PART 3

LAB SESSION

# ✓ Async Programming

# Lab 9a: Synchronous Programming

The purpose of this lab is to illustrate the effects of calling synchronous function.

1. Create the file 9a-async-wait.js

2. Create a function called wait() which does nothing in a while loop until a certain amount of time in milliseconds has passed. This function is synchronous because execution cannot continue until the while loop is exited.

```
// Wait until `ms` milliseconds of time has passed. Date.now() returns
// number of milliseconds elapsed since Jan 1, 1970 00:00:00 UTC
const wait = ms => {
    const end = Date.now() + ms
    while (Date.now() < end) continue
}
```

# Lab 9a: Synchronous Programming

3. Create the functions f1, f2 and f3 which simulate long running processes that completes in 2sec, 4sec and 2sec respectively.

```
// f1, f2 and f3 are long running processes
const f1 = ()=> { wait(2000); console.log("f1"); }
const f2 = ()=> { wait(4000); console.log("f2"); }
const f3 = ()=> { wait(2000); console.log("f3"); }
```

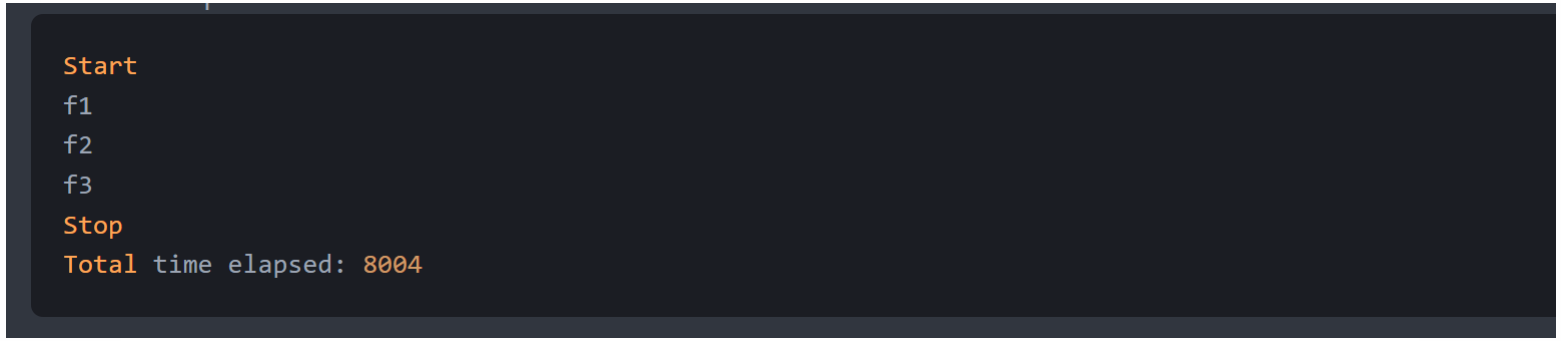
# Lab 9a: Synchronous Programming

4. Run the functions in sequence and track the total elapsed time.

```
// Run f1, f2 and f3 in sequence and show elapsed time
const timeStart = Date.now();
console.log(`Start`);
f1();
f2();
f3();
const timeEnd = Date.now();
console.log(`Stop`);
console.log(`Total time elapsed: ${timeEnd - timeStart}`);
```

# Lab 9a: Synchronous Programming

The output will look like this:




f1(),f2() and f3() are executed in sequence and total execution time will take 8 sec.

# Lab 9b: Asynchronous Programming

- The purpose of this lab is to illustrates the effects of calling asynchronous function.

- In this lab, we will replace the use of synchronous wait() with the Javascript build-in setTimeout() function. This is the asynchronous version of the wait() function that we have created in the earlier example. setTimeout() takes in 2 parameters - a callback function and a duration in milliseconds.

# Lab 9b: Asynchronous Programming

1. Copy the earlier file into the file 9b-async-settimeout.js
2. Replace wait() with setTimeout() in f2 with a callback function ()=>console.log('f2') with a duration of 4000 milliseconds.

```
const f1 = ()=> {wait(2000);console.log("f1");}
const f2 = ()=> setTimeout(()=>console.log("f2"),4000);
const f3 = ()=> {wait(2000);console.log("f3");}
```

# Lab 9b: Asynchronous Programming

3. Run the script and the output is shown below.

```
Start
f1
f3
Stop
Total time elapsed: 4003
f2
```

4. Notice that the output from this example are out of sequence; f2 is displayed at the end instead of after f1. The total time elapsed is reduced to 4 seconds from of 8 seconds.

# Lab 9c: Running asynchronous functions in sequence

The purpose of this lab is to understand how to execute asynchronous functions synchronously. For this lab, please create 3 asynchronous functions but the functions must be executed in sequence.

1. Create the file `9c-async-callback.js` and create the following on your own.
2. Create function f1 that waits asynchronously for 2 sec before displaying 'f1' and calls f2.
3. Create f2 that waits asynchronously for 4 sec before displaying 'f2' and calls f3.
4. Create f3 that waits asynchronously for 2 sec before displaying 'f3' and output the total elapsed time.
5. Run the functions and track the total elapsed time.
   - If the code is written correctly, the output will be:

```
Start
f1
f2
f3
Stop
Total time elapsed: 8000
```

NOTE: The total time will not be 8000 exactly due to slippage caused by running settimeout itself.

# Lab 9c: Running asynchronous functions in sequence

**ANSWER**

```
const f1 = (f2) =>
    setTimeout(() => {
        console.log("f1");
        f2();
    }, 2000);

const f2 = (f3) =>
    setTimeout(() => {
        console.log("f2");
        f3();
    }, 4000);

const f3 = () =>
    setTimeout(() => {
        console.log("f3");
        const timeEnd = Date.now();
        console.log(`Total time elapsed: ${timeEnd - timeStart}`);
    }, 2000);

console.log("Start");
const timeStart = Date.now();

f1(() => f2(f3));

console.log(`Stop`);
```

# Lab 9d: Asynchronous programming with Promise

The purpose of this lab is to illustrate how to use asynchronous functions with Promise instead of callbacks.

In previous lab, we used setTimeout and callback to wait asynchronously. Because setTimeout is a legacy Javascript function, there is no equivalent of a function that returns a Promise. In this lab, we will make a Promise out of setTimeout().

1. Create the file 9d-async-promise.js

2. Create the promiseToWait() function using the code below. This function will create a promise object using the Promise constructor.

```
const promiseToWait = ms => new Promise(resolve => setTimeout(resolve, ms));
```

# Lab 9d: Asynchronous programming with Promise (Cont'd)

3. Replace f1, f2 and f3 with a function that creates a new Promise and replace wait with promiseToWait.

```
const f1 = () =>
    new Promise((resolve) => {
        promiseToWait(2000)
            .then(() => console.log("f1"))
            .then(resolve);
    });

const f2 = () =>
    new Promise((resolve) => {
        promiseToWait(4000)
            .then(() => console.log("f2"))
            .then(resolve);
    });

const f3 = () =>
    new Promise((resolve) => {
        promiseToWait(2000)
            .then(() => console.log("f3"))
            .then(resolve);
    });
```

# Lab 9d: Asynchronous programming with Promise (Cont'd)

4. Run the functions in sequence and track the total elapsed time.

```
console.log(`Start`);
const timeStart = Date.now();
f1()
    .then(f2)
    .then(f3)
    .then(() => {
        const timeEnd = Date.now();
        console.log(`Total time elapsed: ${timeEnd - timeStart}`);
    });
console.log(`Stop`);
```

# Lab 9d: Asynchronous programming with Promise (Cont'd)

The output will look like this:

```
Start
Stop
f1
f2
f3
Total time elapsed: 8012
```