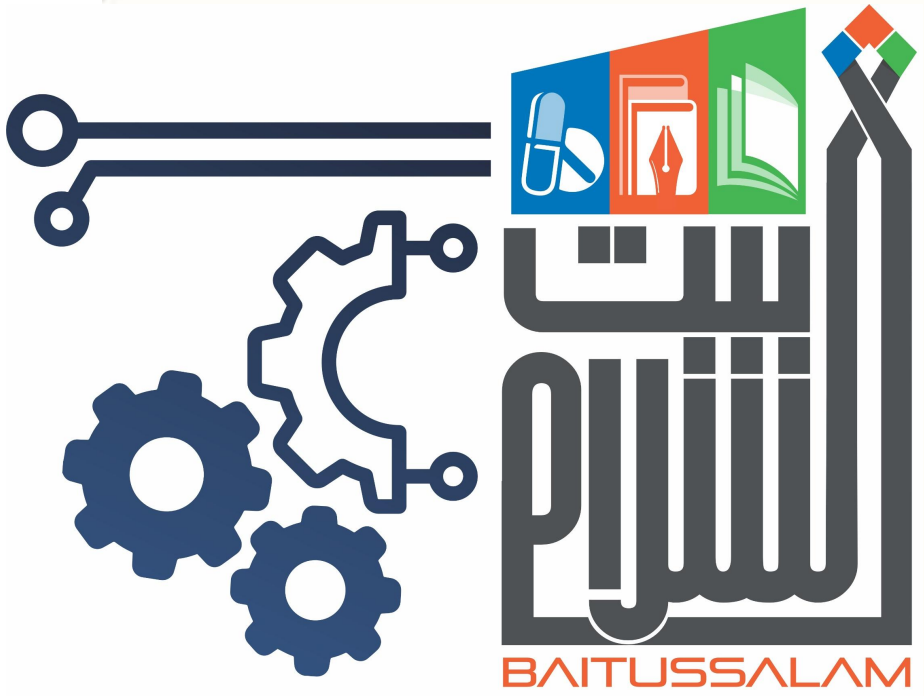

BAITUSSALAM
—TECH PARK—

Class Agenda

BAITUSSALAM
—TECH PARK—

JavaScript Asynchronous Programming, Callback hell, Async Await, Fetching API Data

Introduction

What is asynchronous programming?

Asynchronous programming in JavaScript is a way to handle tasks that take time without blocking the main program flow. It allows your program to start a long-running task and continue executing other code while the task is in progress.

Understanding the Concept

```
1  // Define three functions
2  function firstTask() {
3    console.log('Task 1')
4  }
5
6  function secondTask() {
7    setTimeout(() => {
8      console.log('Task 2')
9    }, 2000)
10 }
11
12 function thirdTask() {
13   console.log('Task 3')
14 }
15
16 // Execute the functions
17 firstTask()
18 secondTask()
19 thirdTask()
```

What will be the output of the following code

Asynchronous Code with Callback

In code, a callback function is a function that is passed as an argument to another function, and it is executed after the first function has finished running. It's commonly used in JavaScript to handle asynchronous operations like fetching data from a server, waiting for a user's input, or handling events.

Asynchronous Code Exercise #1

```
43  let data
44
45  function fetchData() {
46    setTimeout(() => {
47      data = { name: 'John', age: 30 }
48    }, 2000)
49  }
50
51  console.log(data)
52
53  fetchData()
54
55  console.log('Data is being fetched...')
```

Fix this code using callback

Asynchronous Code Exercise #2

```
1  function orderPizza() {
2    setTimeout(() => {
3      console.log('Step 1, ordered')
4    }, 4000)
5  }
6
7  function preparingPizza() {
8    setTimeout(() => {
9      console.log('step 2, prepared')
10   }, 3000)
11 }
12
13 function served() {
14   setTimeout(() => {
15     console.log('step 3, served')
16   }, 1000)
17 }
18
19 orderPizza()
20 preparingPizza()
21 served()
```

Fix the bug, and run the code in order, use callbacks

Step 1
Step 2
Step 3

JavaScript Promises

A promise represents a way of handling asynchronous operations in a more organized way.

It serves the same purpose as a callback but offers many additional capabilities and a more readable syntax.

How Promise Works

```
const promise = new Promise((resolve, reject) => {
  let success = true

  if (success) {
    resolve('Operation was successful!')
  } else {
    reject('Operation failed!')
  }
})
```

A promise has three states:

Pending: initial state, neither fulfilled nor rejected.
Fulfilled: meaning that an operation was completed successfully.
Rejected: meaning that an operation failed.

Using Promises

```
13  promise
14      .then((message) => console.log(message))
15      .catch((error) => console.log(error))
```

then(method)
then() method is used with callback when promises is resolved/fulfilled

catch method()
catch() method is used with the callback when the promise is rejected or error occurs

Finally() method
finally() method gets executed when the promise is either resolve or rejected

Promise Exercise

```
function orderPizza() {
  setTimeout(() => {
    console.log('Step 1, ordered')
  }, 4000)
}

function preparingPizza() {
  setTimeout(() => {
    console.log('step 2, prepared')
  }, 3000)
}

function served() {
  setTimeout(() => {
    console.log('step 3, served')
  }, 1000)
}

orderPizza()
preparingPizza()
served()
```

Fix the bug/code,
and run the code in order,
use promises

Step 1
Step 2
Step 3

Async Await

Async/Await: Simplifies working with promises and makes asynchronous code look like synchronous code.

Why Use Async and Await?

Readability: Makes asynchronous code easier to read and write.
Error Handling: Simplifies error handling with try/catch.
Synchronous Flow: Code executes in a top-down manner, improving flow control.

Async Await Exercise

```
1  function orderPizza() {
2    setTimeout(() => {
3      console.log('Step 1, ordered')
4    }, 4000)
5  }
6
7  function preparingPizza() {
8    setTimeout(() => {
9      console.log('step 2, prepared')
10   }, 3000)
11 }
12
13 function served() {
14   setTimeout(() => {
15     console.log('step 3, served')
16   }, 1000)
17 }
18
19 orderPizza()
20 preparingPizza()
21 served()
```

Run the code in order,
using async and await

Step 1
Step 2
Step 3

Fetch

Purpose: Allows fetching resources asynchronously across the network.

Replacement: Modern alternative to XMLHttpRequest (XHR).

```
fetch(url)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

How to use Fetch

BAITUSSALAM
—TECH PARK—

With then method

```
function fetchData() {
  fetch('https://jsonplaceholder.typicode.com/posts')
    .then((res) => res.json())
    .then((data) => console.log(data))
}
```

With async await

```
async function fetchData() {
  try {
    let response = await fetch('https://jsonplaceholder.typicode.com/posts')
    let data = await response.json()
    console.log(data)
  } catch (error) {
    console.error('Error:', error)
  }
}
```

Assignment: Json Server For Mocking API

BAITUSSALAM
—TECH PARK—

Learn and understand how json server works for mocking API data and performing CRUD operation on it

API Project #2

BAITUSSALAM
—TECH PARK—

Quiz App

API Project #2

BAITUSSALAM
—TECH PARK—

Weather App

The End

BAITUSSALAM
—TECH PARK—