# Node Js

1. What is Node Js?
   * 1. Node Js is a java Script Runtime “java Script on Server”.
     2. Node Js uses V8 (it is written in C++) compiles a java Script to a Machine Code.
2. require(‘fs’);
   * 1. In Java Script require(‘fs’) is function used to include the built-in ‘fs’(file System Module) Module.

#### Topics:

1. **JavaScript Intro**
2. **Node.js Basics**
3. **Efficient Development**
4. **Using Express.js**
5. **Templating Engines**
6. **Model-View-Controller(MVC)**
7. **Advanced Routes & Models**
8. **Node + SQL (MySQL)**
9. **Using Sequalize**
10. **Node + NoSQL (MongoDB)**
11. **Using Mongoose**
12. **Sessions & Cookies**
13. **Authentication**
14. **Sending E-Mails**
15. **Authentication Deep Dive**
16. **User Input Validation**
17. **Error Handling**
18. **File Uploads & Download**
19. **Pagination**
20. **Async Requests**
21. **Handling Payments**
22. **REST API Basics**
23. **Advanced Rest API Features**
24. **Using Async-await**
25. **Web sockets & Socket.io**
26. **GraphQL**
27. **Deployment**
28. **Beyond Web Servers**
29. **Node + TypeScript, Deno**

**The REPL**

**R -** Read -> Read User Input.

**E -** Eval -> Evaluate User Input.

**P -** Print -> Print Output (Result).

**L -** Loop -> Wait For New Input.

**Execute Files (vs) Use the REPL**

Execute Files:

1. Used for Real Apps.
2. Predictable Sequence of steps

Use the REPL:

1. Great Playground.
2. Execute code as you write it.

##### JavaScript Intro:

1. Weakly Typed Language.
   1. No explicit type assignment.
   2. Data types can be switched dynamically.
2. Object Oriented Language.
   1. Data can be organized in logical objects.
   2. Primitive and reference types.
3. Versatile Language.
   1. Runs in browser & directly on a PC/server.
   2. Can perform a broad variety of tasks.

###### Example JS Code:

var name = "Vigneshwaran T";

var age = 23;

var isSingle = true;

function userDetails(userName, userAge, userMaritalStatus) {

  return (

    "Name is " +

    userName +

    " and Age is " +

    userAge +

    " and the user is Single: " +

    (userMaritalStatus ? 'Yes' : 'No')

  );

}

console.log(userDetails(name, age, isSingle));

// Output: Name is Vigneshwaran T and Age is 23 and the user is Single: Yes.

1. **let & const & var:**
   1. **var** can access a variable before it is declared, although its value will be `undefined`.
   2. **var** variables can be reassigned and mutated (changed) throughout the program execution.

// Example for var

  function example() {

    var x = 10;

    if (true) {

      var x = 20;

      console.log(x); // Output: 20

    }

    console.log(x); // Output: 20

  }

* 1. **let** they only are accessible within the block they are defined in (if statement, loop or function).

// Example for let

  function example() {

    let x = 10;

    if (true) {

      let x = 20;

      console.log(x); // Output: 20

    }

    console.log(x); // Output: 10

  }

* 1. **const** is same like **let** they only are accessible within the block they are defined in

// Example for const

  function example() {

      const x = 10;

      if (true) {

        const x = 20;

        console.log(x); // Output: 20

      }

      console.log(x); // Output: 10

    }

1. **Arrow Functions:**
   1. An Arrow function is a shorter defining functions in JavaScript introduced in ECMAScript 6 (ES6).
   2. An Arrow function provide a more compact and expressive syntax, making code easier to read and write.
   3. (=>) -> This is the symbol of arrow function.
   4. **Syntax for arrow function:**

const functionName = (parameter1, parameter2, ...) => {

  // Function body

  // Code to be executed

  // Return statement (if any)

};

* 1. We don’t need to use function keyword at any place. Because, Arrow function (=>) takes that place.
  2. **Example for Arrow function:**

const addName = (firstName, LastName) => {

    return firstName + ' ' + LastName

}

console.log(addName('Vigneshwaran', 'T'));

//Output: Vigneshwaran T

* 1. We can also write a code without using (“{ }”) Curly braces in arrow function that type of syntax.
  2. And also only when the body of the function in one line then only we use without (“{ }”) Curly braces in arrow functions.
  3. And we not need to add return syntax in one line arrow function.
  4. **Example for One-Line Arrow function:**

const mergeName = (firstName, LastName) => firstName + ' ' + LastName;

console.log(mergeName('Vigneshwaran', 'Thiruselvam'));

//Output: Vigneshwaran Thiruselvam

* 1. Arrow function also have some special behaviour such as lexical scoping ‘this’.
  2. **Example for One-Line Arrow function:**

const person = {

  name: 'Vigneshwaran T',

  age : 23,

  myDetails: function() {

    setTimeout(() => {

      console.log('My Name is ' + this.name + ' and my age is ' + this.age + '.');

      console.log(`My Name is ${this.name} and my age is ${this.age}.`);

    }, 1000);

  }

}

person.myDetails();

//Output: My Name is Vigneshwaran T and my age is 23.

//        My Name is Vigneshwaran T and my age is 23.

1. **Working with Objects, properties & Methods:**
   1. **Object:**
      1. We create an object with in the curly braces and assigned to a one variable name.
      2. Inside the object we must use key value pairs.
      3. A Key Value pair is also called a “property” or a “field” of the object.
      4. In Below example ‘name’, ‘age’, ‘role’ is the properties.
      5. **Example for Object:**

// Objects

    const Person = {

        name: 'Vigneshwaran',

        age: 23,

        role: 'Fullstacker'

    }

    console.log(Person.name); //Output: Vigneshwaran

    console.log(Person.age); //Output: 23

    console.log(Person.role); //Output: Fullstacker

* + 1. We can also write a function inside the object. Here is an example,

// Objects with function

    const Car = {

        model: 'Sports',

        color: 'Black',

        year: 2023,

        description() {

            console.log(`Hi I want ${this.model} Car ${this.color} Color ${this.year} Model.`)

        }

    }

    console.log(Car.description());

* + 1. In JavaScript, methods are functions that are associated with objects.

1. **Array & Array Methods:**
   1. Arrays are comes with square brackets (“[ ]”).
   2. We can write different types of data like string, int, etc.., within an array.
   3. **Example of Array Method:**

// Array

    const development = [

        'Front End',

        'Middleware',

        'Back End',

    ];

// Array Methods

    development.push('UI/UX');

    development.forEach((data) => console.log(data));

    console.log(development);

// Output: Front End

//         Middleware

//         Back End

//         UI/UX

//         [ 'Front End', 'Middleware', 'Back End', 'UI/UX' ]

1. **Arrays, Objects & Reference Types:**
   1. Objects & Arrays are also called as Reference Types.
2. **Understanding Spread & Rest Operators:**
   1. **Splice & Slice**
      1. **Splice:**
         1. The splice() method is used to change the contents of an array by Removing, Replacing, or Adding Elements.
         2. It modifies the original array in place.
         3. The basic syntax of splice() is.,

// splice()

array.splice(start, deleteCount, item1, item2);

* + - 1. **Examples for splice():**

// splice()

// array.splice(start, deleteCount, item1, item2);

// splice() examples

const animals = ['Lion', 'Tiger', 'Black Panther'];

// const modAnimals = animals.splice(1,1); // Delete

animals.splice(4,0,'Leapord'); // Insert Elements

animals.splice(1,1,'Cat'); // Replace Elements

console.log(animals); // Output: [ 'Lion', 'Black Panther' ]

// console.log(modAnimals); // Output: [ 'Tiger' ]

* + 1. **Slice:**
       1. The slice() method returns a shallow copy of a portion of a new array object.
       2. It doesn’t modify the original array.
       3. The basic syntax of slice() is.,

// slice()

// array.slice(start, end)

// slice() Examples

const fruits = ['Apple', 'Orange', 'Mango', 'Banana'];

console.log(fruits); // Output: [ 'Apple', 'Orange', 'Mango', 'Banana' ]

const fruitsCopy = fruits.slice();

console.log(fruitsCopy); // Output: [ 'Apple', 'Orange', 'Mango', 'Banana' ]

const fruitsSlice = fruits.slice(1,3);

console.log(fruitsSlice); // Output: [ 'Orange', 'Mango' ]

* 1. **Spread Operator:**
     1. The spread operator is denoted by ( ‘…’ ) three dots and is used to expand or spread the elements of an iterable object, such as an array or a string.
     2. It Allows you to copy conveniently copy elements from one array or object into another array or object, as well pass multiple arguments to the function.
        1. **Copying an array:**

// Spread operator

const originalArray = [1, 2, 3];

const copiedArray = [...originalArray];

copiedArray.push(4);

console.log(originalArray); // [1, 2, 3]

console.log(copiedArray); // [1, 2, 3, 4]

* + - 1. **Concatenating array:**

//// Concatenating arrays

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const concatArray = [...array1, ...array2];

console.log(array1); // [1, 2, 3]

console.log(array2); // [4, 5, 6]

console.log(concatArray); // [ 1, 2, 3, 4, 5, 6 ]

* + - 1. **Speading elements in function arguments:**

//// Spreading elements in function arguments

function sum(a, b, c) {

    return a + b + c;

}

const array = [1, 2, 3];

const arraySum = sum(...array);

console.log(arraySum); //6

* + - 1. **Creating shallow copy of an object:**

//// Creating a shallow copy of an object

const originalObject = { name: 'John', age: 30 };

const copiedObject = { ...originalObject };

console.log(copiedObject); // { name: 'John', age: 30 }

* + - 1. **Merging Objects:**

//// Merging Objects

const obj1 = { a: 1, b: 2 };

const obj2 = { c: 3, d: 4 };

const mergedObject = { ...obj1, ...obj2 };

console.log(mergedObject); // { a: 1, b: 2, c: 3, d: 4 }

* 1. **Rest Operator:**
     1. It is also same like spread operator but the rest operator only used inside the functions:

const arrayEle = (...args) => {

    return args;

}

console.log(arrayEle(1, 'One', 2, 'Two'));

// Output: [ 1, 'One', 2, 'Two' ];

1. **Destructuring:**
   1. Destructuring is a feature in JavaScript that allows you to extract values from arrays or objects and assign them to variables in a concise and convenient way.
   2. It is provides a simpler syntax for extracting values and can make your code more readable.
   3. It is only for Readable purpose for array and object.
   4. Here are some examples of how to use Destructuring in javascript.
      1. **Destructuring an Array:**

///// Destructuring an array:

const numbers = [ 1, 2, 3, 4, 5, 6 ];

// Extracting values from the array

    const [ a, b, c ] = numbers;

    console.log(a,b); // 1,2

    console.log(b);   // 2

console.log(c);   // 3

// Skipping values in the array

    const [, , ,d, e, f] = numbers;

    console.log(d); // 4

    console.log(e); // 5

    console.log(f); // 6

* + 1. **Destructuring an Object:**

const employee = {

    eName: 'Vigneshwaran',

    age: 23,

    salary: 27000,

    designation: 'Software Engineer'

};

// Extracting values from the object

const {eName, age} = employee;

console.log(eName); // Vigneshwaran

console.log(age); // 23

// Assigning to new variable names

const { eName: employeeName, age: employeeAge } = employee;

console.log(employeeName); // Vigneshwaran

console.log(employeeAge); // 23

// Providing default values

const {eName: empName, profession = 'Unknown'} = employee;

console.log(empName); // Vigneshwaran

console.log(profession); // Unknown

* + 1. **Destructuring function Parameters:**

///// Destructuring function Parameters:

const employeeDetails = ({ name, age, salary, designation }) => {

  console.log(

    `Employee name is ${name} and his age is ${age}. He bought ${salary} rupees salary and his designation is ${designation}. `

  );

};

const empDetails = {

    name: 'Vigneshwaran',

    age: 23,

    salary: 27000,

    designation: 'Software Developer'

};

employeeDetails(empDetails); //Employee name is Vigneshwaran and his age is 23. He bought 27000 rupees salary and his designation is Software Developer.

* + 1. **Destructuring nested objects:**

///// Destructuring nested objects:

const studentDetails = {

    name: 'Arun Karthik',

    age: 16,

    class: '12th standard',

    address: {

        city: 'Virudhunagar',

        state: 'Tamilnadu',

        country: 'India',

        zipCode: 626001

    }

}

const {

  name: studentName,

  age: studentAge,

  address: { city: studentCity, zipCode: studentZipCode },

} = studentDetails;

console.log(studentName); // Arun Karthik

console.log(studentAge); // 16

console.log(studentCity); // Virudhunagar

console.log(studentZipCode); // 626001

1. **Async Code & Promises:**
   1. **Sync & Async Code in JavaScript:**
      1. **Sync:**
         1. In Synchronous programming, each operation is executed one after the other in a blocking manner.
         2. This means that each operation must wait for the previous one to complete before it can start.
         3. Here’s an example of Synchronous Code:

const syncOperation = () => {

    console.log('Synchronous Operation 1');

    console.log('Synchronous Operation 2');

    console.log('Synchronous Operation 3');

}

syncOperation();

// Synchronous Operation 1

// Synchronous Operation 2

// Synchronous Operation 3

* + 1. **Async:**
       1. In Asynchronous programming, operations are non-blocking and can be executed independently without waiting for the previous operations to complete.
       2. Asynchronous operations are often used when dealing with time-consuming tasks, such as fetching data from server or reading/ writing files.
       3. Here’s an example of Asynchronous code using callbacks:

///// Asynchronous (Sync) Approach: /////

const asynchronousOperation = (callback) => {

    setTimeout(() => {

        console.log('Asynchronous Operation 1')

    }, 2000);

    callback();

}

const asyncOperation = () => {

    console.log('Asynchronous Operation 2')

}

asynchronousOperation(asyncOperation);

// Asynchronous Operation 2

// Asynchronous Operation 1

* + 1. **Promise:**
       1. A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
       2. Promises are commonly used for handling asynchronous operations such as fetching data from a server, reading a file or making an API call.

///// Promise: /////

const myPromise = new Promise((resolve, reject) => {

    setTimeout(() => {

        let isTransaction = false;

        if(isTransaction) {

            resolve('Your Transaction is Successful!');

        } else {

            reject('Your Transaction is Failed!');

        }

    }, 1000);

})

myPromise.then((result) => {

    console.log(result);

}).catch((error) => {

    console.log(error);

})

// Your Transaction is Failed!

1. **Template Literals:**
   1. One of the other feature, we’ll use from time to time are template literals.
   2. It’s a different way of writing strings.
   3. Instead of using double or single Quotation marks:
   4. 'A String', "Another string", we can use back-ticks (` Another way of writing strings`)

##### Node Basics: