

DOM + Modern JS - 4

* What is an API ?

API stands for Application Programming Interface. It is a particular set of rules and specifications that software program can follow to communicate with each other.

It serves as an interface between different software programs and facilitates their interactions, similar to the way the user interface facilitates interaction between humans and computers.

Methods are:- GET, POST, PUT, HEAD, DELETE, PATCH, CONNECT, TRACE etc.

* Features of Async-Code

- Clean & Concise
- Better Error handling
- Easier debugging

PROMISE

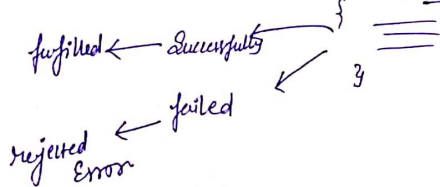
- Promise is used for parallel execution in the background of Javascript.
- A promise is a proxy for a value not necessarily known when the promise is created.
- It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

* A promise has three states:-

- pending :- initial state, neither fulfilled nor rejected.
- fulfilled :- completed successfully. (fulfilled with value)
- rejected :- operation failed. (rejected with error)

* We use Promise() constructor for creating a new promise object.

* Syntax :-
let p = new Promise (callback function());



Creating a Promise :-

```
let meraPromise = new Promise (function (resolve, reject) {  
  setTimeout (function () {  
    console.log ("I am inside Promise");  
  }, 5000);  
  
  resolve (2023); // Resolve with any values.  
  // reject (new Error ('Error Generated ---'));  
});
```

Output:-

I am inside Promise → (after 5 seconds)

console:-

```
meraPromise;  
Promise {fulfilled}: 2023
```

In case of Rejected :-

```
reject (new Error ('Error Generated ---'))
```

Output:-

Error :- 'Error Generated'

console:-

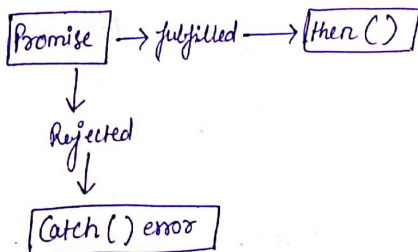
```
meraPromise;  
Promise {rejected}: Error: 'Error Generated ---'
```

parallel execution of Promise :-

```
let promise-1 = new Promise(function(resolve, reject){  
  setTimeout(function(){  
    console.log("I am inside Promise 1");  
  }, 5000);  
});
```

```
let promise-2 = new Promise(function(resolve, reject){  
  setTimeout(function(){  
    console.log("I am inside Promise 2");  
  }, 3000);  
});
```

Promise :- then() and catch()



Example of then() method

① let promise-1 = new Promise(function(resolve, reject){
 resolve(12345678);
});

promise-1.then(value) => {console.log(value)};

Output:-

12345678

explanation:- Resolve is handled by then() method here.

② Example of catch() method

let promise-2 = new Promise(function(resolve, reject) =>
 reject(new Error("Error Generated")));

promise-2.catch(error) => {console.log("Receive an Error")};

Output:-

Receive an Error

explanation:- Reject is handled by catch() method here.

Chaining Promise

promise-1.then(value) => {____}, (error) => {____};

* Promise Parallel execution inside then().

```
let P1 = new Promise (function (resolve, reject) {  
  setTimeout(() => {  
    console.log('SetTime 1 Started');  
  }, 2000);  
  resolve (true);  
})
```

2nd hi 2 sec at print item

```
let Output = P1.then (() => {  
  let P2 = new Promise (function (resolve, reject) {  
    setTimeout (() => {  
      console.log('SetTime 2 Started');  
    }, 3000);  
    resolve ("P2 resolved");  
  })  
  return P2;  
})
```

3rd hi 3 sec at print

4th hi 4 sec Print

Output. then (value) => console.log (value);

Output :-
P2 resolved
SetTime 1 Started
SetTime 2 Started

Async & Await keyword (Avoid Multiple then())

* "async and await" make promises easier to write

→ async :- keyword async place before a function makes the function return a promise.

Example :-

```
async function ABCD () {  
  console.log ("Hello Async");  
  return 2023;  
}  
console.log (ABCD());
```

Any Value resolve
or any string

Output :-
Hello Async
Promise {<fulfilled>: 2023} → async return a promise here.

→ await :- 'await' keyword can only be used inside an async function.

It makes the function pause the execution and await for a resolved promise before it continues.

~~ex. :- async function ABCD () {
 return "completed";
}~~

P.T.O

example of await :-

```
async function Utility() {  
  let delhi = new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve("Delhi is Capital"); }, 1000);  
    });
```

```
  let mumbai = new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve("Mumbai is Finance Capital"); }, 3000);  
    });
```

```
  let d = delhi await delhi; // first completely execute this.  
  let m = await mumbai; // after 'd' completion,
```

```
  return [d, m];
```

```
}
```

Fetch API :-

* The fetch api provides a JS interface for accessing and manipulating parts of the protocol such as requests and responses.

* It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.

→ fetch() method starts the process of fetching a resource from a server.

→ Fetch() method returns a Promise that resolves to a response object.

→ List of some freely available API :-

- 7 timer! (weather forecasts)
- Dogs (Random dog images)
- Joke API (Jokes)
- JSONPlaceholder (Fake Rest API for testing)

!
}
etc.

Retrieving data through fetch api (Get Call) using API

```
async function utility() {  
  let content = await fetch('https://jsonplaceholder.  
    tyipicode.com/posts/1');  
  let output = await content.json();  
  console.log(output);  
}  
utility();
```

→ converting content into json format

Output:-

```
{  
  userId: 1,  
  id: 1,  
  title: 'sunt aut facere -----',  
  body: 'quid -----' }
```

We can check

- ✓ content.status
- ✓ content.ok
- ✓ content.json()
- ✓ content.text()
- etc.

JSON :- JavaScript Object Notation

JSON :- JavaScript Object Notation

→ JSON is a lightweight format for storing and transporting data.

→ JSON is often used when data is sent from a server to a web page.

JSON Syntax Rule :-

- * key value pairs
- * separated by comma
- * within {} hold objects
- * [] holds array of objects.

Example :-

```
{  
  "employees": [  
    { "firstname": "John", "lastname": "Doe" },  
    { "firstname": "Peter", "lastname": "Jones" }  
  ]  
}
```

Send Data through Fetch Api (POST Calls)

fetch('url') → Get ✓

Now,

fetch('url', options) → POST

Object for authenticity / POST method.

```
async function helper() {  
  let options = {  
    method: 'POST',  
    body: JSON.stringify({  
      title: 'foo',  
      body: 'bar',  
      userId: 91, weight: 100,  
    }),  
    headers: {  
      'Content-type': 'application/json',  
      charset: 'UTF-8',  
    },  
  };  
  
  let content = await fetch('https://jsonplaceholder.typicode.com/posts',  
    options);  
  let response = content.json();  
  return response;  
}
```

POST calls available on Free API

```
async function utility() {  
  let ans = helper();  
  console.log(ans);  
}
```

Output:- { title: 'foo', body: 'bar', userId: 91, weight: 100, id: 101 }

Note:- Stringify() :- convert JS object into JSON string.

CLOSURES

A closure is the combination of a function bundled together with references to its surrounding state (the lexical environment).

In other words, A closure gives you access to an outer function's scope from an inner function.

In JavaScript, closures are created every time a function is created, at function creation time.

Problem in Normal Case

```
function init() {  
  var name = "Mozilla";  
  function displayName() {  
    console.log(name);  
  }  
  displayName();  
}
```

// local variable

```
init();
```

Note:- Variable name cannot be accessed outside `init()` function. Outside the `init()`, variable name will be dead.

But Closures make it accessible with the references.

```
function makeFunc() {  
  const name = "Mozilla";  
  function displayName() {  
    console.log(name);  
  }  
  return displayName;  
}
```

// name is surrounding

// here making closure function

```
const myFunc = makeFunc();  
myFunc();
```

// myFunc is reference

In this case, As functions in JS forms closures.

* myFunc is a reference to the instance of the function displayName() that is created when makeFunc() is run.

* The instance of displayName() maintains a reference to its lexical environment, within the variable name exists.