Week 2 Exploration

- Zainab Nusaiba

Problem Statement:

```
setTimeout(() => { console.log("Hello world!"); }, 2000);

for(var i=0; i < 10; i++) {
   runIt(i);
}

function runIt(i) {
   setTimeout(function(){
     console.log("In-loop "+(i+1)+" Sec");
   }, i * 1000);
}
```

Objective:

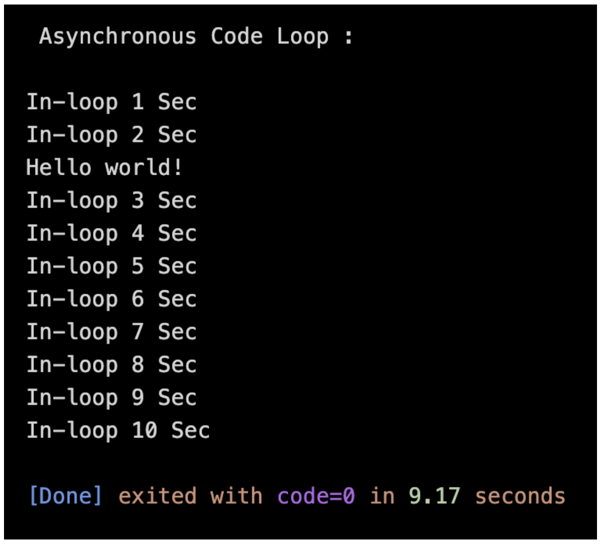
- Document the outcome of the above function.
- Are the sequences of events synchronous? If so, how would you make it asynchronous or vice-versa?

Outcome:

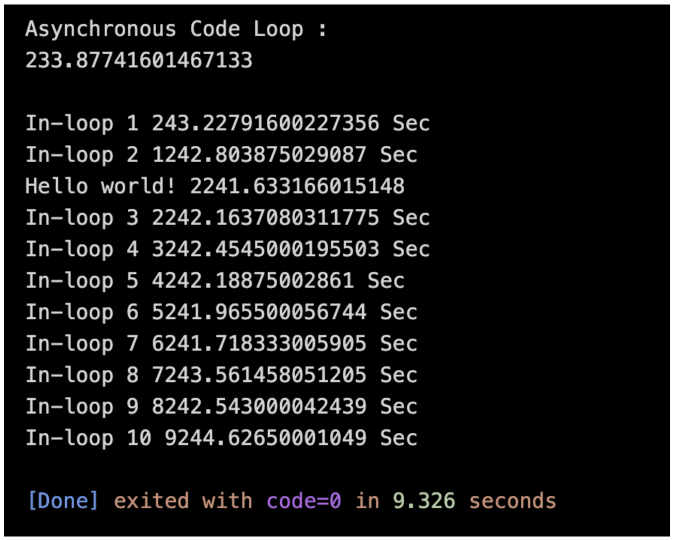
The output of the above program is given below. Some of the functionings involved in the program are :

setTimeout()

An asynchronous method executes the given data or anonymous function after the timer. This shows that it consists of two parameters the anonymous function and time represented in milliseconds
Eg: setTimeout (greet() , 2000)



```
 Asynchronous Code Loop :

In-loop 1 Sec
In-loop 2 Sec
Hello world!
In-loop 3 Sec
In-loop 4 Sec
In-loop 5 Sec
In-loop 6 Sec
In-loop 7 Sec
In-loop 8 Sec
In-loop 9 Sec
In-loop 10 Sec

[Done] exited with code=0 in 9.17 seconds
```



```
Asynchronous Code Loop :
233.87741601467133

In-loop 1 243.22791600227356 Sec
In-loop 2 1242.803875029087 Sec
Hello world! 2241.633166015148
In-loop 3 2242.1637080311775 Sec
In-loop 4 3242.4545000195503 Sec
In-loop 5 4242.18875002861 Sec
In-loop 6 5241.965500056744 Sec
In-loop 7 6241.718333005905 Sec
In-loop 8 7243.561458051205 Sec
In-loop 9 8242.543000042439 Sec
In-loop 10 9244.62650001049 Sec

[Done] exited with code=0 in 9.326 seconds
```

Are they Synchronous:

No, the sequences of events are Asynchronous.

Making it Synchronous:

It is asynchronous because setTimeout() is an asynchronous method. It also interrupts the program between the loop calls. And, setTimeout() can be written synchronously by using

1. Callback

This is done by passing a function into a timeout as an argument then the function will be executed after the timeout is executed.

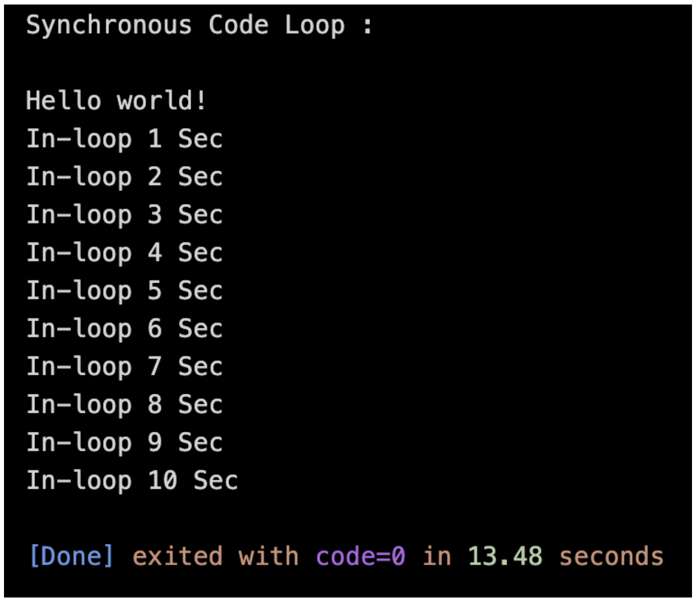
2. async, await, and promise

Await is used along with the async function. Async makes the function asynchronous and when paired with the await, it waits for the acknowledgment and then executes after that. Promises are the microtasks that can be called. It is usually executed after all the normal callbacks in the program.

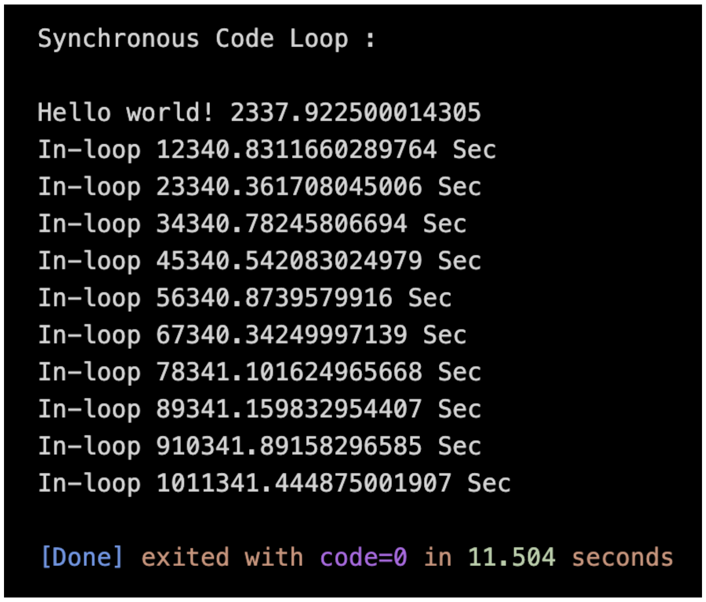
Synchronous Code:

```
console.log("\nSynchronous Code Loop :\n");

function helloMssg() {
 return new Promise((res, rej) => {
   setTimeout(() => {
     res("Hello world!");
     rej("Rejected");
   }, 2000);
 });
}

async function loopMssg() {
 await helloMssg()
   .then((success) => {
     console.log(success);
   })
   .catch((error) => {
     console.log(error);
   });

 for (var i = 0; i < 10; i++) {
   runIt(i);
 }

 function runIt(i) {
   setTimeout(function () {
     console.log("In-loop " + (i + 1) + " Sec");
   }, i * 1000);
 }
}
loopMssg();
```

Synchronous Outcome:



```
Synchronous Code Loop :

Hello world!
In-loop 1 Sec
In-loop 2 Sec
In-loop 3 Sec
In-loop 4 Sec
In-loop 5 Sec
In-loop 6 Sec
In-loop 7 Sec
In-loop 8 Sec
In-loop 9 Sec
In-loop 10 Sec

[Done] exited with code=0 in 13.48 seconds
```



```
Synchronous Code Loop :

Hello world! 2337.922500014305
In-loop 12340.8311660289764 Sec
In-loop 23340.361708045006 Sec
In-loop 34340.78245806694 Sec
In-loop 45340.542083024979 Sec
In-loop 56340.8739579916 Sec
In-loop 67340.34249997139 Sec
In-loop 78341.101624965668 Sec
In-loop 89341.159832954407 Sec
In-loop 910341.89158296585 Sec
In-loop 1011341.444875001907 Sec

[Done] exited with code=0 in 11.504 seconds
```