

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

الحمد لله رب العالمين ، نحمده ونستعينه
ونستغفره والصلاة والسلام على أشرف
الأنبياء والمرسلين نبينا محمد ﷺ



پوهنتون کاردان
KARDAN UNIVERSITY

Asynchronous JavaScript

Lecture 5

Khalilullah Akbari
k.akbari@kardan.edu.af

What is Asynchronous JavaScript?

Definition: Non-blocking programming paradigm allowing tasks to run in the background while the main thread continues execution.

Importance: Improves user experience by preventing UI freezing.



Synchronous vs Asynchronous

Synchronous: Tasks are executed sequentially, blocking subsequent operations.

Asynchronous: Tasks run independently, enabling concurrent operations.

Example Code:

```
console.log('Task 1');  
setTimeout(() => console.log('Task 2'), 1000);  
console.log('Task 3');
```



What is a Promise?

Definition:

- A **Promise** is a **JavaScript object** that represents the **result of an asynchronous operation**, which may **complete in the future**—either **successfully** or with an **error**.

States:

- Pending
- Fulfilled
- Rejected



Creating Promises

Example Code:

```
const myPromise = new Promise((resolve, reject) => {  
  let success = true;  
  if (success) {  
    resolve('Operation Successful!');  
  } else {  
    reject('Operation Failed.');  }  
});
```

Think of a Promise like an empty box:

- ❖ JavaScript says: “Hey, I’ll give you something later (like data from a server), but not right now.”
- ❖ You can use `.then()` to get the value when it’s ready.
- ❖ You can use `.catch()` to handle errors if something goes wrong.



Handling Promises

Methods:

- `.then()`: Handles fulfillment.
- `.catch()`: Handles rejection.
- `.finally()`: Executes regardless of outcome.

Example Code:

`myPromise`

```
.then(response => console.log(response))  
.catch(error => console.error(error))  
.finally(() => console.log('Done.'));
```



Async/Await

Definition: Syntactic sugar for promises, making asynchronous code look synchronous.



Writing Async Functions

Example Code:

```
async function fetchData() {  
  try {  
    let response = await fetch('https://api.example.com/data');  
    let data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error(error);  
  }  
}
```



What is Fetch API?

Definition: Modern interface for making HTTP requests.

Replaces: XMLHttpRequest.



Fetch with Async/Await

Example Code:

```
async function getData() {  
  const response = await fetch('https://api.example.com/data');  
  const data = await response.json();  
  console.log(data);  
}  
getData();
```



POST Requests with Fetch

Example Code:

```
fetch('https://api.example.com/data', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify({ key: 'value' })  
})  
  .then(response => response.json())  
  .then(data => console.log(data))  
  .catch(error => console.error(error));
```



Error Handling in Fetch

Example Code:

```
fetch('https://api.example.com/data')  
  .then(response => {  
    if (!response.ok) {  
      throw new Error('Network response was not ok');  
    }  
    return response.json();  
  })  
  .catch(error => console.error('Fetch error:', error));
```



Comparison: Promises vs Async/Await

Feature	Promises	Async/Await
Readability	Moderate	High
Error Handling	Requires .catch()	Try/Catch
Complexity	Moderate	Low



Destructuring

Extracting data from array or object

```
let arr = ["ali","ahmad"]  
let [name1,name2] = arr  
  
console.log(name1)      'ali'  
console.log( name2)     'ahmad'
```



Tools for Async Operations



Axios

Lodash

Bluebird

