**Async/Await Assignment**

1). How does async/await help with performance and scalability?

**async** makes a function return a Promise

**await** makes a function wait for a Promise

Increase the performance and **responsiveness** of your application, particularly when you have long-running operations that do not require to block the execution. In this case, you can perform other work while waiting for the result from the long running task. This inturn enhance speed and scalability. The response time of one request does not affect the response time of others. Due to its non-blocking nature, async can send multiple requests to the server. Due to its ability to run multiple operations simultaneously, async increases the throughput.Due to the non-blocking nature, this will help to achieve high performance. Async/await helps us to write less code and that code is easy to maintain.

2). Is it possible to use async/await with promise chains? If yes,

how can this be achieved?

YES, We can use async/await with promise chains. Promise comes into picture inorder to avoid the problem of callback hell. First, await the initial promise in the chain,then continue the execution with the results of the promise,if it is synchronous.

function a() {

   b();

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

    setTimeout(() => {

      resolve('resolved a');

    }, 2000);

  });

}

function b() {

  console.log('calling b');

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

    setTimeout(() => {

      resolve('resolved b');

    }, 2000);

  });

}

 async function asyncCall() {

  console.log('calling async function');

  const result = await a();

  console.log("res----",result);

}

asyncCall();

output:

calling async function

calling b

res---- resolved a

3). Give 3 real world examples where async/await has been used?

1.At hotel/Restaurant

From one table Waiter gets the order list from the customer/consumer…until it is ready ..waiter goes to next table and performs supplying of food when the food is ready.here, the waiter does not supply the ordered food till it is ready..when the promise of ordered food is resolved then only the supplier supplies the food.

2. Even our daily Routine follow async/await…When the brushing activity promise is resolved then only we can have breakfast..after bathing promise is resolved..then only we can dressup well and then go to office.

4.Find output.

async function inc(x) {

x = x + await 1

return x;

}

async function increment(x){

x = x + 1

return x

}

inc(1).

then(function(x){

increment(x).then(function(x){

console.log(x)

})

})

Output: **3**

First step:inc(1)---is called..this will execute with x value as 1.. x=x+await , this will return x as 2..

async function inc(1) gives 2..after that x as 2 enters increment(x)function then x value computed as 2+1 , 3..and returned..then this value enters console.log then it is printed.

5.Find output.

let p = new Promise(function (resolve, reject)

{

reject(new Error("some error"));

setTimeout(function(){

reject(new Error("some error"));

},1000)

reject(new Error("some error"));

});

p.then(null, function (err)

{

console.log(1);

console.log(err);

})

.catch(function (err) {

console.log(2);

console.log(err);

});

Output:

1

Error: some error

Even promise having reject condition..p.then()is executing.. reject(new Error("some error"));even in the above program, 2 reject condition with error message is used and third one with settimeout..first reject(error) is considered. this “some error” is passed as err in function,so logging 1 and then printing some error. Usually catch is used for reject condition.if function(err)is not paused in p.then….then catch(err) will be executed.

6. Find output

async function f1() {

console.log(1);

}

async function f1() {

console.log(2);

}

console.log(3); prints //3

f1();

console.log(1);

f2();

async function f2() {

console.log("Go!");

}

Output:

3

2

1

GO

First console.log(3)—prints 3…then f1()is called..here revised f1()will be called so it prints 2..then usual 1 is logged then f2 is called..so GO will be printed.

7.Find output:

function resolveAfterNSeconds(n,x) {

return new Promise(resolve => {

setTimeout( ( ) = {

resolve(x);

}, n);

});

}

(function(){

let a = resolveAfterNSeconds(1000,1)

a.then(async function(x){

let y = await resolveAfterNSeconds(2000,2)

let z = await resolveAfterNSeconds(1000,3)

let p = resolveAfterNSeconds(2000,4)

let q = resolveAfterNSeconds(1000,5)

console.log(x+y+z+await p +await q);

})

})()

Output:

15

Function(){} is called then resolveAfterNseconds is called with a timer and x value..from that promise is resolved and x value obtained is 1..and then it is stored in a;when a is resolved.then a.then function will be called…after that y value need to wait till resolveAfterNSeconds(2000,2) is resolved..like this z also need to resolved.then p and q is computed then value is computed as 15.

8). Is it possible to nest async functions in JavaScript? Explain with

examples.

async function fun2() {

    console.log('fun2 called');

    let fun2Promise = new Promise(function (resolve, reject) {

      setTimeout(() => resolve('fun2 done'), 1000);

    });

    console.log(await fun2Promise);

  }

  async function fun3() {

    console.log('fun3 called');

    let fun3Promise = new Promise(function (resolve, reject) {

      setTimeout(async () => {

        await fun2();

        resolve('fun3 done');

      }, 2000);

    });

    console.log(await fun3Promise);

  }

  async function fun4() {

    console.log('fun4 called');

    let fun4Promise = new Promise(function (resolve, reject) {

      setTimeout(async () => {

        await fun3();

        resolve('fun4 done');

      }, 5000);

    });

    console.log(await fun4Promise);

  }

fun4();

output:

fun4 called

fun3 called

fun2 called

fun2 done

fun3 done

fun4 done

fun4()is called, when before f4()promise resolves,await fun3(),so it waits till fun3()promise is resolved..within fun3(), await fun2(),so again fun3 waits till fun2 resolves its promises..when fun2 is done after that fun3() is done after that fun4() willbe done.

9). What is the best way to avoid deadlocks when using

async/await?

10). In which scenarios would you use synchronous code instead of

asynchronous code?

Async is multi-thread, which means operations or programs can run in parallel. Sync is **single-thread, so only one operation or program will run at a time**

Asynchronous is a non-blocking architecture, so the execution of one task isn't dependent on another. Tasks can run simultaneously. **Synchronous is a blocking architecture, so the execution of each operation is dependent on the completion of the one before it**.

synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed. the codes work in a sequence. Every line of code waits for its previous one to get executed first and then it gets executed.

When synchronous code can be used instead of asynchronous code is that..when the execution of each operation is doesnot depend on one another..and resouces is fairly available.