# DAY - 01

## Problems

**Problem 1**: Conditional Statements (if-else)

You run a movie theater, and you want to offer discounts based on a person's age. Write a JavaScript program that asks the user for their age and then displays a message:
- If the age is less than 18, display "You get a 20% discount!"
- If the age is between 18 and 65 (inclusive), display "Normal ticket price applies."
- If the age is 65 or older, display "You get a 30% senior discount!"

**Problem 2**: Variables (var and const)

Create a JavaScript program to calculate the area of a rectangle. Ask the user for the length and width of the rectangle and store them in variables. Calculate and display the area using the formula: `area = length * width`.

**Problem 3**: Objects and Properties

You're creating an online store. Define a JavaScript object named "product" with the following properties:
- name (string)
- price (number)
- inStock (boolean)

Create at least three products using your object template and display their details using console.log.

**Problem 4**: Arrays

You're organizing a party and want to keep track of the guest list. Create an array called "guestList" and add the names of at least five guests. Then, write a program that checks if a given name is on the guest list. If the name is found, display "Welcome to the party, [name]!"; otherwise, display "Sorry, you're not on the guest list."

**Problem 5**: JSON (JavaScript Object Notation)

You're working on a weather app. Create a JSON object representing the weather forecast for a specific day. Include properties like "date," "temperature," "conditions," and "humidity." Display the information using console.log.

Remember to encourage your students to experiment and think creatively while solving these problems. They can use the concepts you've taught them to come up with their own solutions. This will not only help solidify their understanding but also foster their problem-solving skills in JavaScript.

## Solution

**Problem 1**: Conditional Statements (if-else)

```
const age = prompt("Please enter your age:");
if (age < 18) {
  console.log("You get a 20% discount!");
} else if (age >= 18 && age <= 65) {
  console.log("Normal ticket price applies.");
} else {
  console.log("You get a 30% senior discount!");
}
```

**Problem 2**: Variables (var and const)

```
const length = parseFloat(prompt("Enter the length of the rectangle:"));
const width = parseFloat(prompt("Enter the width of the rectangle:"));
const area = length * width;
console.log("The area of the rectangle is:", area);
```

## **Problem 3**: Objects and Properties

```
const product1 = {
  name: "Smartphone",
  price: 399.99,
  inStock: true
};

const product2 = {
  name: "Laptop",
  price: 999.99,
  inStock: false
};

const product3 = {
  name: "Headphones",
  price: 49.99,
  inStock: true
};

console.log(product1);
console.log(product2);
console.log(product3);
```

## **Problem 4**: Arrays

```
const guestList = ["Alice", "Bob", "Charlie", "David", "Eve"];

const nameToCheck = prompt("Enter your name:");
if (guestList.includes(nameToCheck)) {
  console.log(`Welcome to the party, ${nameToCheck}!`);
} else {
  console.log("Sorry, you're not on the guest list.");
}
```

**Problem 5**: JSON (JavaScript Object Notation)

```
const weatherForecast = {
  date: "2023-08-06",
  temperature: 28,
  conditions: "Sunny",
  humidity: 60
};

console.log("Weather Forecast for", weatherForecast.date);
console.log("Temperature:", weatherForecast.temperature + "°C");
console.log("Conditions:", weatherForecast.conditions);
console.log("Humidity:", weatherForecast.humidity + "%");
```

## DAY - 02

### Problems

**Problem 1: NPM and Package.json**

You're starting a new project and want to manage your project's dependencies using NPM. Explain the purpose of NPM and how it helps in managing packages. Create a simple `package.json` file for your project, specifying the name, version, and a few dependencies of your choice.

**Problem 2: Writing Functions**

Write a JavaScript function named `calculateCircleArea` that takes the radius of a circle as a parameter and returns the area of the circle. You can use the formula `area = π * radius^2`. Test the function with a few different radii.

**Problem 3: Callback Functions**

Create a function named `performOperation` that takes two numbers and a callback function as parameters. The callback function should determine the operation to be performed (addition, subtraction, multiplication, or division) on the two numbers. Call the `performOperation` function with different numbers and callback functions for each mathematical operation.

**Problem 4: Using the 'fs' Module**

Write a Node.js program that uses the `fs` module to read the contents of a text file named "notes.txt" and display them in the console.

**Problem 5: Using 'os' Module**

Create a Node.js program that uses the `os` module to display the following system information:

- Total memory available (in bytes)
- Free memory available (in bytes)
- Operating system platform
- Number of CPU cores

**Problem 6: 'lodash' Usage**

Use the `lodash` library to solve the following problem: Given an array of numbers, write a function that returns the sum of all even numbers in the array. Use the `_.sumBy` function from `lodash` to achieve this.

## Solution

**Problem 1: NPM and Package.json**

NPM (Node Package Manager) is a tool that helps you manage libraries and packages in your Node.js projects. It allows you to easily install, update, and remove packages that you need for your project.

To create a `package.json` file for your project, you can use the following example:

```json
{
  "name": "my-project",
  "version": "1.0.0",
  "dependencies": {
    "lodash": "^4.17.21"
  }
}
```

**Problem 2: Writing Functions**

```javascript
function calculateCircleArea(radius) {
  return Math.PI * radius ** 2;
}

console.log(calculateCircleArea(5)); // Output: 78.53981633974483
console.log(calculateCircleArea(10)); // Output: 314.1592653589793
```

**Problem 3: Callback Functions**

```javascript
function performOperation(num1, num2, operationCallback) {
  return operationCallback(num1, num2);
}

function add(x, y) {
  return x + y;
}

function subtract(x, y) {
  return x - y;
}

function multiply(x, y) {
  return x * y;
}

function divide(x, y) {
  return x / y;
}

console.log(performOperation(10, 5, add));      // Output: 15
console.log(performOperation(10, 5, subtract)); // Output: 5
console.log(performOperation(10, 5, multiply)); // Output: 50
console.log(performOperation(10, 5, divide));   // Output: 2
```

**Problem 4: Using 'fs' Module**

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

fs.readFile('notes.txt', 'utf8', (err, data) => {
  if (err) {
    console.error("Error reading file:", err);
    return;
  }
  console.log(data);
});
```

**Problem 5: Using 'os' Module**

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

console.log("Total Memory:", os.totalmem());
console.log("Free Memory:", os.freemem());
console.log("Platform:", os.platform());
console.log("Number of CPU Cores:", os.cpus().length);
```

**Problem 6: 'lodash' Usage**

```
const _ = require('lodash');

function sumOfEvenNumbers(numbers) {
  const evenNumbers = _.filter(numbers, num => num % 2 === 0);
  return _.sumBy(evenNumbers);
}

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(sumOfEvenNumbers(numbers)); // Output: 30 (2 + 4 + 6 + 8 + 10)
```

## DAY - 03

## Problems

### Problem 1: Understanding Servers and Express.js

Explain in your own words what a server is in the context of Node.js. Then, write step-by-step instructions on how to create a basic server using Express.js.

### Problem 2: JSON Manipulation

a) Define JSON and explain its importance in web development.

b) Given a JSON data string: `{"name": "Alice", "age": 25, "hobbies": ["reading", "painting"]}`, explain how you would extract the value of the "age" key.

c) How would you convert the following object into a JSON data string? `{"title": "Book", "pages": 200}`

### Problem 3: API and Endpoints

a) Define what an API is and its role in software development.

b) Explain what an endpoint is in the context of web APIs.

c) Provide an example of an endpoint you might find in a social media application.

### Problem 4: Creating a Route with Express.js

a) Explain what the HTTP GET method is used for in the context of web development.

b) Write the code to create a simple Express.js route that responds with "Hello, World!" when a user visits the root URL ("/").

**Problem 5: JSON Parsing and Object Conversion**

a) Given a JSON data string: `{"product": "Laptop", "price": 999.99}`, explain how you would parse it into a JavaScript object.

b) You have an object: `{ "name": "Bob", "age": 30 }`. How would you convert it into a JSON data string?

**Problem 6: Building a Basic API**

Imagine you're building an API for a weather app. Your API needs an endpoint to retrieve the current weather. Create an Express.js route that responds with a JSON object containing the current temperature, conditions, and city.

## Solution

**Problem 1: Understanding Servers and Express.js**

- A server in Node.js is a computer program that receives and responds to requests from clients (like web browsers or mobile apps) over a network. It processes requests and sends back appropriate responses.
- To create a basic server using Express.js:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello from Express server!');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

**Problem 2: JSON Manipulation**

a) JSON (JavaScript Object Notation) is a lightweight data interchange format used to exchange data between a server and a client. It's easy for humans to read and write, and it's easy for machines to parse and generate.

b) To extract the value of the "age" key from the JSON data:

```
const age = jsonObject.age;
```

c) To convert an object into a JSON data string:

```
const jsonString = JSON.stringify({"title": "Book", "pages": 200});
```

**Problem 3: API and Endpoints**

a) An API (Application Programming Interface) is a set of rules and protocols that allows different software components to communicate and interact with each other. It defines how requests and responses should be structured.

b) An endpoint is a specific URL (Uniform Resource Locator) that represents a particular function or service provided by an API. It's the specific location where clients can make requests to access certain data or perform actions.

c) Example of an endpoint in a social media app: `/users/{username}` to retrieve user information based on their username.

**Problem 4: Creating a Route with Express.js**

a) The HTTP GET method is used to request data from a server. It's often used to retrieve information or resources from a specified URL.

b) Code to create a simple Express.js route:

```
app.get('/', (req, res) => {
  res.send('Hello, World!');
});
```

**Problem 5: JSON Parsing and Object Conversion**

a) To parse a JSON data string into a JavaScript object:

```
const jsonData = '{"product": "Laptop", "price": 999.99}';
const parsedObject = JSON.parse(jsonData);
console.log(parsedObject.product); // Output: Laptop
```

b) To convert an object into a JSON data string:

```
const objectToConvert = { "name": "Bob", "age": 30 };
const jsonString = JSON.stringify(objectToConvert);
console.log(jsonString); // Output: {"name":"Bob","age":30}
```

**Problem 6: Building a Basic API**

```
const express = require('express');
const app = express();

app.get('/weather', (req, res) => {
  const weatherData = {
    temperature: 25,
    conditions: 'Sunny',
    city: 'Los Angeles'
  };
  res.json(weatherData);
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

# DAY - 04

## Problems

**Question 1: Difference Between MongoDB Server & Node.js Server**

Explain the difference between a MongoDB server and a Node.js server in terms of their roles and functionalities. Provide examples of situations where you would use each server type.

**Question 2: MongoDB Queries**

a) Write a MongoDB query to create a new document in a collection named "students" with fields "name," "age," and "grade."

b) Write a MongoDB query to update the "age" field of a document in the "employees" collection with the name "John" to 30.

c) Write a MongoDB query to delete a document from the "products" collection with the name "Product A."

d) Write a MongoDB query to retrieve all documents from the "orders" collection where the total amount is greater than $100.

**Question 3: SQL vs. MongoDB**

Explain the main differences between SQL databases and MongoDB in terms of data structure, querying language, and use cases. Provide examples of scenarios where you might choose one over the other.

**Question 4: Query Comparison**

Compare and contrast the following MongoDB and SQL queries for retrieving data:

a) MongoDB: `db.products.find({ category: "Electronics" })`

SQL: `SELECT * FROM products WHERE category = "Electronics"`

b) MongoDB: `db.users.findOne({ username: "Alice" })`

SQL: `SELECT * FROM users WHERE username = "Alice"`

c) MongoDB: `db.orders.aggregate([{ $group: { _id: "$status", total: { $sum: "$amount" } } }])`

SQL: `SELECT status, SUM(amount) as total FROM orders GROUP BY status`

## Solution

**Question 1: Difference Between MongoDB Server & Node.js Server**

**Answer**: The MongoDB server is responsible for storing and managing data in the MongoDB database. It handles data storage, retrieval, and manipulation operations. On the other hand, a Node.js server is a runtime environment that executes JavaScript code. It handles incoming requests from clients, processes them, and can interact with databases like MongoDB to retrieve or update data. You would use a MongoDB server to store and manage data, while a Node.js server is used to handle application logic and serve client requests.

**Question 2: MongoDB Queries**

a) Answer:

```
db.students.insertOne({ name: "John", age: 20, grade: "A" });
```

b) Answer:

```
db.employees.updateOne({ name: "John" }, { $set: { age: 30 } });
```

c) Answer:

```
db.products.deleteOne({ name: "Product A" });
```

d) Answer:

```
db.orders.find({ totalAmount: { $gt: 100 } });
```

**Question 3: SQL vs. MongoDB**

**Answer**: SQL databases use structured tables with rows and columns to store data, while MongoDB uses flexible and dynamic documents in collections. SQL databases use SQL as a querying language, while MongoDB uses a JavaScript-like syntax for queries. SQL databases are suitable for applications with well-defined and structured data, such as financial systems. MongoDB is better for projects with changing or unstructured data, like content management systems or real-time analytics.

**Question 4: Query Comparison**

a) Answer: Both queries retrieve products with the category "Electronics."

b) Answer: Both queries retrieve a user named "Alice."

c) Answer: Both queries calculate the total amount of orders grouped by status.

# DAY - 05

## Problems

**Question 1: Create a POST Method API**

Create a POST method API to store data or menu items as per the schema we discussed ( /menu )

**Question 2: Create a GET Method API**

Create a GET method API to List down the All Menu Items as per the schema we discussed ( /menu )

**Question 3: Creating a POST API with Express and Mongoose**

You're building a simple task management application. Your task is to create a POST API endpoint for adding new tasks to the database. Assume you've already set up an Express application and connected it to your MongoDB using Mongoose.

a) Design the Mongoose schema for a "Task" with fields for "title," "description," "priority," and "dueDate."

b) Create a POST API endpoint `/api/tasks` that allow clients to submit new tasks to the database. Ensure it handles request validation and responds with the newly created task.

**Question 4: Creating a GET API with Express and Mongoose**

Continuing with the task management application, you need to create a GET API endpoint for retrieving a list of tasks from the database.

Create a GET API endpoint `/api/tasks` that retrieve a list of all tasks from the database. Ensure it handles errors and responds with the list of tasks in JSON format.

## Solution

**Answer 1:** POST method API to store data or menu items as per the schema we discussed on /menu

```
app.post('/menu', async (req, res) => {
    try {
      const menuItemData = req.body; // Assuming the request
body contains menu item data

      // Create a new menu item using the Mongoose model
      const menuItem = new MenuItem(menuItemData);

      // Save the new menu item to the database
      const menu_data = await menuItem.save();

      console.log('Menu item saved');
      res.status(201).json(menu_data);
    } catch (error) {
      console.error('Error creating menu item:', error);
      res.status(500).json({ error: 'Internal server error' });
    }
  });
```

**Answer 2:** GET method API to List down the All Menu Items as per the schema we discussed on /menu

```
app.get('/menu', async (req, res) => {
    try {
      // Use the Mongoose model to find all menu items in the
database
      const menuItems = await MenuItem.find();

      // Send the list of menu items as a JSON response
      res.json(menuItems);
    } catch (error) {
      console.error('Error fetching menu items:', error);
      res.status(500).json({ error: 'Internal server error' });
    }
  });
```

**Answer 3: Creating a POST API with Express and Mongoose**

a) Design the Mongoose schema for a "Task" with fields for "title," "description," "priority," and "dueDate."

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

const taskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    trim: true,
  },
  description: {
    type: String,
    required: true,
  },
  priority: {
    type: String,
    enum: ['High', 'Medium', 'Low'],
    default: 'Medium',
  },
  dueDate: {
    type: Date,
    required: true,
  },
});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;
```

b) Create a POST API endpoint `/api/tasks` that allow clients to submit new tasks to the database. Ensure it handles request validation and responds to the newly created task.

```
// POST /api/tasks - Create a new task
app.post('/api/tasks', async (req, res) => {
  try {
    const task = new Task(req.body);
    await task.save();
    res.status(201).send(task);
  } catch (error) {
    res.status(400).send(error);
  }
});
```

**Answer 4: Creating a GET API with Express and Mongoose**

a) Create a GET API endpoint `/api/tasks` that retrieve a list of all tasks from the database

```
app.get('/api/tasks', async (req, res) => {
  try {
    const tasks = await Task.find();
    res.status(200).send(tasks);
  } catch (error) {
    res.status(500).send(error);
  }
});
```

# DAY - 06

## Problems

**Question 1: Create a parameterized GET Method API for the Menu Item on the Basis of taste Type via using Express Router**

**( /menu/:taste )**

**Question 2: Create a PUT Method API to update the MenuItem Records ( menu/:id )**

**Question 3: Create a DELETE Method API to delete the MenuItem Records ( menu/:id )**

## Solution

**Answer 1:** parameterized GET Method API

```
router.get('/:taste', async (req, res) =>{
    try{
        const tasteType = req.params.taste; // // Extract the taste type
from the URL parameter
        if(tasteType == 'sweet' || tasteType == 'sour' || tasteType ==
'spicy' ){
            const response = await MenuItem.find({taste: tasteType});
            console.log('response fetched');
            res.status(200).json(response);
        }else{
            res.status(404).json({error: 'Invalid Taste type'});
        }
    }catch(err){
        console.log(err);
        res.status(500).json({error: 'Internal Server Error'});
    }
})
```

**Answer 2:** Update Menu Item Method API

```
router.put('/:id', async (req, res)=>{
    try{
        const menuId = req.params.id; // Extract the id of Menu Item from the
URL parameter
        const updatedMenuData = req.body; // Updated data for the Menu Item

        const response = await MenuItem.findByIdAndUpdate(menuId,
updatedMenuData, {
            new: true, // Return the updated document
            runValidators: true, // Run Mongoose validation
        })

        if (!response) {
            return res.status(404).json({ error: 'Menu Item not found' });
        }

        console.log('data updated');
        res.status(200).json(response);
    }catch(err){
        console.log(err);
        res.status(500).json({error: 'Internal Server Error'});
    }
})
```

**Answer 3:** Delete Menu Item Method API

```
router.delete('/:id', async (req, res) => {
    try{
        const menuId = req.params.id; // Extract the Menu's ID from the URL
parameter
        // Assuming you have a MenuItem model
        const response = await MenuItem.findByIdAndRemove(menuId);
        if (!response) {
            return res.status(404).json({ error: 'Menu Item not found' });
        }
        console.log('data delete');
        res.status(200).json({message: 'Menu Deleted Successfully'});
    }catch(err){
        console.log(err);
        res.status(500).json({error: 'Internal Server Error'});
    }
})
```