Synchronous -> line by line

Asynchronous -> with delay

Best example is of facebook. Like when we open fb, it shows news feed along with new msgs and post, it takes new post from server, so it might take some time. So if line by line code runs, then in that case after opening fb, first new posts and notifications comes then we can see news feed, but that is not the case. This is due to Asynchronous. In that case other components runs synchronously(line by line) while new posts and notifications runs Asynchronously on some another server and comes with some delay, so due to Asynchronous, fb is independent of new post and notifications, means to say other things load synchronously line by line while posts and notifications comes with some delay Asynchrnously with effecting its other parts

CALLBACK

Callback-> ek function ko call karo, woh kisi aur function ko call karega(call one function, and that function will call some other function in it). This is in synchronous js.

In asynchronous, we call one function , then it call some other function and let say that other function has some delay due to its working, till then we excute other parts of program and when that other function is completed, we show its results.

Callback function is somehow related to api, used when we request api. Like we said api that first call data, when data comes then call function, then that function comes then you call some other function and hierarchy goes on and on.

CALLBACK HELL(PYRAMID OF DOOM)

When we create function inside function and follow procedure for like 500 to 1000 lines, then this create mess, like we ourselves confused which function calling some other function. This is called callback hell. To get rid from this we use promises.

In short, call back hell is unwieldy number of nested if statements or functions

PROMISES:

Promises manages asynchronous function and safe us from callback hell

Its like normal promise, either it will complete or it will fail, and these two terms in js are called resolve and reject.

These callbacks and promise used in api fetching. If we fetch api and we get data successfully, then promise will run resolve function, otherwise if api through some error, then we run reject function. In both condition, it will run compulsory. Either fail or success.

Promise is an object. To create its instance, we use new Promise

Const promisename=new Promise((resolve, reject)) => {

}

As fetching api takes time, so promise is an asynchronous function which runs separately and in the mean time other parts of application load and runs, and when result from fetching api came, promise shows, if result is success then promise runs function of resolve otherwise it runs function of reject.

As fetching data from api take some time, so just for example we place some random function into setTimeout, this is not done in case of api, so other parts of application run without any disturbance while fetching api run in some other server.

1. This is ome random function working as a api in this case, not in actual

const anyFunction = () => {

    return (((Math.floor(Math.random()\*10))%2) === 0) ? true:false

}

Const promisename = new Promise((resolve, reject) => {

setTimeout(function()=>( anyFunction() ? resolve() : reject()), 2000) //this will fetch api in some otherserver

}

const result = new Promise((resolve, reject)=>{

    setTimeout(()=>(anyFunction() ? resolve(): reject()), 2000)

})

without interruption and if successful then return resolve function otherwise reject function with 2 sec delay.

When fetching api done, if it is successful then resolve function run otherwise rejectfunction run.

For resolve we use then and incase of reject, we say promise to catch that error and immediately run reject function

1. Resolve and reject functiona as per we need

const resolveFunction = () => {console.log("success")}

const rejectFunction = () => {console.log("failure")}

Promisename.then(resolvefunction).catch(rejectfunction)

result.then(resolveFunction).catch(rejectFunction)

console.log("normal synchrnous 1");//these two lines run normally as promise is an asynchronous and they are synchronous, so they print immediately while promise take 2 sec to print just like api take time to fetch the data

console.log("normal synchronous 2");

ASYNC AND AWAIT

Await is like wait till promise is done and then return resolve response which we can store in some variable and can print it

Async is called with function declaration and await with function which we want to wait

async function startProcess(){

    let foodvalue = await prepareFood();//wait till preparefood complete and returns either reject or resolve response which will store in foodvalue and we can print it

    console.log("food value=", foodvalue);

}

startProcess();

//this saves us from call complete heirarchy of promise

By using async, we get rid of long hierarchy of then, we can use await as many time as we want with many function

So it simplify calling a async function which contains timout or api. We get rid of then, we just need to use await with delayed or async function( function needs some time or function which has setTimeout or api to fetch) and store its resolve response in some variable which we can print

async function startProcess(){

    let foodvalue = await prepareFood();//wait till preparefood complete and returns resolve response which will store in foodvalue and we can print it

    console.log("Food=", foodvalue)

    let toastvalue = await prepareFrenchToast();

    console.log("Toast=", toastvalue);

    let coffeevalue = await prepareCoffee();

    console.log("Coffee=", coffeevalue);

}

startProcess();

So in short, await replace then and simplify.

What if some function returns reject when calling with await, we just need to use try and catch syntax there. If response is resolve then we know await will run and if response is reject, then to catch the error, we use catch and for resolve we place code in try just like try and except in python

async function startProcess(){

    try{  let foodvalue = await prepareFood(2);//as we have passed 2, so as per our if else condition in prepareFood, rejects run, so await only handles resolve, to handle reject or error, we simply use try catch where catch is for the error

        console.log("Food=", foodvalue)

        let toastvalue = await prepareFrenchToast();

        console.log("Toast=", toastvalue);

        let coffeevalue = await prepareCoffee();

        console.log("Coffee=", coffeevalue);

    }

    catch(error){

        console.log("Error =", error)

    }

}

startProcess();