Date : 11-6-2020
Morning Session : 9am – 11 PM
By ~ Sundeep Charan Ramkumar Today

Topics: Intro To Async Programming

Synchronous vs Asynchronous

Allowing a code to run in a separate way such that succeeding code will execute.

https://community.tealiumiq.com/t5/iQ-Tag-Management/Synchronous-vs-Asynchronous-JavaScript-sync-vs-async/ta-p/13490

We always use Asynchronous programming in 3 operations

1) i/o Operations (file read and file ride)
2) Network Request(HTTP)
3) Stream Data

Types of Asynchronous:

1) Callback
2) Promises

Callback:

setTimeout: sets a timer which executes a function or specified piece of code once the timer expires.

```
// setTimeout - Event Loop
setTimeout(function () {
  console.log("Hello world");
}, 2000);
```

setInterval:

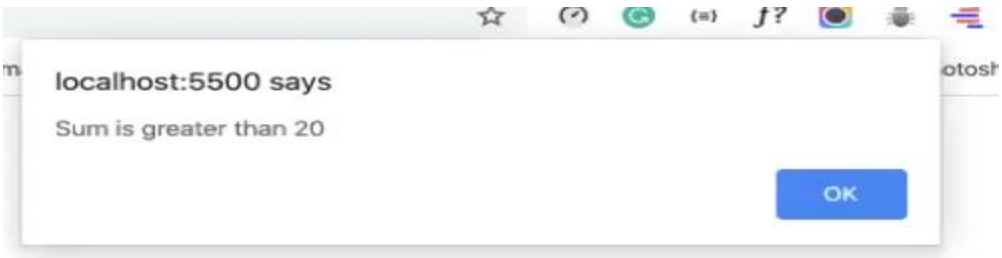
repeatedly calls a function or executes a code snippet, with a fixed time delay between each call

```
// setInterval - Event loop
setInterval(function () {
  console.log("I am getting repeated");
}, 2000);
```

Problem Statement:

```
function addNumbersSlowly(callbackFunction, a, b) {
  setTimeout(function () {
    var sum = a + b;
    if (sum > 20) {
      callbackFunction("Sum is greater than 20", null);
    } else {
      callbackFunction(null, sum);
    }
  }, 5000);
}

addNumbersSlowly(
  function (err, result) {
    if (err !== null) alert(err);
    else console.log(`Process is successful. Here is the result ${result}`);
  },
  10,
  20
);
```



```
function addNumbersSlowly(callbackFunction, a, b) {
  setTimeout(function () {
    var sum = a + b;
    if (sum > 20) {
      callbackFunction("Sum is greater than 20", null);
    } else {
      callbackFunction(null, sum);
    }
  }, 5000);
}

addNumbersSlowly(
  function (err, result) {
    if (err !== null) alert(err);
    else console.log(`Process is successful. Here is the result ${result}`);
  },
  10,
  5
);
```

```
rocess is successful. Here app.js:31
is the result 15
>
```

Promises: **which allows you to have either response or error.**

Promises Consists 3 states 1) pending, 2) resolved, 3) rejected

1) **Creating promise**

   Syntax:

New

```
new Promise(fn(res, rej)
  {
    // res("      ")
    // rej("      ")
  })
```

```
function addNumbersSlowly(callbackFunction, a, b) {
  setTimeout(function () {
    var sum = a + b;
    if (sum > 20) {
      callbackFunction("Sum is greater than 20", null);
    } else {
      callbackFunction(null, sum);
    }
  }, 5000);
}

addNumbersSlowly(
  function (err, result) {
    if (err !== null) alert(err);
    else console.log(`Process is successful. Here is the result ${result}`);
  },
  10,
  5
);
```

```
function addNumbersPromise(a, b) {
  return new Promise(function (resolveFunction, rejectFunction) {
    setTimeout(function () {
      var sum = a + b;
      if (sum > 20) {
        // Error.
        rejectFunction("Sum is greater than 20");
      } else {
        resolveFunction(`The Sum is ${sum}`);
      }
    }, 3000);
  });
}

var sum = addNumbersPromise(1, 4);
console.log(sum);
```

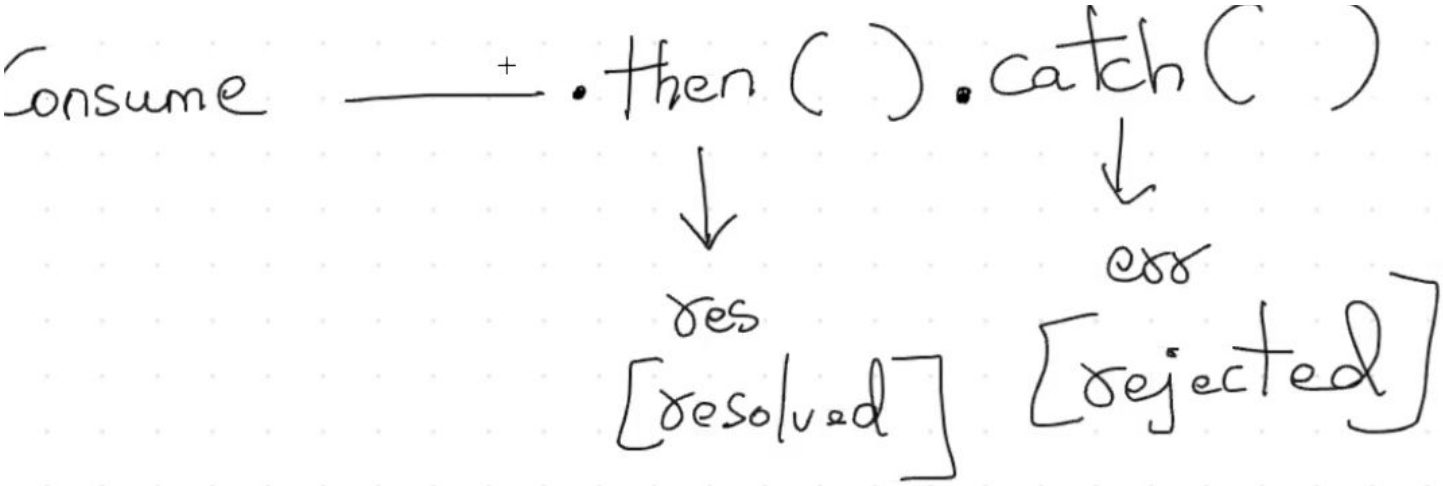
It will show pedding.

```
                                        app.js:54
▼ Promise {<pending>}
  ▶ __proto__: Promise
    [[PromiseStatus]]: "resolved"
    [[PromiseValue]]: "The Sum is 5"
Process ended                           app.js:56
15                                      app.js:33
```

2) **Consuming promise**

Syntax:

Consume Attach promise . then() . Catch()




```
addNumbersPromise(10, 20)
  .then(function (result) {
    console.log(result);
  })
  .catch(function (error) {
    alert(`Error: ${error}`);
  });

console.log("Process ended");
```

MCQ1 :

In asynchronous programming, we are bound to wait for a huge task and then move on to the next line. True or false?

2

- [ ] True
- [x] False 52.17%

MCQ2 :

What is callback hell? 2

- [x] Having multiple functions requiring the need to wait for the previous function's execution 72.46%
- [ ] Linear execution of functions without having the need to wait for a result.

MCQ3 :

Which of the following is an invalid Promise state Not Submitted

- [x] accepted 81.16%
- [ ] resolved
- [ ] rejected
- [ ] pending

MCQ4:

How do we create a promise? 2

- [x] By using the new keyword 88.41%
- [ ] By returning the promise itself.