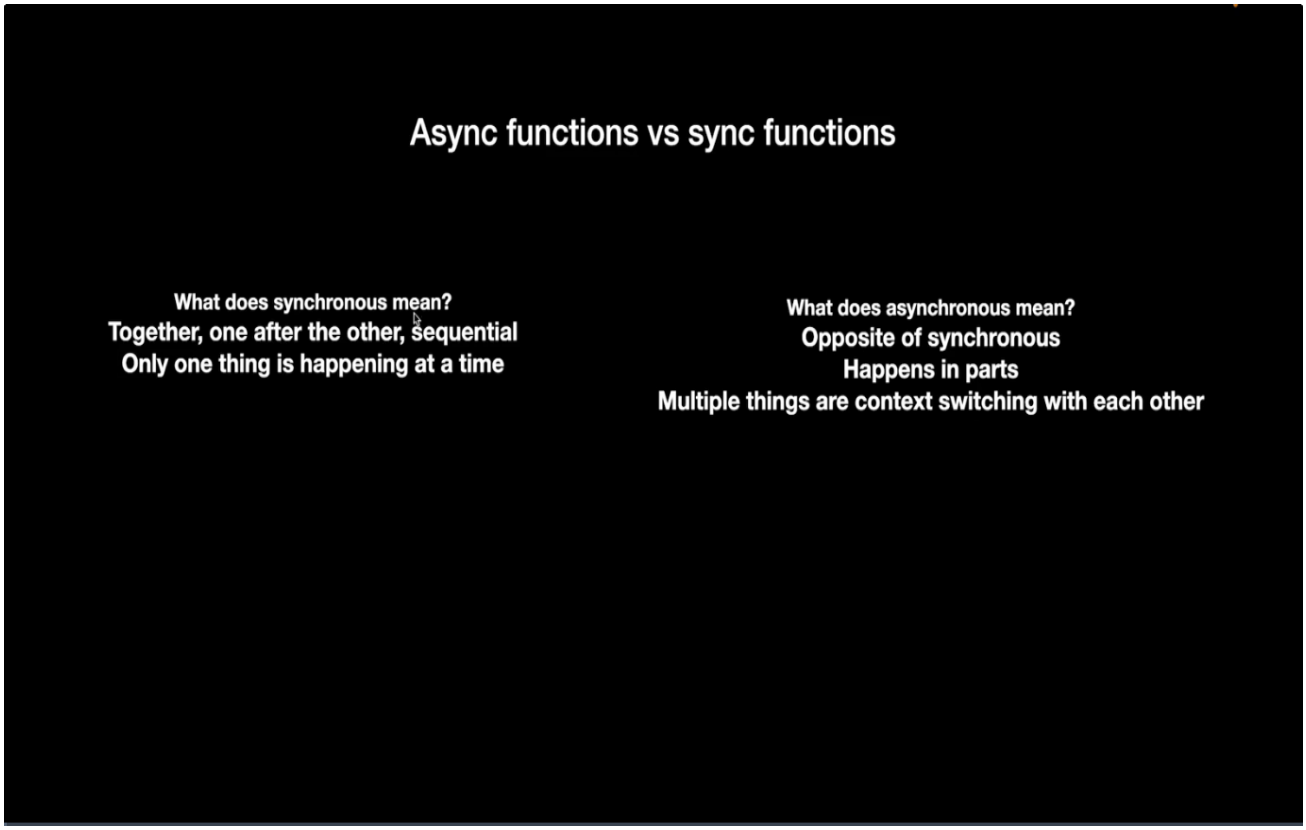
HARKIRAT DEVOPS

LEC-2 [Async, Await and Promises]

Async Functions Vs Sync Functions



If we have to perform several tasks then either we delegate our task or context switch(do all work bit by bit parallely).

Javascript also does the same.

JS uses the asynchronous function to delegate tasks and to make a context switch.

Async functions vs sync functions

Lets introduce an asynchronous function (setTimeout)

https://gist.github.com/hkirat/a75987a32fdfcab27672410930327f1a

```
function findSum(n) {
  let ans = 0;
  for (let i = 0; i<n; i++) {
    ans += i;
  }
  return ans;
}

function findSumTill100() {
  return findSum(100);
}

setTimeout(findSumTill100, 1000)
console.log("hello world");
```

Calling an async function

Console.log(“hello world”); will print first then after 1 sec setTimeout will run.

Some Common Async Functions

What are common async functions?
setTimeout
fs.readFile - to read a file from your filesystem
Fetch - to fetch some data from an API endpoint

NOTE:- callbacks are generally used in Asynchronous functions.

PROMISES:-

Initially, by using a callback, we are writing this code:-

```
function findSum(n) {
  let ans = 0;
  for (let i = 0; i<n; i++) {
    ans += i;
  }
  return ans;
}

function findSumTill100() {
  return findSum(100);
}

setTimeout(findSumTill100, 1000)
console.log("hello world");
```

By the usage of promises, we can make this code more readable otherwise rest of the functionalities are the same.

We can create our own Asynchronous function by the usage of promises.

index.js > kiratsReadFile > ...

```javascript
const fs = require('fs');

// my own asynchronous function
function kiratsReadFile() {
  return new Promise(function(resolve) {
    fs.readFile("a.txt", "utf-8", function(err, data) {
      resolve(data);
    });
  })
}

// callback function to call
function onDone(data) {
  console.log(data)
}

kiratsReadFile().then(onDone);
```

The Promise function is the object of the Promise class where the argument inside the Promise function should be a function that itself has a first argument as resolve.

This resolve(data) and .then() are closely related, which means whatever we do in resolve() is directly passed into .then().

After resolve(), the .then() determines what to do next with the resolved data.

Ugly code

index.js > ...

```
const fs = require('fs');

// my own asynchronous function
function kiratsReadFile(cb) {
  fs.readFile("a.txt", "utf-8", function(err, data) {
    cb(data);
  });
}

// callback function to call
function onDone(data) {
  console.log(data)
}

kiratsReadFile(onDone)
```

Pretty code

index.js > kiratsReadFile > ...

```
const fs = require('fs');

// my own asynchronous function
function kiratsReadFile() {
  return new Promise(function(resolve) {
    fs.readFile("a.txt", "utf-8", function(err, data) {
      resolve(data);
    });
  })
}

// callback function to call
function onDone(data) {
  console.log(data)
}

kiratsReadFile().then(onDone);
```

Syntax to create a promise:-

What even is a promise?

Whenever u create it, you need to pass in a function as the first argument which has resolve as the First argument

index.js | a.txt

index.js > ...

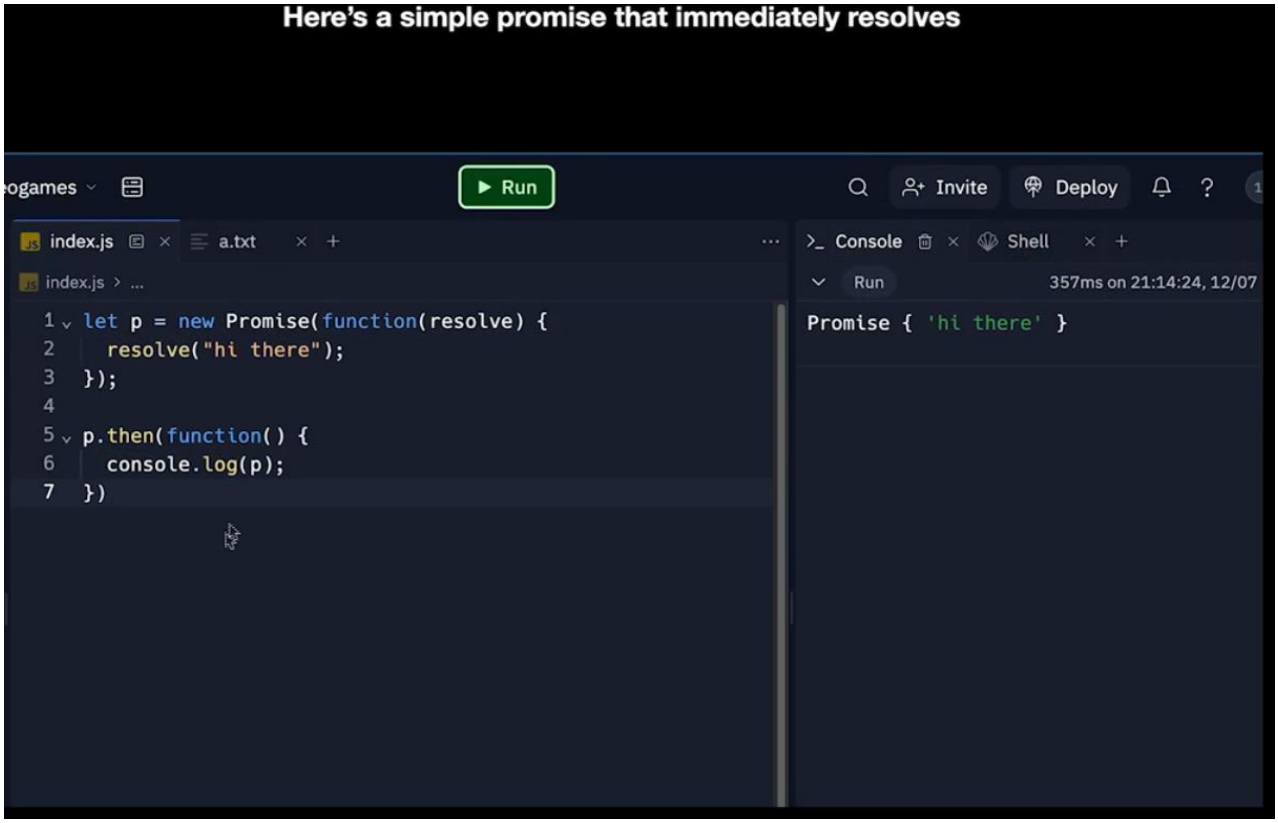
```
let p = new Promise(function(resolve) {

});
console.log(p)
```

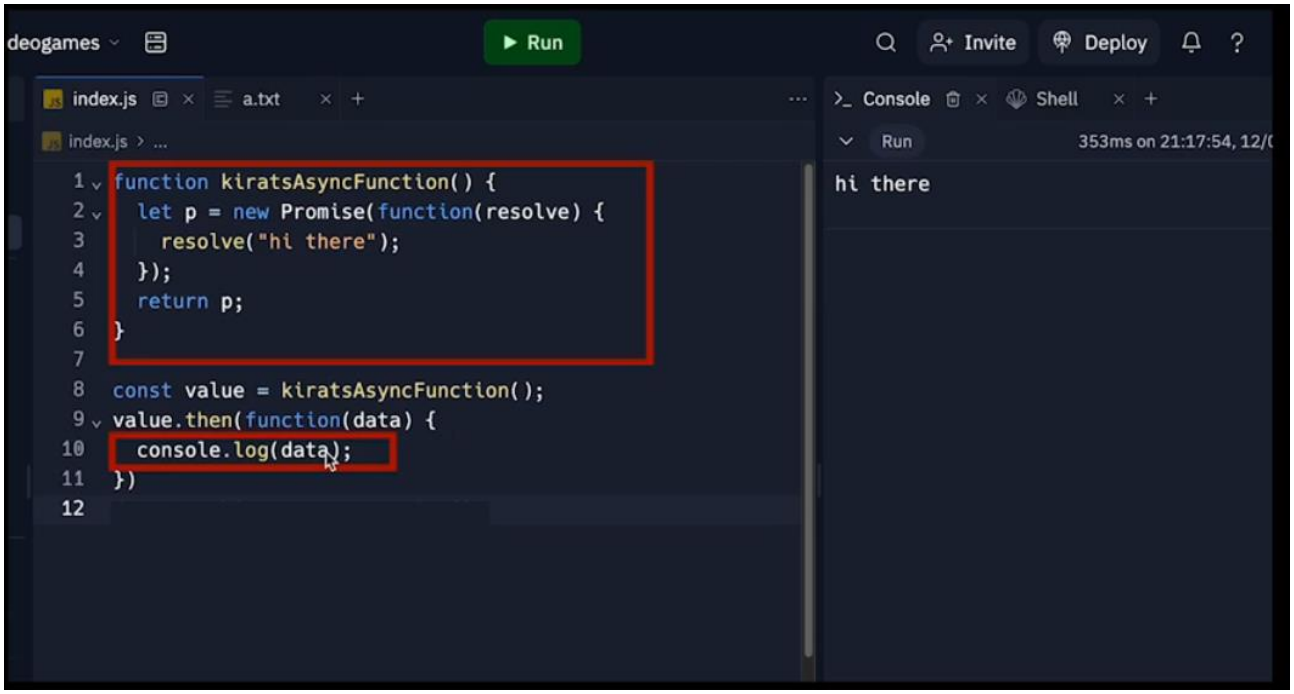
Console | Shell

Run 219ms on 2:13:23,

```
Promise { <pending> }
```



Ex:-



We can write this also:-

```javascript
function ayush(){
  return new Promise(function(resolve){
    resolve("HELLO");
  })
}
ayush().then((val)=>{
  console.log(val)
});

```

They both will do the same thing:-

Intimidating async function

```javascript
function kiratsAsyncFunction() {
  let p = new Promise(function(resolve) {
    setTimeout(resolve, 2000)
  });
  return p;
}

const value = kiratsAsyncFunction();
value.then(function() {
  console.log("hi there");
})
```

Simple function

```javascript
function kiratsAsyncFunction(callback) {
  setTimeout(callback, 2000);
}

kiratsAsyncFunction(function() {
  console.log("hello!");
});
```

ASYNC AWAIT:-

By the usage of async await we write the code of Promises more beautifully.

Normal syntax

index.js × a.txt × +

index.js > ...

```
1  function kiratsAsyncFunction() {
2    let p = new Promise(function(resolve) {
3      // do some async logic here
4      resolve("hi there!")
5    });
6    return p;
7  }
8
9  function main() {
10   kiratsAsyncFunction().then(function(value) {
11       console.log(value);
12   });
13 }
14
15 main();
```

Async/await syntax

index.js × a.txt × +

index.js > ...

```
1  function kiratsAsyncFunction() {
2    let p = new Promise(function(resolve) {
3      // do some async logic here
4      resolve("hi there!")
5    });
6    return p;
7  }
8
9  async function main() {
10   const value = await kiratsAsyncFunction();
11   console.log(value);
12 }
13
14 main();
```

But there is a slight difference:-

```
1  function kiratsAsyncFunction() {
2    let p = new Promise(function(resolve) {
3      // do some async logic here
4      setTimeout(function() {
5        resolve("hi there!")
6      }, 3000)
7    });
8    return p;
9  }
10
11 async function main() {
12   // no callbacks, no .then syntax
13   kiratsAsyncFunction().then(function(value) {
14     console.log(value);
15   })
16   console.log("hi there1");
17 }
18
19 main();
```

This is a normal syntax, but here at line 16," hi there1" will immediately be called whereas "hi there" will be printed after 3 seconds.

```
1  function kiratsAsyncFunction() {
2    let p = new Promise(function(resolve) {
3      // do some async logic here
4      setTimeout(function() {
5        resolve("hi there!")
6      }, 3000)
7    });
8    return p;
9  }
10
11 async function main() {
12   // no callbacks, no .then syntax
13   let value = kiratsAsyncFunction()
14   console.log("hi there1");
15   console.log(value);
16 }
17
18 main();
```

By using Async Await, "hi there1" will called only after when kiratAsyncFunction() gets resolved. And both "hi there" and "hi there1" will be printed together.