1. What is Node.js?

- Node.js is an open-source platform that lets you run JavaScript on the server, not just in web browsers.
- It's built on the **V8 JavaScript engine** (used by Google Chrome) and helps build fast, scalable network applications, like web servers or APIs.
- It uses an event-driven, non-blocking I/O model. This makes it efficient and perfect for handling many requests at once.
- NPM-Node Package Manager

->What Can Node.js Do?

. Node.js can generate dynamic page content

. Node.js can create, open, read, write, delete, and close files on the server

. Node.js can collect form data

. Node.js can add, delete, modify data in your database

Example: Simple Node.js Server

javascript

Copy code

```
const http = require('http');


http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, World!');
}).listen(8080, () => {
  console.log('Server is running on http://localhost:8080');
});
```

---

2. JavaScript Event Loop

- JavaScript is single-threaded, meaning it can only do one thing at a time. But it handles many tasks efficiently using the **event loop**.
- When JavaScript needs to do something that takes time (like reading a file), it uses the event loop to avoid blocking other tasks.
- **Asynchronous operations** (like reading files, making HTTP requests) are sent to the background. Once complete, a **callback** function is used to handle the result.

---

3. Types of Functions in JavaScript

- **Regular (Named) Function**:

javascript

Copy code

```
function add(a, b) {
  return a + b;
}
console.log(add(5, 10)); // Output: 15
```

- **Arrow Function**:

javascript

Copy code

```
const add = (a, b) => a + b;
console.log(add(5, 10)); // Output: 15
```

- **Anonymous Function** (function without a name):

javascript

Copy code

```
const multiply = function(a, b) {
  return a * b;
};
```

console.log(multiply(3, 4)); // Output: 12

---

4. Data Types in JavaScript

- **Primitive Data Types**:
    - **String**: Textual data. Example: 'Hello'
    - **Number**: Numbers (integers, decimals). Example: 42, 3.14
    - **Boolean**: true or false.
    - **Undefined**: A variable declared but not assigned a value.
    - **Null**: Represents "no value".
    - **Symbol**: Unique value (advanced).
    - **BigInt**: Very large numbers.

Example:

javascript

Copy code

let name = 'Alice'; // String

let age = 25; // Number

let isStudent = true; // Boolean

let city; // Undefined

- **Non-Primitive (Reference) Data Types**:
    - **Object**: Collection of key-value pairs. Example: {name: 'John', age: 30}
    - **Array**: Ordered list of values. Example: [1, 2, 3]
    - **Function**: Block of code to perform tasks.

---

5. Variable Declaration and Hoisting

- Variables declared with var are **hoisted** (moved to the top of their scope), but their value is undefined until assigned.

- let and const are also hoisted but stay uninitialized, leading to ReferenceError if used before declaration.

Example:

javascript

Copy code

```
console.log(x); // Output: undefined (due to hoisting)

var x = 5;


console.log(y); // Error: Cannot access 'y' before initialization

let y = 10;
```

HighOrderFunction

. Function that takes other functions as an arrgument

Global Variable

. Can be accessed and modify from any part of the program,including functional brakets and modules.(can be accessed using 'this' function)

->**This Function**

.in java:this can be used to access global variables

.in js: if we call a function inside an object and with '.this' it will completely print the entire function and object.

Local Variables

. Can not be access from outside the function or block in which they are declared.

Closure

.Allow inner function to access the outer scope of a function.

. two functions, main and inner functions ,local scope of main function is inner function and lexical scope of inner function is main function

eg:

```
                function display(){          _
          _             function display1(){              |
          |             console.log(a);                   |
Local     |             }              _ _          |   Lexical scope
Scope     |       display()    ____|
          |       }
          |             const a=100
          |__           display1()
```

->Local scope, also known as function scope, refers to the visibility and accessibility of variables within a specific function or block of code.

->Lexical scope, also known as static scope or closure scope, is a concept in which the scope of a variable is determined by its location within the source code at the time of its declaration.

---

6. Function Parameters and Arguments

- **Parameters** are variables defined in the function.
- **Arguments** are the actual values passed to the function.

Example:

javascript

Copy code

```
function sum(a, b) { // a and b are parameters
  return a + b;
}
console.log(sum(3, 4)); // 3 and 4 are arguments
```

---

7. Callback Functions

- **Callback**: A function passed as an argument to another function, executed after the first function completes.
- **Callback Hell**: When multiple nested callbacks make code unreadable and hard to manage.
- **case of multiple** callback are nested on top of each other.Its complex to identify so we use PROMISE to reduce the complexity.

Example:

javascript

Copy code

```
function fetchData(callback) {
  setTimeout(() => {
    callback('Data fetched!');
  }, 1000);
}
fetchData(data => console.log(data)); // Output after 1 second: 'Data fetched!'
```

---

8. Promises

- A **promise** represents a value that may be available now, in the future, or never.
- It has three states: **Pending**, **Fulfilled**, or **Rejected**.
- can be used to reduce callback hell
- promise is an object that reperasent asynchronus operation

Example:

javascript

Copy code

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

```
  let success = true;
  if (success) resolve('Promise fulfilled');
  else reject('Promise rejected');
});


promise
  .then(data => console.log(data))  // Executes if resolved
  .catch(error => console.log(error));  // Executes if rejected
```

9. async/await

- **async/await** makes it easier to work with asynchronous code in a synchronous-like manner.

Example:

javascript

Copy code

```
async function fetchData() {
  try {
    let data = await new Promise(resolve => setTimeout(() => resolve('Fetched!'), 1000));
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
fetchData(); // Output after 1 second: 'Fetched!'
```

---

10. Node.js Modules

- **Modules** allow code to be split into separate files and reused.

- module.exports makes functions or variables available to other files using require().

Core modules:-

.HTTP

.FS

.OS

.PATH

>Http Modules

. Used to create a server

. To include the HTTP module, use the require() method:

eg: var http = require('http');

Example:

javascript

Copy code

```
// file1.js
function greet(name) {
  return `Hello, ${name}!`;
}
module.exports = greet;


// file2.js
const greet = require('./file1');
console.log(greet('Alice')); // Output: 'Hello, Alice!'
```

---

11. File System (fs) Module

- fs module lets you work with files.

Example: Writing to a file

javascript

Copy code

```
const fs = require('fs');
fs.writeFile('hello.txt', 'Hello, world!', (err) => {
  if (err) throw err;
  console.log('File written successfully');
});
```

---

12. HTTP Methods: GET and POST

- **GET**: Used to fetch data from the server.
- **POST**: Used to send data to the server.

Example: Simple GET Server

javascript

Copy code

```
const http = require('http');
http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, World!');
}).listen(8080);
```

---

13. Form Handling with formidable

- formidable helps handle form data and file uploads easily.

---

14. Connecting to MongoDB

- Connect to MongoDB using the mongodb package.

Example:

javascript

Copy code

```
const { MongoClient } = require('mongodb');
const url = 'mongodb://localhost:27017';
const client = new MongoClient(url);

async function connectDB() {
  try {
    await client.connect();
    console.log('Connected to MongoDB');
  } catch (err) {
    console.error(err);
  } finally {
    await client.close();
  }
}
connectDB();
```

=> ARRAY METHODS

1) Map

. Used to display array elemnts

. value can be return

. Syntax

```
a.map((abc)=>{
console.log(abc);
```

})