**Module :12 JavaScript ES 6 Object Oriented Concept**

**● What is JSON**

Ans:- JSON stands for 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 is "self-describing" and easy to understand

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

{"firstName":"John", "lastName":"Doe"}

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated using a string as input.

First, create a JavaScript string containing JSON syntax:

"employees":[  
    {"firstName":"John", "lastName":"Doe"},  
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]

**● What is promises**

Ans:- The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

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. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

A Promise is in one of these states:

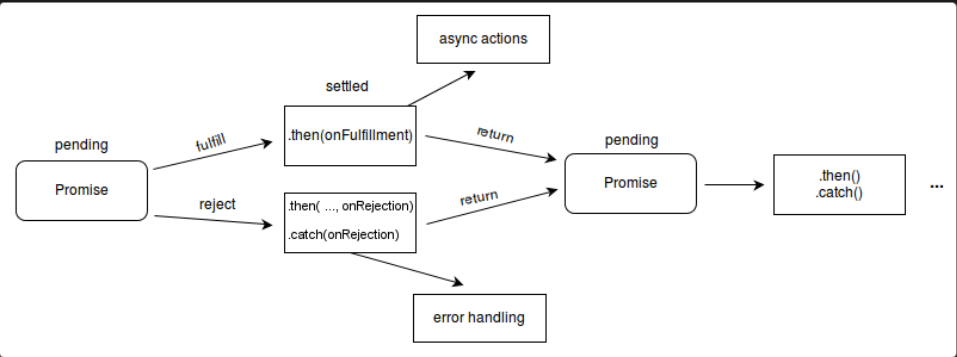
pending: initial state, neither fulfilled nor rejected.

fulfilled: meaning that the operation was completed successfully.

rejected: meaning that the operation failed.

The eventual state of a pending promise can either be fulfilled with a value or rejected with a reason (error). When either of these options occur, the associated handlers queued up by a promise's then method are called. If the promise has already been fulfilled or rejected when a corresponding handler is attached, the handler will be called, so there is no race condition between an asynchronous operation completing and its handlers being attached.

A promise is said to be settled if it is either fulfilled or rejected, but not pending.



**● Write a program of promises and handle that promises also**

Ans:- fetch('https://fakestoreapi.com/products')

.then(res=>res.json())

.then(json=>console.log(json))

.catch(err => console.log(err);

**● Use fetch method for calling an api** [**https://fakestoreapi.com/products**](https://fakestoreapi.com/products)

Ans :-

**● Display all the product from the api in your HTML page**

Ans:- <!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

</head>

<body>

<div class="container">

<div class="text-center" id="spn">

<div class="spinner-border" role="status">

<span class="visually-hidden">Loading...</span>

</div>

</div>

<div class="row" id="btn"></div>

</div>

<script>

let spn = document.getElementById('spn')

fetch("https://fakestoreapi.com/products")

.then(response => {

console.log(response)

return response.json();

}).then(data => {

console.log(data);

showdata(data);

spn.style.display = "none"

}).catch(err => {

return console.log(err)

})

let btn = document.getElementById('btn')

function showdata(data) {

for (const mydata of data) {

let htmldata = `<div class="card col-md-3">

<img src="${mydata.image}" class="card-img-top" alt="...">

<div class="card-body">

<h5 class="card-title">${mydata.title}</h5>

</div>

</div>`

btn.insertAdjacentHTML('beforeend', htmldata);

}

}

</script>

</body>

</html>