Chapter 10

Introduction to **Web Design**

JS

JavaScript Asynchronous Programming

Prepared by **Hok Tin**

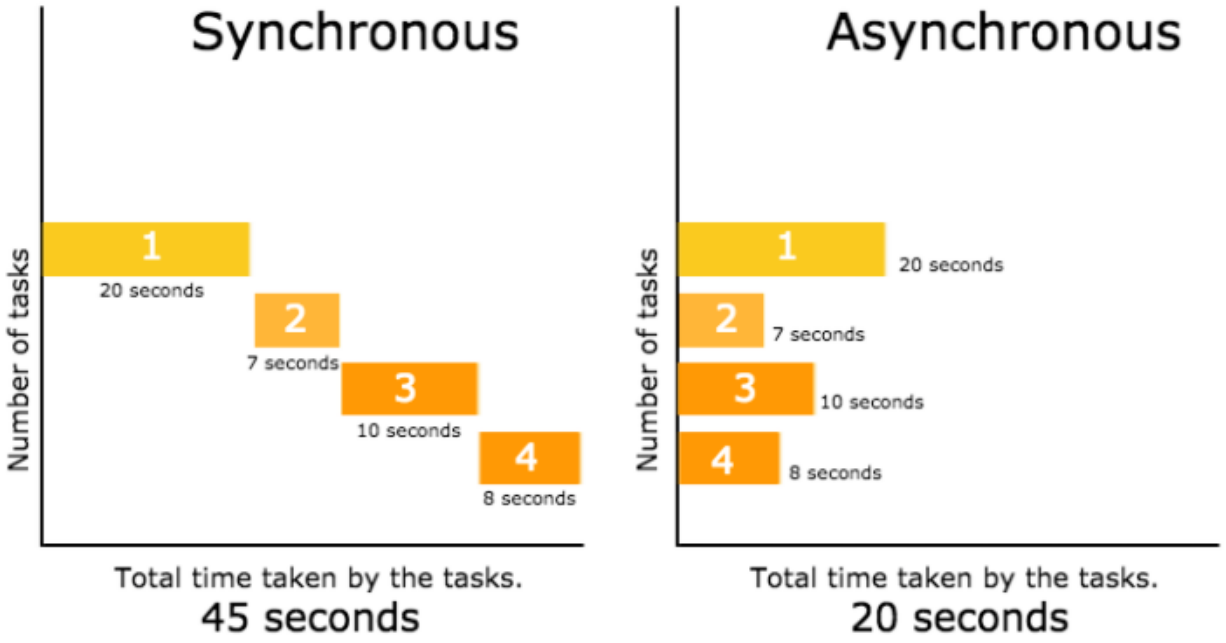
Content

- What is Asynchronous?
- Callback
- Promises
- Async/Await
- Error Handling

What is Asynchronous?

❑ Introduction to Asynchronicity

Understand the difference between **Synchronous** and **Asynchronous**



Callback

❑ Introduction to Asynchronicity

Callback functions are invoked as soon as the blocking function finishes

```
function complete(){
  console.log("function completed!");
  return;
}
setTimeout(complete, 2000.0); // 2000 = 2 seconds of idle time
```

Output

```
function completed!
```

Now, take the previous example but add slight alterations.

```
function complete(name){
  return function (){console.log('function complete: ',name)};
}

setTimeout(complete('setTimeout 5000'), 5000.0);
setTimeout(complete('setTimeout 2000'), 2000.0);
console.log('End of program');
```

Output

```
End of program
function complete:  setTimeout 2000
function complete:  setTimeout 5000
```

Promises

❑ Introduction to Asynchronicity

- A promise is a class meant to neatly *execute asynchronous functions*.
- It acts as a wrapper to execute asynchronous functions with a callback by providing a nicer syntax

Syntax of promises

```
var promise = new Promise(func);
```

→ The function func passed as an argument takes two arguments

- **Resolve function** is invoked, if **it works as intended**. The argument of the resolve function is given the return value.
- **Reject function** is invoked if something **fails**. It is usually used for error handling. The argument of the reject function would be given the return value, which usually is an error object Error.

Promises

❑ Introduction to promises

A **promise** is a class meant to neatly execute asynchronous functions. It acts as a wrapper to execute asynchronous functions with a callback by providing a nicer syntax

```
var func = function(resolve, reject){
  setTimeout(function(){
    console.log('in setTimeout callback');
    resolve('Timed out for five seconds');
  }, 5000); // setTimeout for 5 seconds
}
var callbackfn = function(value){
  console.log('In callbackfn and printing value:');
  console.log(value); // print the value received
}
var promise = new Promise(func); // create Promise with func
promise.then(callbackfn); // run promise with callback in then method
```

Output

```
in setTimeout callback
In callbackfn and printing value:
Timed out for five seconds
```

Promises

❑ Chaining promises

The tasks are done in a sequential chain. The first func function is invoked and passes the value passed to resolve function to next callback function.

```
var func = function(resolve,reject){
  console.log('in func');
  resolve(10);
}
var syncTask1 = function(val){ // first then method task
  console.log('in task 1 with val:',val);
  return val + 1;
}
var syncTask2 = function(val){ // second then method task
  console.log('in task 2 with val:',val);
  return val + 1;
}

var promise = new Promise(func); // create promise with func function
promise // call promise by adding .then method to it
.then(syncTask1) // invoke syncTask1 with value passed to resolve
.then(syncTask2) // invoke syncTask2 with value retured by syncTask1
.then(val => { // adding arrow function directly
  console.log('End of the chain with val:', val);
})
```

Output

```
in func
in task 1 with val: 10
in task 2 with val: 11
End of the chain with val: 12
```

Await/Async

❑ Introduction to async

async token lets you declare a special type of function that always returns a promise.

Let's declare an async function

```
var func = async function(){
    // Add tasks async or sync
}
```

```
var func = async () => {
    // Add tasks async or sync
}
```

These functions always return a promise. See that in the code below.

```
var func1 = async function (){
  return 1;
}
console.log(func1); // async function
console.log(func1()); // promise
func1().then(val => { // invoke promise
  console.log(val); // get value
})
```

Output

```
[AsyncFunction: func1]
Promise { 1 }
1
```

Await/Async

❑ Introduction to await

The await keyword is used only inside the async function

```
await somePromise;
```

- Getting value returned by the promise

```
var someVar = await somePromise;
```

Check out an example below

```
var func = async function (){
  let promise1 = new Promise((resolve, reject)=>{
    setTimeout(()=>{
      console.log('timeout1 for 3 seconds');
      resolve('done');
    }, 3000);
  })

  await promise1;
  console.log('promise compelete');
}

func();
```

Output

```
timeout1 for 3 seconds
promise compelete
```

Error Handling

❑ Introduction to try and catch statements

JavaScript error handling with asynchronous and synchronous programming

Syntax

```
try {
  // execute statements
} catch (err){
  // handle error err received from try block
}
```

Learn from the example below:

```
try{
  console.log('entered try block');
  undefinedFunc(); // invoke undefined func
  console.log('invoked func in try block'); // will not be executed
} catch (err){
  console.log("Err:", err);
}
```

Output

```
entered try block
Err: ReferenceError: undefinedFunc is not defined
    at Object.<anonymous> (/usercode/index.js:3:3)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
```

Error Handling

❑ Introduction to try and catch statements

JavaScript error handling with asynchronous and synchronous programming

```
try {
  throw 'Will not execute anything';
  console.log('entered try block');
  undefinedFunc(); // invoke undefined func
  console.log('invoked func in try block'); // will not be executed
} catch (err) {
  console.log('entered catch block');
  console.log("Err:", err);
  console.log('Printed error');
}
```

Output

```
entered catch block
Err: Will not execute anything
Printed error
```

Good luck 👌

References

1. https://www.educative.io/ (JavaScript in Detail: From Beginner to Advanced)
2. https://www.w3schools.com/js/js_htmldom_document.asp