//callback function

const posts=[

{

title:'post one', body:'this is post one'

},

{

title:'post two', body:'this is post two'

}];

function getposts(){

setTimeout(function(){

let output='';

posts.forEach(function(postVal,index){

output+=`${postVal.title}`;

})

console.log(output);

},1000);

}

/\*function createpost(post){

setTimeout(function(){

posts.push(post);

},2000)

}

getposts();

createpost({title:'post three',body:'this is post three'});\*/

//here we are not getting desired output for third post, because create post is

//taking more than get post time

function createpost(post,callback){

setTimeout(function(){

posts.push(post);

callback();

},2000)

}

createpost({title:'post three',body:'this is post three'},getposts);

//here this call back will solve our problem, after 2 sec we will get desired output

// resolving same issue with promise

const posts=[

{

title:'post one', body:'this is post one'

},

{

title:'post two', body:'this is post two'

}];

function getposts(){

setTimeout(function(){

let output='';

posts.forEach(function(postVal,index){

output+=`${postVal.title}`;

})

console.log(output);

},1000);

}

function createpost(post){

return new Promise(function(resolve,reject){

setTimeout(function(){

posts.push(post);

const error=false;

if(!error){

resolve();

}else{

reject('Error: something went wrong!');

}

},2000)

});

}

createpost({title:'post three',body:'this is post three'})

.then(getposts).catch((err)=>{

console.log("the error is: ",err);

})

//promise all: wait till all promise will get executed, so whateve

//promise will take maximum time it will wait till that

const promise1=Promise.resolve('hello world!');

const promise2=10;

const promise3=new Promise((resolve,reject)=>{

setTimeout(resolve,2000,'good bye');

})

const promise4=fetch('https://jsonplaceholder.typicode.com/users').then(user=>{

return user.json();

})

Promise.all([promise1,promise2,promise3,promise4]).then(value=>{

console.log(value);

})

//async await

//see below example, without asysnc await we will get error: res.json is not a function

//because it not waiting for response

// function fetchusers(){

// console.log("before fetch");

// const res= fetch('https://jsonplaceholder.typicode.com/users');

// const data= res.json();

// console.log(data);

// console.log("after data");

// }

async function fetchusers(){

console.log("before fetch");

const res=await fetch('https://jsonplaceholder.typicode.com/users');

const data=await res.json();

console.log(data);

console.log("after data");

}

fetchusers();

//call(),apply(),bind()

You can use call()/apply() to invoke the function immediately. bind() returns a bound function that, when executed later, will have the correct context ("this") for calling the original function. So bind() can be used when the function needs to be called later in certain events when it's useful.

//Demo with javascript .call()

var obj = {name:"Rajeev"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

console.log(greeting.call(obj,"Whitefield","Bangalore","KA"));

// returns output as welcome Rajeev to Whitefield Bangalore in KA

*//Demo with javascript .apply()*

**var** obj = {name:"Rajeev"};

**var** greeting = **function**(a,b,c){

**return** "welcome "+**this**.name+" to "+a+" "+b+" in "+c;

};

*// array of arguments to the actual function*

**var** args = ["Whitefield","Bangalore","KA"];

console.log("Output using .apply() below ")

console.log(greeting.apply(obj,args));

*/\* The output will be*

*Output using .apply() below*

*welcome Rajeev to Whitefield Bangalore in KA \*/*

*//Use .bind() javascript*

***var*** *obj = {name:"Rajeev"};*

***var*** *greeting =* ***function****(a,b,c){*

***return*** *"welcome "+****this****.name+" to "+a+" "+b+" in "+c;*

*};*

*//creates a bound function that has same body and parameters*

***var*** *bound = greeting.bind(obj);*

*console.dir(bound); ///returns a function*

*console.log("Output using .bind() below ");*

*console.log(bound("Whitefield","Bangalore","KA")); //call the bound function*

*/\* the output will be*

*Output using .bind() below*

*welcome Rajeev to Whitefield Bangalore in KA \*/*

Summary for call,apply,bind:

functionName.call(object,arg1,arg2,arg3);//parameters as comma separated

functionName.apply(object,[arg1,arg2,arg3]);//parameters as array

Var boundData=functionName.bind(onj);//same as call, but first need to bind then call that function with comma separated parameters

bound(arg1,arg2,arg3)