.

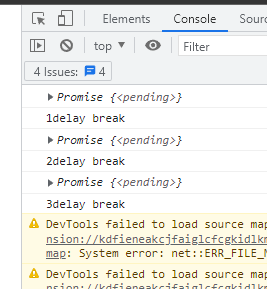
There are three states in promise to be resulted:

* **Resolve**
* **Pending**
* **Reject**

Because evertime we callout our function it goes on pending since it doesn’t know when to execute first argmument call and when other ones .

So make them asynchrounoulsy work , we need to make function async + await.

Lets see what happened if we don’t apply async or await.



See the problem occur.

Example 02 :

function greet(x){    // a first function is made

    const myprom=new Promise(resolve =>{ // adding promise with set time out function of 2 seconds delay

        setTimeout(() => {

            resolve("Something will happen as : "+x); // the settimeout body executes afet 2 seconds delay

        }, 2000);

    });

    return myprom    // promise object name return

}

async function greetcatcher(x){   // async funtion

    const talkout= await greet(x);   // greet catcher waits for timout of 2 seconds held off to show message inside promise

    console.log(talkout);           // message under await , where await waits 2 seconds for first function called.

}

greetcatcher("function called after 2 sec delay !");   // function message print run after 2 seconds.

console.log(1+"delay break");

greetcatcher("function called after 2 sec delay second time !");

console.log(2+"delay break");

greetcatcher("function called after 2 sec delay third time !");

console.log(3+"delay break");

Example 03:

          // THROUGH AJAX CALLOUT OTHER EXTERNAL/INternal PAGES alongwith async and promise , await

          async function getFile(){

            let promi=new Promise(function (resolve,reject){

            let req = new XMLHttpRequest();  // ajax calling

            req.open("GET","index.html");  // giving path and operation to perform on triggered file here open file.

            req.onload = function(){

                if(req.status == 200){

                    resolve(req.response);

                }

                else{

                    reject("file not availble in the triggered directory !");

                }

            };

            req.send();

            });

            document.getElementById("demo").innerHTML = await promi;

            }

            getFile();  //callout whole function

***Event Loop :***

Javascript is single threaded means it can handle one task at a time , so it needs to be asynchorouny which is we learnt how to do manually.

Well, even though JavaScript is single-threaded, it doesn't mean that it cannot outsource some tasks and wait for them to come back. This is exactly how JavaScript manages to do things in a multithreaded manner.

That’s why we use event loop

**Call stack and callback**

queue JavaScript works with a call stack, and all the actions that it has to execute are queued up here. The event loop is a process that is constantly monitoring this

call stack, and whenever there are tasks to do, the event loop does them one by one. The tasks on top get executed first.

 This will output the exact same thing. **setTimeout()** will also be outsourced when the timer is at 0 . The **callback** is placed in the callback queue right away, but the event loop won't even check the **callback queue** until the **callstack** is empty. So it will still print Sorry I'm late after 9 , even though the timer is at 0