WEEK-01:Javascript Using Node.js

1. Javascript

1.1 Introduction

JavaScript was created around April 1995 by Brendan Eich and used as the programming language for web development. Most websites use JavaScript, and it's the most widely used programming language because all modern web browsers on computers, tablets, and phones have built-in tools to understand and run JavaScript code. The arrival of Node.js in the last decade has enabled JavaScript programming outside of web browsers. With the dramatic success of the Node.js runtime environment for JavaScript, it has become one of the most-used programming languages for software development. All the JavaScript programs in the lab will be executed using Node.js.

Some of the key features of JavaScript are:

i. **High-Level Language :** It is a high-level and interpreted programming language which is easy to read, write and understand.
ii. **Interpreted Language :** It is executed by an interpreter which is built into browser or other runtime environment like Node.
iii. **Object Oriented :** It supports object-oriented programming allowing to create and manipulate objects with properties and methods.
iv. **Dynamic Typing :** Variables in JavaScript are not explicitly typed. Their types are determined dynamically at runtime.
v. **Asynchronous Programming :** It is designed to handle asynchronous operations efficiently. This is crucial for tasks like handling user input, making network requests, and managing events.
vi. **Event-Driven Programming :** It is event-driven which means it can respond to user actions and events triggered by the browser or other elements on a webpage.

1.2 Installation of Node on Ubuntu

There are multiple ways to install Node on Ubuntu operating system. Here two most preferred ways of installing it on Ubuntu 22.04 is provided.

Option 1 : Installation Using Snap

Step 1 : Visit the website https://snapcraft.io/node

Step 2 : Check the stable version of Node and install it using the following command

```
sudo snap install node  - -classic
```

Step 3 : Verify the version of the Node installed using the following command

```
node --version
```

Step 4 : If required to update the Node to latest version use the following command

```
sudo snap refresh node –channel=20
```

Option 2 : Installation Using Node Version Manager(NVM)

Step 1 : Visit the NVM gitub page https://github.com/nvm-sh/nvm and copy the curl command from the README file that displays on the mainpage.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Step 2 : Once the nvm script is installed into particular account source the .bashrc file using the following command

```
source ~/.bashrc
```

Step 3: List the versions of Node to install using the following command

```
nvm list-remote
```

Step 4: Install the stable version of Node using the following command

```
nvm install 20.10.0 or 18.18.0
```

1.3 Installation of Visual Studio Code on Ubuntu

Here the steps are provided to install Visual Studio Code on Ubuntu 22.04.

Step 1 : Visit the website https://code.visualstudio.com/Download and download .deb file into your local file system.

Step 2 : Open the terminal and go to specific directory where .deb file is located and execute the following command to install VS Code on the system

```
sudo dpkg –i  filename.deb
```

Step 3 : To open the VS Code run the following command

```
code
```

1.4 Javascript Programming Fundamentals

HelloWorld Program

Create a new JavaScript file named **demo.js** in the VS code editor and write the following line of code

```
console.log("Hello World");
```

Save the program and run the demo.js script using the following command

```
node demo.js
```

Variables

A variable is a named storage for JavaScript data. JavaScript variables can be declared in three ways:

i. Using the **var** keyword
The **var** keyword declares function-scoped or globally-scoped variables, optionally initializing each to a value.

```
var name = “Manipal”;
```

ii. Using the **let** keyword
The **let** keyword declares re-assignable, block-scoped local variables, optionally initializing each to a value.

```
let num = 10;
if(num==10){
  let num = 20;
  console.log(num);
}
console.log(num);
```

Expected Output

```
20
10
```

iii. Using the **const** keyword
The **const** keyword declares block-scoped local variables. The value of a constant can't be changed through reassignment using the assignment operator, but if a constant is an object, its properties can be added, updated, or removed.

```
const val = 100;

try {
  val = 10;
} catch (err) {
  console.log(err);
}

console.log(val);
```

Expected Output

```
TypeError: Assignment to constant variable.
100
```

Data Types

The latest ECMAScript standard defines eight data types for JavaScript out of which seven are primitive types and one is the Object type.:

i. **boolean :** stands for **true** or **false**.
ii. **null :** A special keyword denoting a null value.
iii. **undefined :** A top-level property whose value is not defined.
iv. **number :** An integer or floating point number. For example: 100 or 200.4715.
v. **bigint:** An integer with arbitrary precision. For example: 9007199254740992n.
vi. **string:** A sequence of characters that represent a text value. For example: "Manipal".
vii. **symbol:** A data type whose instances are unique and immutable.
viii. **object:** objects can be seen as a collection of properties.

Operators

Operators are the symbols which are used to perform operations on the data. Some of the widely used operators are assignment operators, arithmetic operators, comparison operators, and logical operators.

Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand. The simple assignment operator is equal (=), which assigns the value of its right operand to its left operand.

```
// Assign the string value “Hello” to the 'message' variable
let message = "Hello";

// Assign the Boolean value true to the 'on' variable
let on = true;
```

Arithmetic Operators

An arithmetic operator takes numerical values (either literals or variables) as their operands and returns a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/).

```
// Perform addition using ‘+’ operator
console.log(3+5)
// Get the remainder using ‘%’ operator
Console.log(12%5)
```

Comparison Operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true. The operands can be numerical, string, logical, or object values. In most cases, if the two operands are not of the same type, JavaScript attempts to convert them to an appropriate type for the comparison. This behavior generally results in comparing the operands numerically. The sole exceptions to type conversion within comparisons involve the === and !== operators, which perform strict equality and inequality comparisons.

```
// Equal comparison operator
const var1 = 3;
const var2 = 4;

console.log(var1==3); // returns true
console.log(var1==”3”);  // returns true
```

Logical Operators

Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value. However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.

```
// Logical AND operator
const a1 = 5;
const a2 = 100;
console.log(a1<10 && a2>10);
```

Control Flows(Conditionals)

Control flow is a feature in a programming language that allows for selectively running specific pieces of code based on different conditions that may arise. Using control flow allows for defining multiple paths a program can take based on the conditions present in your program.

The two commonly used conditional control flow statements are : **if…else** and **switch…case**.

```
// If…else Demo

let CGPA = 8;

if (CGPA >= 7.5) {
  console.log(‘Eligible for Campus Interview’);
}
else{
    console.log(‘Not Eligible’);
}


// Switch…Case Demo
```

```
let rating = 5;
switch(rating){
   case 1 : console.log('Very Poor');
            break;

   case 2 : console.log('Poor');
            break;

   case 3 : console.log('Average');
            break;

   case 4 : console.log('Good');
            break;

   case 5 : console.log('Very Good');
            break;

   default : console.log('No comments');
             break;

}
```

Control Flows(Loops)

Looping in JavaScript allows for the repeated execution of a block of code until a certain condition is met. The primary loop structures in JavaScript are the **for**, **while**, and **do-while** loops. These constructs enable efficient iteration through arrays, processing of data, and execution of tasks until a specified condition becomes false, providing a fundamental mechanism for handling repetitive tasks in programming.

```
// for loop demo
// Print the numbers from 1 to 5
for(let num=1;num<=5;num++){
   console.log(num);
}

// while loop demo
// Print the numbers from 1 to 5
let num = 1;
while(num<=5){
console.log(num);
num++; }
```

Arrays

An array is an object data type that can be used to hold more than one value. An array can be a list of strings, numbers, booleans, objects, or a mix of them all.

To create an array, use square brackets [] and separate the items using a comma.

```
// Array Declaration Demo

// Declaring an array of numbers
let numbers = [10, 20, 30, 40, 50];

// Declaring an array of strings
let sweets = ['Laddu','MysorePak','Jalebi','Rasmalai','Peda'];

// Declaring an array of mixed types
let mix = [1, 5.5789, 'Manipal', true];

// Accessing and displaying the array elements using index position
for (let i = 0; i < sweets.length; i++) {
   console.log(sweets[i]);
}
```

As array is an object in JavaScript, it offers special methods and properties to manipulate the array values.

```
// length property to get the size of an array
let proglangs = ['C', 'C++', 'Python', 'Java', 'Go'];
console.log('Length of the array is '+proglangs.length);

// push() method to add an item to the end of the array
proglangs.push('JavaScript');
console.log(proglangs);

// pop() method can be used to remove an item from the end of an array
proglangs.pop();
console.log(proglangs);

// unshift() method can be used to add an item from the front at index 0
proglangs.unshift('Ruby');
console.log(proglangs);

//shift() method can be used to remove an item from index 0
proglangs.shift();
console.log(proglangs);
```

Functions

A function is simply a section (or a block) of code that's written to perform a specific task. For example : log(), charAt(), indexOf() are some of the built-in functions available in the JavaScript. Besides the built-in functions, user-defined (custom) functions can also be created.

```
// Simple user-defined function
function displayName(){
   console.log('My name is ABC');
}

displayName();

// Function parameters and arguments
function displayName(name){  // name is the parameter passed
   console.log('My name is '+ name)
}
}

displayName('ABC'); // ‘ABC’ is the argument passed to displayName() function

//Function with default value for parameter
function displayName(name = 'ABC') {
   console.log(`My name is ${name}`);
}

displayName();

//Function with return value
function addition(num1,num2){
   return num1+num2;  // return statement
}

let result = addition(10,5);
console.log('Result of adding two numbers is: ',result);


//Function with rest parameter
// It is a parameter which can accept any number of values as its arguments
function printValues(...args){
   console.log(args);
}

printValues('A','B','C','D');
// Arrow function
// JavaScript arrow functions are concise functions with simplified syntax and lexical scoping
const displayName = (name) =>{
  console.log('My name is ' + name);
};

displayName('ABC');
```

Callback Functions

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action. There are two ways in which the callback may be called: synchronous and asynchronous. Synchronous callbacks are called immediately after the invocation of the outer function, with no intervening asynchronous tasks, while asynchronous callbacks are called at some point later, after an asynchronous operation has completed.

```
//callback function
const message = function() {
    console.log("This message is shown after 3 seconds");
}

// log a message to the console after 3 seconds using simple callback
setTimeout(message, 3000);
```

forEach Loop

forEach is a higher-order function in JavaScript that is used to loop over arrays and execute a function for each element. The function takes a callback function as an argument, which is executed for each element in the array.

```
const fruits = ["apple", "banana", "orange"];

fruits.forEach(function(fruit, index) {
  console.log(index, fruit);
});
```

Objects

An object is a special data type that allows storing more than one value, just like an array. The difference between an object and an array is that an array stores data as a list of items, while an object stores data in a **key:value** pair format.

```
// Object Demo
```

```
let Student = {'Name':'Ram', 'Roll No': 100, 'Course': 'M.Tech'};

console.log('Student Name: '+Student.Name);
console.log('Course Name: ' + Student.Course);
```

<u>Classes</u>

A class is a blueprint/template for creating objects, providing a convenient way to define object structure and behavior. It encapsulates data (properties/attributes) and methods (functions) that operate on that data within a single unit.

```
// class demo
class Person {  // Define the class
   constructor(name, age) {
     this.name = name;
     this.age = age;
   }
   greet() {
     console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
   }
 }

 // Creating an instance of the class
 const person = new Person('Ram', 25);
 // Accessing properties and calling a method
 console.log(person.name);
 person.greet();
```

2. Node.js

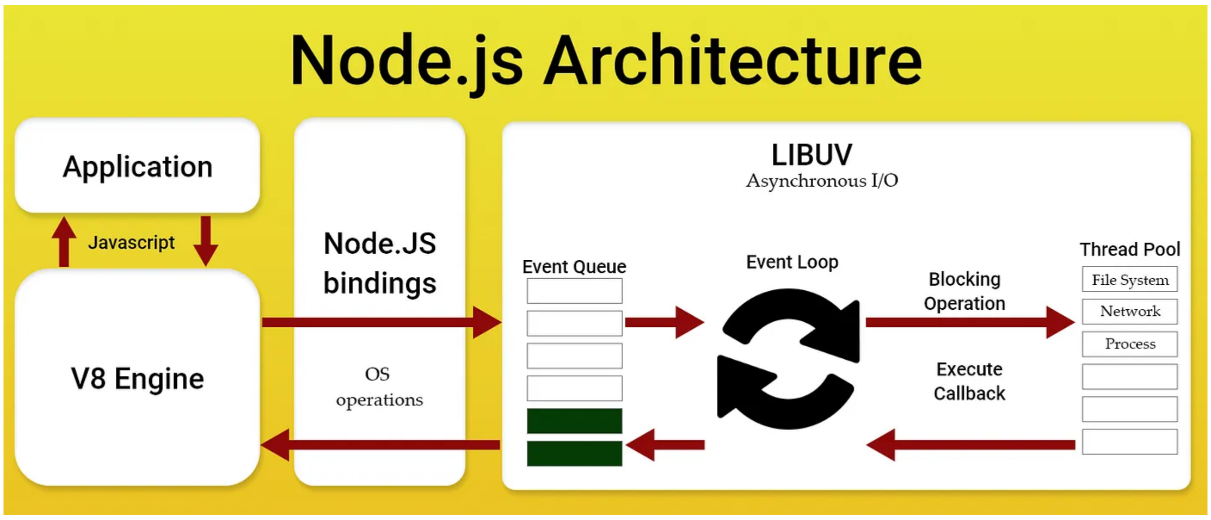
2.1 Introduction

Node.js is an open-source and cross platform runtime environment for executing JavaScript code. It is built with Google Chrome’s JavaScript engine which is able to support JavaScript code execution outside the browser, similar to general purpose programs. With this capability, JavaScript can now be used for server-side programming which was previously used for client-side scripting. The key advantages of Node.js are as follows:

i. It is good for prototyping and agile development
ii. It is highly scalable and offer quicker responses.
iii. It makes Javascript available everywhere
iv. It provides more cleaner and consistent way to build codebase.
v. It offers non-blocking I/O and follows asynchronous architecture.

vi. It is suitable for I/O intensive applications

2.2 Node.js Architecture



The main components of Node.js architecture are as follows:

i. The **V8 engine** is the fundamental part of Node.js architecture which helps to convert JavaScript code to machine code which is C++ that can be understood by machines.
ii. **LIBUV** is an open-source library that focuses on asynchronous I/O which provides access to Node.js as a computer operating system, file system, and networking.
iii. **Event Queue** manages the incoming client request and passes them to the event loop sequentially.
iv. The **Event Loop** is responsible for handling small tasks such as executing callback functions or network I/O.
v. **Thread Pool** offloads the events to the thread pool is responsible for handling heavy tasks such as file access, cryptography related things, file compression and DNS lookups.

2.3 Global Object

A global object is an object that always exists in the global scope. These objects are available in all modules. They are built-in objects that are part of the JavaScript and can be used directly in the application without importing any particular module.

For example : **console** is an in-built global object which can be used print to **stdout** and **stderr**.

2.4 Node.js Modules

A Node.js application is made up of many JavaScript files. For the application to stay organized and efficient, these files need to have access to one another's contents whenever necessary. Each JavaScript file or folder containing a code library is called a **module**.

A module is essentially a reusable block of code that can be used to perform a specific set of tasks or provide a specific functionality. A module can contain variables, functions, classes, objects, or any other code that can be used to accomplish a particular task or set of tasks.

Creating and Importing a Module

A module can be defined by a user by writing a function, such as **greetMessage()**, in a file named **greet.js**. This module can then be exported for use by others, namely in **app.js**. **module.exports** is a special object in Node.js that allows to export functions, objects, or values from a module, so that other modules can access and use them. **require()** is a function in Node.js that is used to load external modules or files into JavaScript program.

greet.js

```
function greetMessage(name){
   console.log(`Good Day ${name}`);
}
module.exports.greetMessage = greetMessage;  // export the function
```

app.js

```
const greet = require('./greet'); // import the module

greet.greetMessage('ABC');
```

There are two types of modules in Node.js

i. **Built-in modules** are included in Node by default, allowing users to utilize them without the need for installation. It is required to import the modules and begin using their functionalities. Some of the popular built-in modules are **os**, **path**, **fs**, **stream**, **http** etc.

ii. **External Modules** are modules created by other developers which are not included in Node by default. As a result the installation of module is required before using them.

Using Built-in JavaScript Modules

Some of the most commonly used built-in modules in Node.js:

i. **fs :** This module provides a way to interact with the file system in Node.js. For example, it can be used to read and write files, create directories, and perform other file-related operations.

```
// Demo of fs module
const fs = require('fs');
const files = fs.readdirSync('./');
console.log(files);
```

ii. **os :** This module provides many useful properties and methods to interact with the operating system and server.

```
// Demo of os module
const os = require('os');
// return the cpu architecture
console.log("CPU architecture: " + os.arch());
// It returns the amount of free system memory in bytes
console.log("Free memory: " + os.freemem());
// It return total amount of system memory in bytes
console.log("Total memory: " + os.totalmem());
```

iii. **path :** This module provides utilities for working with file and directory paths. For example, it can help to join and normalize path segments, extract the directory name or file extension from a path, and perform other path-related operations.

```
// Demo of path module
const path = require('path');
const fullPath = path.join('/my', 'path', 'to', 'ABC.txt');
console.log(fullPath);
```

iv. **http:** This module provides a way to create HTTP servers and clients in Node.js. For example, it can be used to create a simple web server that serves static files or dynamically generated content.

```
// Demo of http module
const http = require('http');
// create an instance of server
const server = http.createServer(function(req,res){
   if(req.url==='/'){
      res.write('Welcome to Node.js programming');
      res.end();
   }
});
```

```
server.listen(3000); // Make server to listen at port 3000
console.log('Server listening on port 3000...');
```

Questions To Be Solved

1. Write a JavaScript program that defines a class named **Employee** featuring properties such as **‘name’**, **‘monthlySalary’** and **‘qualification’**. Within this class, incorporate a method designed to compute the annual salary of an employee. Extend this functionality by introducing a subclass named **Manager** which inherits from the **Employee** class. The **Manager** class should add an extra property named **‘department’.** Override the annual salary calculation method within the **Manager** subclass to accommodate 20% bonuses on annual salary specifically for managers. Instantiate two objects of the **Manager** class and determine their respective annual salaries.

2. Write a JavaScript program to create a sales register machine that can manage a shopping cart. The program should be able to add items to the cart, compute the total price, calculate discounts, and accept payments in INR. The available merchandise includes four items: Phone – ₹10,000, Smart TV – ₹15,000, Laptop – ₹40,000, and Gaming Console – ₹20,000. A 10% discount should be applied when the total purchase price exceeds ₹20,000. Implement methods named **displayMerchandise()**, **addItemToCart()**, **computeTotalPrice()**, and **acceptPayment()** to meet the following requirements.
   a) The **displayMerchandise()** method should take an appropriate merchandise list and display the items available for sale.
   b) The **addItemToCart()** method should take the name of an item as a parameter. If the item is available in the merchandise, it should be added to the shopping cart. If the item is not available, the method should display an appropriate message indicating that the item is not in stock.
   c) The **computeTotalPrice()** method should iterate over the items in the shopping cart, calculate the total price for all items, and apply a 10% discount if the total price exceeds ₹20,000. Finally, it should display the total amount to be paid by the customer.
   d) The **acceptPayment()** method should accept the payment from the customer. If the amount paid by the customer is equal to or greater than the total price, then display the amount of change to be given back and thank the customer for the purchase. If the amount paid by the customer is less than the total price, display a message indicating that the customer does not have enough money to purchase the items.
3. Write a Javascript program to develop a recipe book application that allows users to add, view, and search for recipes. The program should meet the following requirements for the application.
   a) Create a **recipe.js** file and export a **Recipe** object with properties such as **name**, **ingredients**, **instructions**, **rating**, **category**, etc.
   b) Create a **recipeData.json** file that will be used to store all the recipes.

c) Create a **searchDisplay.js** file and export a **searchDisplayRecipes()** function that takes a search term as a parameter and displays all the values of particular recipe from **recipeData.json** file that match the search term.
d) Create an **addRecipe.js** file and export an **addRecipe()** function that takes a **Recipe** object as a parameter and adds it to the **recipeData.json** file.
e) Create a **deleteRecipe.js** file and export a **deleteRecipe()** function that takes a recipe name as a parameter and deletes the recipe entry in **recipeData.json** file.
f) Create an **app.js** file that will act as the entry point to the application. This file should import all the modules created and use them to create a simple command-line interface for the recipe book.

Additional Questions

1. Write a JavaScript program that generates a subset of a given string. Consider the following sample data: "dog." The expected output should be an array containing all possible subsets, such as ["d", "do", "dog", "o", "og", "g"].
2. Write a JavaScript program to simulate a clock, with the output displayed every second. The expected console output should follow the format: "hh:mm:ss".