KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Pollachi Main Road, Eachanari Post, Coimbatore – 641 021, INDIA

ADVANCED WEB FRAMEWORKS

(23BECS542A)

For

V SEMESTER B.E (CSE)

Faculty of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Prepared By

Mrs. A. Shalini & Mrs. M. Abinaya
Assistant Professor
Department of Computer Science and Engineering

ADVANCED WEB FRAMEWORKS

(23BECS542A)

For

V SEMESTER B.E (CSE)

Signature
Staff in Charges

Signature
Head of the Department

Faculty of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore

ADVANCED WEB FRAMEWORKS - 23BECS542A

LAB MANUAL

LIST OF EXPERIMENTS

1. Demonstrate the Git Commands for Version Controlling
2. Programs using flow control statements, arrays and arrow functions.
3. Develop simple application using NodeJS.
4. Develop Rest API with NodeJS.
5. Develop simple application using MongoDB.
6. Develop Rest API with NodeJS and MongoDB.
7. Develop simple application using ReactJS.
8. Develop simple application using ReactJS Components.
9. Develop simple application using React Context with styles.
10. Develop Rest API with Axios.
11. Develop Rest API with react-query and SWR.
12. Developing full stack application using ReactJS and MongoDB.

Total: 30 Hours

Ex. No: 1 Demonstrate the Git Commands for Version Controlling

AIM:
To demonstrate basic Git commands for version controlling a simple HTML/CSS application.

ALGORITHM:
Step 1: Create a new folder for the simple application and add files like index.html and style.css.
Step 2: Open the terminal and navigate to the project folder using the cd command.
Step 3: Initialize a Git repository using the command git init.
Step 4: Stage all the project files for commit using the command git add ..
Step 5: Commit the changes to the local repository using the command
git commit -m "Initial commit of simple HTML project".
Step 6: Connect the local repository to a remote Git repository using:

- git remote add origin <repository-URL>
- git push -u origin main

PROGRAM:
1. Initialize a Repository:

To start version controlling your project, navigate to your project directory in the terminal and run:
git init
This initializes a new Git repository in your project directory.

2. Add Files to Staging Area:

Before committing changes, you need to add files to the staging area. You can add specific files or add all files with:

git add <file1> <file2> ...

or

git add <file> # Add a specific file
git add . # Add all files in the current directory

3. Commit Changes to Repository:

Once files are added to the staging area, you can commit them with a descriptive message:
git commit -m "Your commit message"

4. Check Status of Working Directory:

To see the status of your repository and any changes that need to be committed or staged, use:
git status

5.View Commit History:

To view the commit history of your repository, use:
git log

6. Create Branches:

Branches allow you to work on different features or versions of your project simultaneously. To create a new branch, use:

git branch <branch_name>

7. Switch to a Branch:

To switch between branches, use:

git checkout <branch_name>

8. Merge Changes from One Branch to Another:(Merge Branches)

After completing work on a feature branch, you can merge it back into the main branch (e.g., master or main) using:

git merge <branch_name>

9. Push Changes to Remote Repository:
If you're collaborating with others or using a remote repository (e.g., GitHub), you can push your changes with:

git push <remote_name> <branch_name>

10. Pull Latest Changes from Remote Repository:
To get the latest changes from a remote repository, use:

git pull <remote_name> <branch_name>

11. Clone a Repository:

To clone a repository from a remote Git server (such as GitHub, GitLab, or Bitbucket) to your local machine, you can use the git clone command followed by the URL of the repository you want to clone.
Here's the general syntax:
git clone <repository_URL>
Replace <repository-url> with the URL of the repository you want to clone.
For example, to clone a repository named "example-repo" from GitHub, you would use a command like this:

git clone https://github.com/username/example-repo.git

This command will create a new directory named "example-repo" in your current directory and clone the repository into it. If you want to clone the repository into a directory with a different name, you can specify the directory name as an additional argument:

git clone https://github.com/username/example-repo.git my-custom-directory

This would clone the repository into a directory named "my-custom-directory" instead.
After cloning the repository, you can navigate into the directory and start working with the code just like any other Git repository on your local machine.

12. Add a Remote Repository:

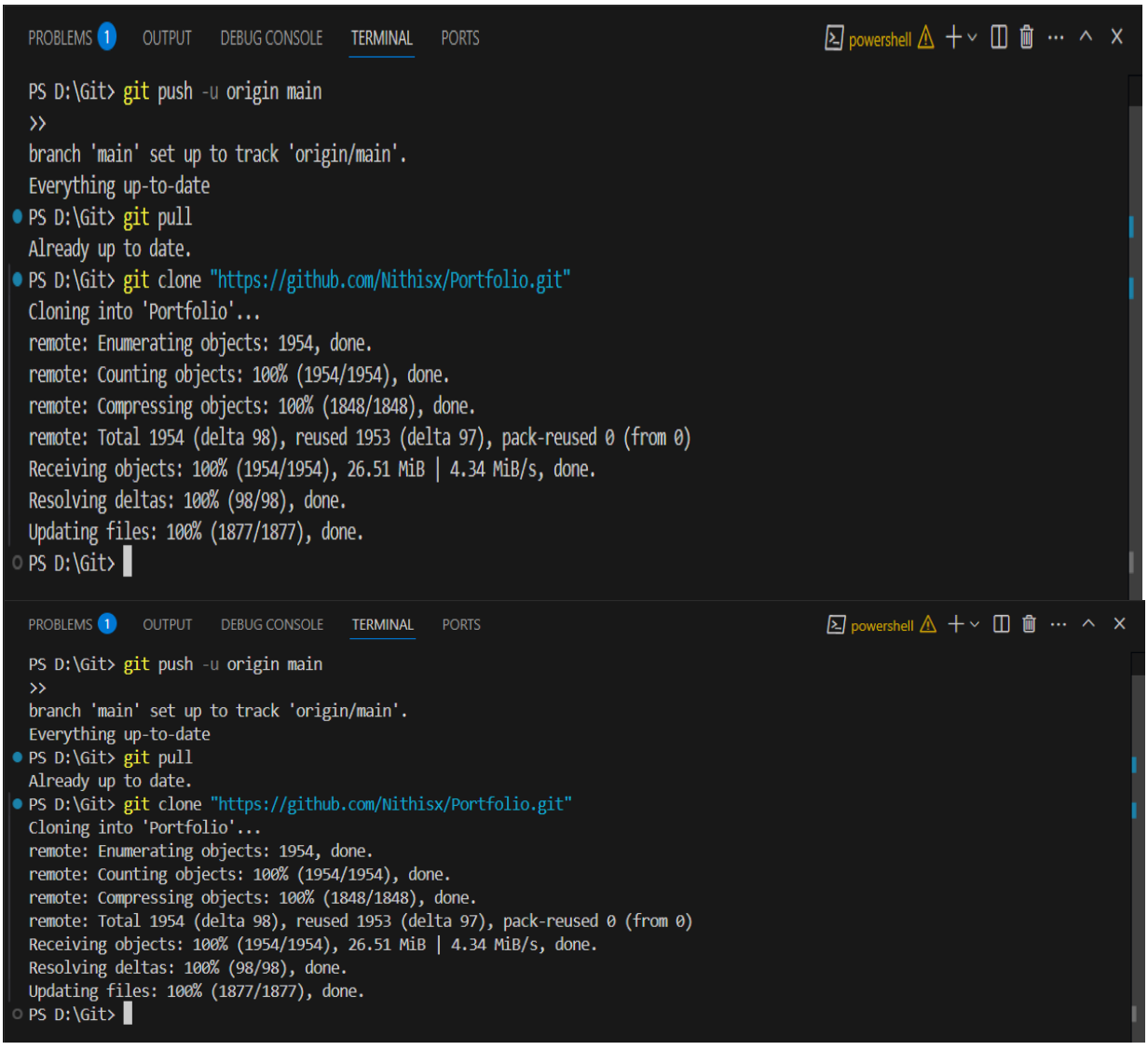
To add a remote repository to your local Git repository, you can use the git remote add command followed by a name for the remote and the URL of the remote repository.

git remote add <remote_name> <repository_URL>

Replace <remote-name> with a name for the remote (usually "origin" for the primary remote) and <repository-url> with the URL of the remote repository.
For example, to add a remote named "origin" for a repository hosted on GitHub, you would use a command like this:
git remote add origin https://github.com/username/example-repo.git

This command tells Git to add a remote named "origin" with the specified URL to your local repository. You can then use this remote to push and pull changes to and from the remote repository. After adding the remote repository, you can verify that it has been added correctly by running: git remote -v

OUTPUT:



RESULT:

Hence, some of the most common Git commands have been used for version control. With these commands, we can effectively manage changes to our project and collaborate with others.

Ex. No: 2 Programs using flow control statements, arrays and arrow functions.

AIM

To write and execute JavaScript programs using flow control statements, arrays, and arrow functions.

ALGORITHM:

Step 1: Create a JavaScript file named main.js.
Step 2: Implement flow control statements (if-else, switch, for, while).
Step 3: Declare and manipulate arrays (push, pop, map, filter).
Step 4: Write functions using arrow syntax.
Step 5: Run the JavaScript file using node main.js.
Step 6: Observe and verify the output.

PROGRAM

Program 1 Using Flow Control Statement

```
// Array of numbers
const numbers = [1, 2, 3, 4, 5];

// Function to filter even numbers using arrow function
const filterEvenNumbers = (arr) => arr.filter(num => num % 2 === 0);

// Function to calculate the sum of numbers using arrow function
const calculateSum = (arr) => arr.reduce((acc, curr) => acc + curr, 0);

// Function to find the maximum number using arrow function
const findMaxNumber = (arr) => Math.max(...arr);

// Function to determine if a number is prime using arrow function
const isPrime = (num) => {
    if (num <= 1) return false;
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) return false;
    }
    return true;
```

```
};

// Use of flow control statements and array methods
const evenNumbers = filterEvenNumbers(numbers);
const sum = calculateSum(numbers);
const maxNumber = findMaxNumber(numbers);

// Output
console.log("Original Array:", numbers);
console.log("Even Numbers:", evenNumbers);
console.log("Sum of Numbers:", sum);
console.log("Maximum Number:", maxNumber);

// Check if a number is prime and output the result
const numToCheck = 7;
if (isPrime(numToCheck)) {
   console.log(`${numToCheck} is a prime number.`);
} else {
   console.log(`${numToCheck} is not a prime number.`);
}
```

Program 2: Finding the Maximum Number in an Array

This program will use an arrow function and a flow control statement (a for loop) to find the maximum number in an array.

```
// Define an array of numbers
const numbers = [10, 5, 7, 23, 18, 4, 15];

// Define an arrow function to find the maximum number in the array
const findMax = (arr) => {
  let max = arr[0]; // Assume the first number is the maximum

  // Loop through the array
  for (let i = 1; i < arr.length; i++) {
    // If the current number is greater than the current maximum, update the maximum
    if (arr[i] > max) {
```

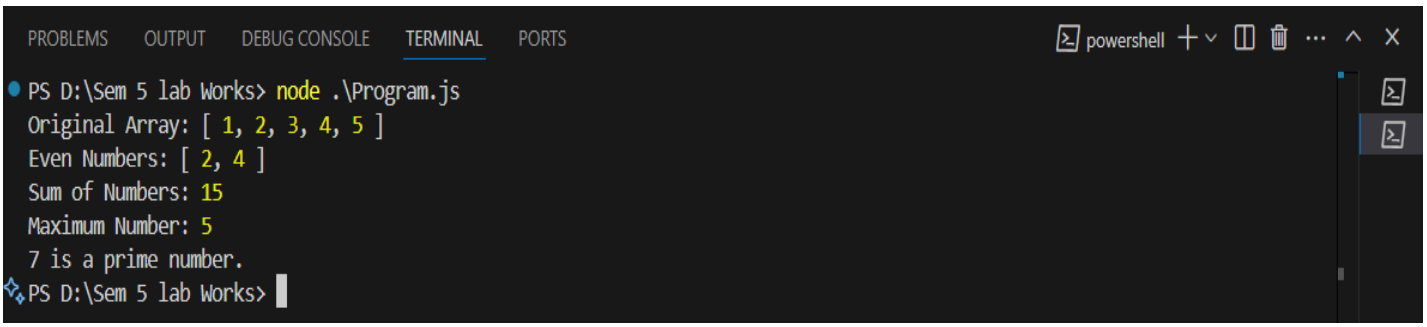
```
      max = arr[i];
    }
  }
  return max; // Return the maximum number
};
// Call the function and log the result
console.log("The maximum number in the array is:", findMax(numbers));
```

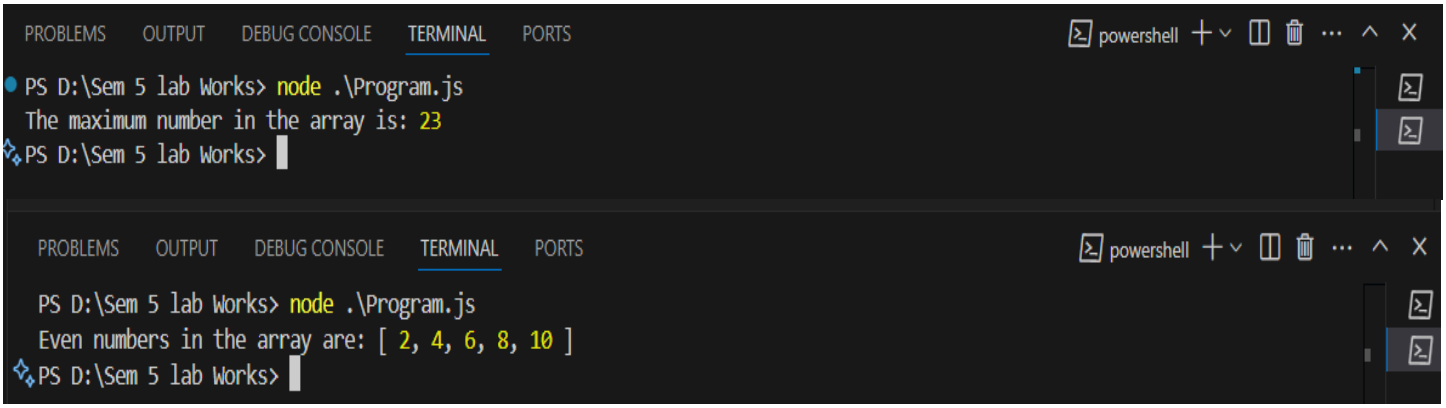
Program 3: Filtering Even Numbers from an Array

This program will use an arrow function and a flow control statement (a forEach loop) to filter even numbers from an array.

```
// Define an array of numbers
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
// Define an arrow function to filter even numbers from the array
const filterEvenNumbers = (arr) => {
  let evenNumbers = [];
  // Loop through the array using forEach
  arr.forEach((num) => {
    // Check if the number is even
    if (num % 2 === 0) {
      evenNumbers.push(num); // If even, add it to the evenNumbers array
    }
  });
  return evenNumbers; // Return the array of even numbers
};
// Call the function and log the result
console.log("Even numbers in the array are:", filterEvenNumbers(numbers));
```

OUTPUT:





RESULT:

Hence, JavaScript programs using flow control statements, arrays, and arrow functions have been successfully implemented and executed.

Ex. No: 3 Develop simple application using NodeJS.

AIM:

To develop a simple application using NodeJS.

ALGORITHM:

Step 1: Create a project folder and initialize it using npm init -y.
Step 2: Create a file named app.js.
Step 3: Import the required built-in modules (e.g., http, fs).
Step 4: Create a basic HTTP server and write a response.
Step 5: Start the server using node app.js.
Step 6: Open the browser and verify output on http://localhost:3000.

PROGRAM:

```
// Import the http module
const http = require('http');
// Define the hostname and port number
const hostname = '127.0.0.1';
const port = 3000;
// Create a HTTP server
const server = http.createServer((req, res) => {
  // Set the response HTTP header with status code 200 and Content-Type as text/plain
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Write the response body "Hello, world!"
  res.end('Hello, world!\n');
});
// Start the server and make it listen for incoming requests
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Save the file. Now, let's run the server. Open your terminal or command prompt, navigate to your project directory, and run the following command:

node app.js

You should see the message "Server running at http://127.0.0.1:3000/" printed in the terminal, indicating that your server is running.Now, open your web browser and navigate to http://127.0.0.1:3000/. You should see the message "Hello, world!" displayed in the browser.

OUTPUT:

OUTPUT:

RESULT:

Hence, a simple NodeJS application was successfully created and executed.

Ex. No: 4 Develop Rest API with NodeJS.

AIM:

To develop a simple REST API using NodeJS.

ALGORITHM:

Step 1: Create a project directory and initialize with npm init -y.
Step 2: Install Express using npm install express.
Step 3: Create a file server.js and import Express.
Step 4: Define routes for GET, POST, PUT, DELETE methods.
Step 5: Start the server using node server.js.
Step 6: Test the API using Postman or browser.

PROGRAM:

Let us create a new directory for your project and navigate into it using your terminal or command prompt.
Now, let's initialize a new Node.js project. Run the following command in your terminal:
npm init -y
This will create a package.json file with default values.
Now, let's install Express. Run the following command:
npm install express
Once Express is installed, create a new file called server.js in your project directory. Open this file in a text editor and add the following code:

```
// Import the Express framework
const express = require('express');
// Create an Express application
const app = express();
// Define a route for the root URL
app.get('/', (req, res) => {
  res.send('Hello, this is a simple REST API!');
});
// Define a route to handle GET requests to /api
app.get('/api', (req, res) => {
  // Define sample data
  const data = {
    message: 'Welcome to the API!',
    info: 'This is a sample response from the API.'
  };
```

```
  // Send the data as JSON
  res.json(data);
});
// Define a route to handle POST requests to /api
app.post('/api', (req, res) => {
  // Dummy response for POST requests
  res.send('POST request received!');
});
// Define a route to handle PUT requests to /api
app.put('/api', (req, res) => {
  // Dummy response for PUT requests
  res.send('PUT request received!');
});
// Define a route to handle DELETE requests to /api
app.delete('/api', (req, res) => {
  // Dummy response for DELETE requests
  res.send('DELETE request received!');
});
// Start the server and make it listen for incoming requests on port 3000
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

OUTPUT:

RESULT:

Hence, a REST API was developed using NodeJS and verified using API testing tools.

Ex. No: 5 Develop simple application using MongoDB

AIM:
To develop a simple application using MongoDB for database operations.
ALGORITHM:
Step 1: Install MongoDB and start the MongoDB server.
Step 2: Create a NodeJS project and install mongodb package using npm install mongodb.
Step 3: Create a file db.js and connect to MongoDB using MongoClient.
Step 4: Perform basic CRUD operations (Insert, Read, Update, Delete).
Step 5: Run the script using node db.js.
Step 6: Verify database changes using MongoDB Compass or terminal.

PROGRAM:
Step 1: Set Up the Node.js Project
Create a new directory for your project and navigate into it using your terminal or command prompt. Then, initialize a new Node.js project by running the following command:

```
npm init -y
```

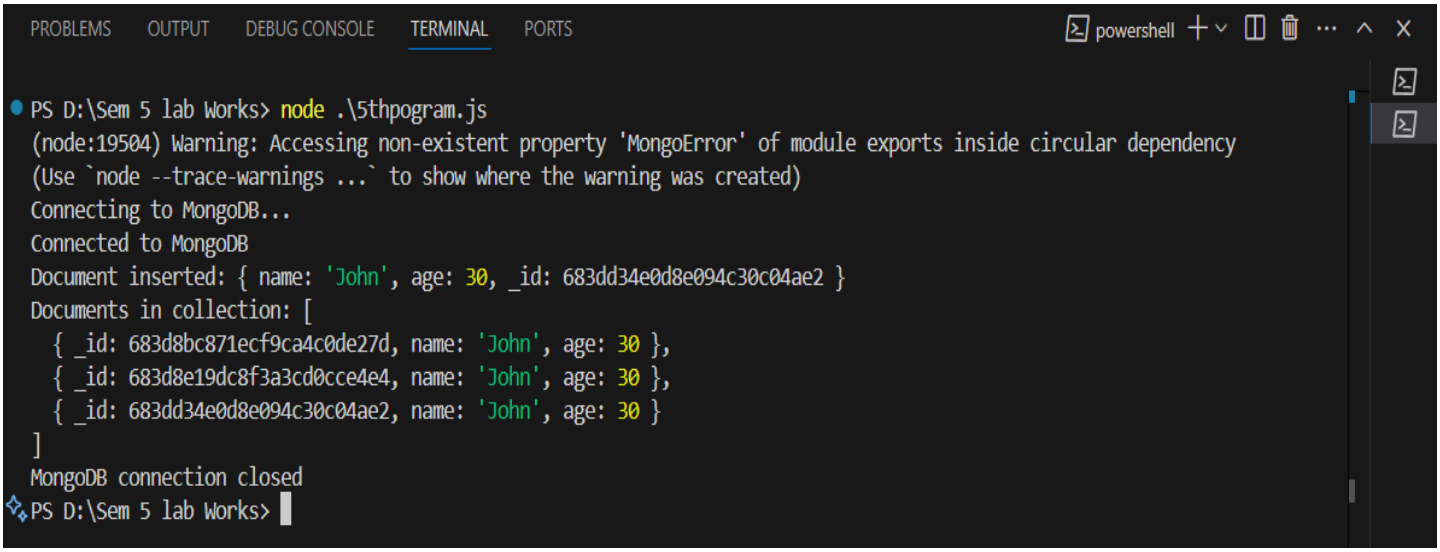
This will create a package.json file with default values.
Step 2: Install Dependencies
Next, install the MongoDB driver for Node.js. Run the following command:
npm install mongodb

Step 3: Create the Application
Now, let's create a file called app.js in your project directory. Open this file in a text editor and add the following code:

```
// Import the MongoDB client
const MongoClient = require('mongodb').MongoClient;

// MongoDB connection URI
const uri = "mongodb://localhost:27017";

// Create a new MongoClient
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

// Connect to MongoDB
```

```
client.connect((err) => {
  if (err) {
    console.error("Error connecting to MongoDB:", err);
    return;
  }

  console.log("Connected to MongoDB");

  // Get the database instance
  const db = client.db('test');

  // Define a collection (similar to a table in relational databases)
  const collection = db.collection('users');

  // Insert a document into the collection
  collection.insertOne({ name: 'John', age: 30 }, (err, result) => {
    if (err) {
      console.error("Error inserting document:", err);
      return;
    }
    console.log("Document inserted:", result.ops[0]);

    // Query the collection
    collection.find({}).toArray((err, documents) => {
      if (err) {
        console.error("Error querying collection:", err);
        return;
      }
      console.log("Documents in collection:", documents);
      // Close the MongoDB connection
      client.close();
      console.log("MongoDB connection closed");
    });
  });
});
```

OUTPUT:



RESULT:

Hence, a simple application using MongoDB for database operations was successfully created and executed.

Ex. No: 6 Develop Rest API with NodeJS and MongoDB

AIM:
To develop a REST API using NodeJS and MongoDB for backend operations.
ALGORITHM:
Step 1: Create a project folder and initialize with npm init -y.
Step 2: Install required packages using npm install express mongoose.
Step 3: Connect to MongoDB using mongoose.connect.
Step 4: Define schema and create models using Mongoose.
Step 5: Create RESTful routes (GET, POST, PUT, DELETE).
Step 6: Test the API endpoints using Postman.

PROGRAM:
Step 1: Set Up the Node.js Project
Create a new directory for your project and navigate into it using your terminal or command prompt. Then, initialize a new Node.js project by running:
npm init -y
Step 2: Install Dependencies
Install Express.js and the MongoDB Node.js driver by running:
npm install express mongodb
Step 3: Create the Express Server
Create a file named server.js and add the following code:

```
const express = require('express');
const { MongoClient, ObjectID } = require('mongodb');
const app = express();
const port = 3000;
const mongoURI = 'mongodb://localhost:27017';
const dbName = 'test';
const collectionName = 'todos';
app.use(express.json());
MongoClient.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true }, (err,
client) => {
  if (err) {
    console.error('Error connecting to MongoDB:', err);
    return;
  }
  console.log('Connected to MongoDB');
```

```
const db = client.db(dbName);
const collection = db.collection(collectionName);
// GET all todos
app.get('/api/todos', async (req, res) => {
  try {
    const todos = await collection.find({}).toArray();
    res.json(todos);
  } catch (err) {
    console.error('Error fetching todos:', err);
    res.status(500).json({ error: 'Failed to fetch todos' });
  }
});
// POST a new todo
app.post('/api/todos', async (req, res) => {
  const todo = req.body;
  try {
    const result = await collection.insertOne(todo);
    res.json(result.ops[0]);
  } catch (err) {
    console.error('Error creating todo:', err);
    res.status(500).json({ error: 'Failed to create todo' });
  }
});
// DELETE a todo
app.delete('/api/todos/:id', async (req, res) => {
  const id = req.params.id;
  try {
    const result = await collection.deleteOne({ _id: ObjectID(id) });
    if (result.deletedCount === 0) {
      res.status(404).json({ error: 'Todo not found' });
    } else {
      res.json({ message: 'Todo deleted successfully' });
    }
  } catch (err) {
    console.error('Error deleting todo:', err);
    res.status(500).json({ error: 'Failed to delete todo' });
  }  });
```

```
  // Start the server
  app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
  });
});
```

OUTPUT:

RESULT:

Hence, a REST API integrating NodeJS and MongoDB was successfully developed and tested.

Ex. No: 7 Develop simple application using ReactJS.

AIM:

To develop simple application using ReactJS

ALGORITHM:

Step 1: Setup Environment and install Node.js and npm (Node Package Manager).

Step 2: Open command prompt or terminal and navigate to Project Directory

Step 3: Launch the development server

Step 4: Edit App Component

Step 5: Add basic CSS styling.

Step 6: Stop the development server

PROGRAM:

First, ensure Node.js installed on the system. Once Node.js is installed, create a new React application using Create React App, which is a popular tool for generating React projects with a pre-configured setup. Open your terminal or command prompt and run the following command:

npx create-react-app todo-app

This command will create a new directory called todo-app with all the files and configurations needed for a React application. Navigate into the newly created directory:

cd todo-app

Now, let's modify the generated code to create a simple to do list application.

i. **Modify App.js**

Open the src/App.js file in the text editor and replace its content with the following code:

```
import React, { useState } from 'react';
import './App.css';
function App() {
```

```
  const [todos, setTodos] = useState([]);
  const [inputValue, setInputValue] = useState('');
  const addTodo = () => {
    if (inputValue.trim() !== '') {
      setTodos([...todos, inputValue]);
      setInputValue('');
    }
  };
  const deleteTodo = (index) => {
    const updatedTodos = todos.filter((_, i) => i !== index);
    setTodos(updatedTodos);
  };
  return (
    <div className="App">
      <h1>Todo List</h1>
      <div className="todo-input">
        <input
          type="text"
          value={inputValue}
          onChange={(e) => setInputValue(e.target.value)}
          placeholder="Enter a new todo"
        />
        <button onClick={addTodo}>Add</button>
      </div>
      <ul className="todo-list">
        {todos.map((todo, index) => (
          <li key={index}>
            {todo}
            <button onClick={() => deleteTodo(index)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

ii. Modify App.css

Open the src/App.css file and replace its content with the following CSS:

```
.App {
  text-align: center;
  margin-top: 50px;
}
.todo-input {
  margin-bottom: 20px;
}
.todo-list {
  list-style: none;
  padding: 0;
}
.todo-list li {
  margin-bottom: 10px;
}
.todo-list button {
  margin-left: 10px;
}
```

Now, let's run the application using terminal or command prompt:

npm start

This command will start the development server and open the to do list application in your default web browser.

OUTPUT:

RESULT:

Thus, a new React application using ReactJS has been created and executed successfully.

Ex. No: 8 Develop simple application using ReactJS Components.

AIM:

To develop simple application using ReactJS Components

ALGORITHM:

Step 1: Initialize ReactJS Project

Step 2: Navigate to Project Directory

Step 3: Create New ReactJS Component

Step 4: Edit and Add the App Component

Step 5: Start React Development Server to view output

PROGRAM:

First, ensure that Node.js has been installed on your system.
Once Node.js is installed, create a new React application using Create React App, which is a popular tool for generating React projects with a pre-configured setup. Open the terminal or command prompt and run the following command:

npx create-react-app counter-app

This command will create a new directory called counter-app with all the files and configurations needed for a React application. Navigate into the newly created directory:

cd counter-app

Now, let's modify the generated code to create the counter application.

i. **Modify App.js**

Open the src/App.js file in your text editor and replace its content with the following code:

```
import React, { useState } from 'react';
import './App.css';
function Counter() {
  const [count, setCount] = useState(0);
```

```
  const increment = () => {
    setCount(count + 1);
  };
  const decrement = () => {
    setCount(count - 1);
  };

  return (
    <div className="counter">
      <h2>Counter: {count}</h2>
      <div className="buttons">
        <button onClick={increment}>Increment</button>
        <button onClick={decrement}>Decrement</button>
      </div>
    </div>
  );
}
function App() {
  return (
    <div className="App">
      <h1>Simple Counter App</h1>
      <Counter />
    </div>
  );
}
export default App;
```

ii. Modify App.css

Open the src/App.css file and replace its content with the following CSS:

```
.App {
  text-align: center;
  margin-top: 50px;
}
.counter {
  margin-bottom: 20px;
}
.buttons button {
```

```
  margin-left: 10px;
}
```

Now, let's run the application in the terminal or command prompt:

```
npm start
```

OUTPUT:

RESULT:

Thus, a simple counter application using React.js components has been executed successfully and it can be further customize and expand upon this application by adding more features and functionality as needed.

Ex.No: 9 Develop simple application using React Context with styles

AIM:

To develop simple application using ReactJS context with styles

ALGORITHM:

Step 1: Initialize React Project

Step 2: Navigate to Project Directory

Step 3: Create Context and Provider Component

Step 4: Create a New Component to Use Context

Step 5: Add Styling to the Component

Step 6: Use Context Provider in App Component

Step 7: Start Server and View Output

PROGRAM:

Open the terminal or command prompt and run the following command:

npx create-react-app theme-toggler-app

This command will create a new directory called theme-toggler-app with all the files and configurations needed for a React application. Navigate into the newly created directory:

cd theme-toggler-app

Now, let's modify the generated code to create our theme toggler application.

i. **Modify App.js**

Open the src/App.js file in the text editor and replace its content with the following code:

```
import React, { createContext, useContext, useState } from 'react';
import './App.css';
// Create a theme context with default value 'light'
const ThemeContext = createContext('light');
```

```
function ThemeProvider({ children }) {
  const [theme, setTheme] = useState('light');
  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));
  };
  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}
function ThemeToggler() {
  const { theme, toggleTheme } = useContext(ThemeContext);
  return (
    <button className="theme-toggler" onClick={toggleTheme}>
      {theme === 'light' ? 'Switch to Dark Theme' : 'Switch to Light Theme'}
    </button>
  );
}
function App() {
  return (
    <ThemeProvider>
      <div className="App">
        <h1>Theme Toggler App</h1>
        <ThemeToggler />
        <div className="content">
          <h2>Welcome to the {useContext(ThemeContext).theme} theme!</h2>
          <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis
            dapibus risus non risus convallis, nec semper risus dignissim.
          </p>
        </div>
      </div>
    </ThemeProvider>
  );
}
export default App;
```

ii. Modify App.css

Open the src/App.css file and replace its content with the following CSS:

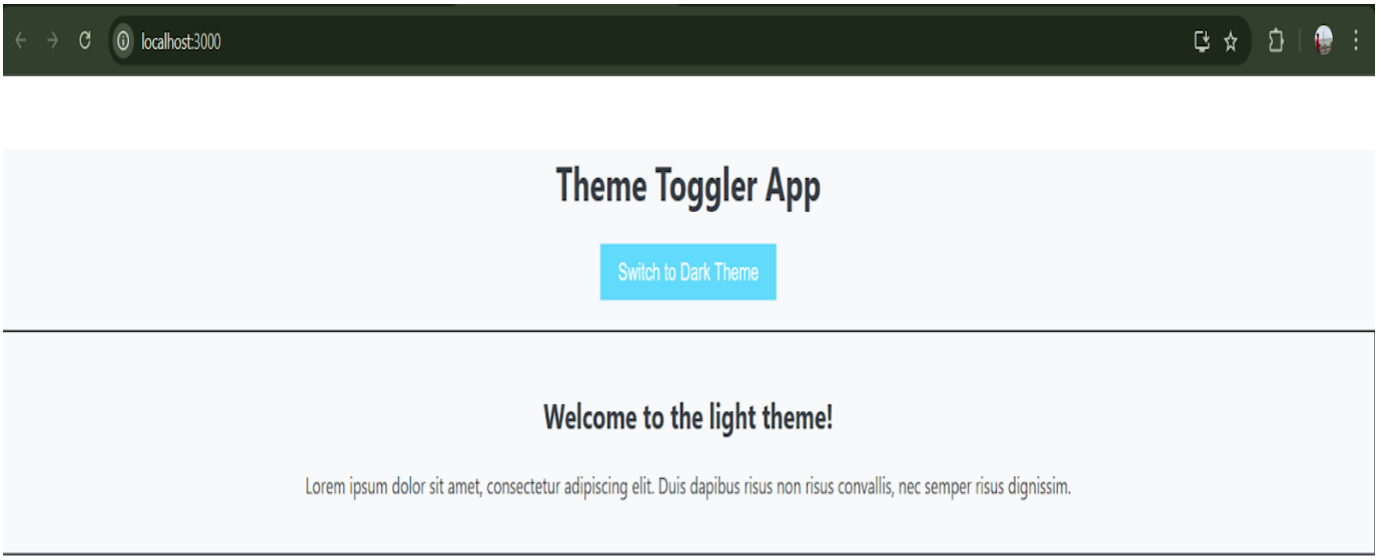
```
.App {
  text-align: center;
  margin-top: 50px;
}
.theme-toggler {
  padding: 10px 20px;
  background-color: #61dafb;
  border: none;
  color: #fff;
  font-size: 16px;
  cursor: pointer;
}
.theme-toggler:hover {
  background-color: #007bff;
}
.content {
  margin-top: 20px;
  padding: 20px;
  border: 2px solid;
}
.light {
  background-color: #f8f9fa;
  color: #343a40;
}
.dark {
  background-color: #343a40;
  color: #f8f9fa;
}
```

Now, let's run the application in the terminal or command prompt:

npm start

This command will start the development server and open the theme toggler application in your default web browser.

OUTPUT:



RESULT:

Thus, a simple theme toggler application with a button to switch between light and dark themes using React Context with styles has been developed and executed successfully.

Ex. No: 10 Develop Rest API with Axios

AIM:

To develop simple rest API with Axios

ALGORITHM:

Step 1: Set Up React Application

Step 2: Install Axios Library

Step 3: Create an API Service File

Step 4: Create a Component to Call API

Step 5: Integrate the Component in App.js

Step 6: Run the React Application

PROGRAM:

To develop a REST API client using Axios in a React application, consume an existing one. Axios is a popular HTTP client library for making requests to APIs from JavaScript applications.
REST API using Axios:

i. Set Up a React Application

Create React App:
npx create-react-app axios-rest-api
cd axios-rest-api

ii. Install Axios

Install Axios in your React application:
npm install axios

iii. Create a Component to Fetch Data

Create a new component, for example DataFetcher.js, inside the src directory:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
```

```
function DataFetcher() {
  const [data, setData] = useState([]);
  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => {
        setData(response.data);
      })
      .catch(error => {
        console.error('Error fetching data:', error);
      });
  }, []);

  return (
    <div>
      <h2>Posts</h2>
      <ul>
        {data.map(item => (
          <li key={item.id}>{item.title}</li>
        ))}
      </ul>
    </div>
  );
}

export default DataFetcher;
```

iv. Use the Component in App.js

Import and use the DataFetcher component in your App.js:

```
import React from 'react';
import './App.css';
import DataFetcher from './DataFetcher';
function App() {
  return (
    <div className="App">
      <h1>Fetching Data with Axios</h1>
      <DataFetcher />
    </div>
```

```
  );
}

export default App;
```

Run the Application
Start your React application:
npm start

OUTPUT:

RESULT:
Thus, a simple React application that consumes a REST API using Axios has been developed and executed.

Ex. No: 11 Develop Rest API with react-query and SWR

AIM:

To develop simple rest API with react-query and SWR

ALGORITHM:

Step 1: Set Up the Backend (Node.js & Express)

Step 2: Test API with Postman

Step 3: Set Up the Frontend (React App)

Step 4: Create Data Fetching Functions

Step 5: Use React-Query to Fetch Data

Step 6: Use SWR to Fetch Data in another Component

Step 7: Test and Compare React-Query and SWR

PROGRAM:

First, install react-query:
npm install react-query

Steps to use react-query to fetch data from a REST API:

```
import React from 'react';
import { useQuery } from 'react-query';

const fetchPosts = async () => {
  const response = await fetch('https://jsonplaceholder.typicode.com/posts');
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
};
function Posts() {
  const { data, status } = useQuery('posts', fetchPosts);
  if (status === 'loading') return <div>Loading...</div>;
  if (status === 'error') return <div>Error fetching data</div>;
  return (
    <div>
      <h2>Posts</h2>
      <ul>
```

```
        {data.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
export default Posts;
```

Then, in the main component, use the Posts component:

```
import React from 'react';
import Posts from './Posts';
function App() {
  return (
    <div className="App">
      <h1>Fetching Data with react-query</h1>
      <Posts />
    </div>
  );
}
export default App;
```

Using SWR
First, install swr:
npm install swr

Steps to use SWR to fetch data from a REST API:

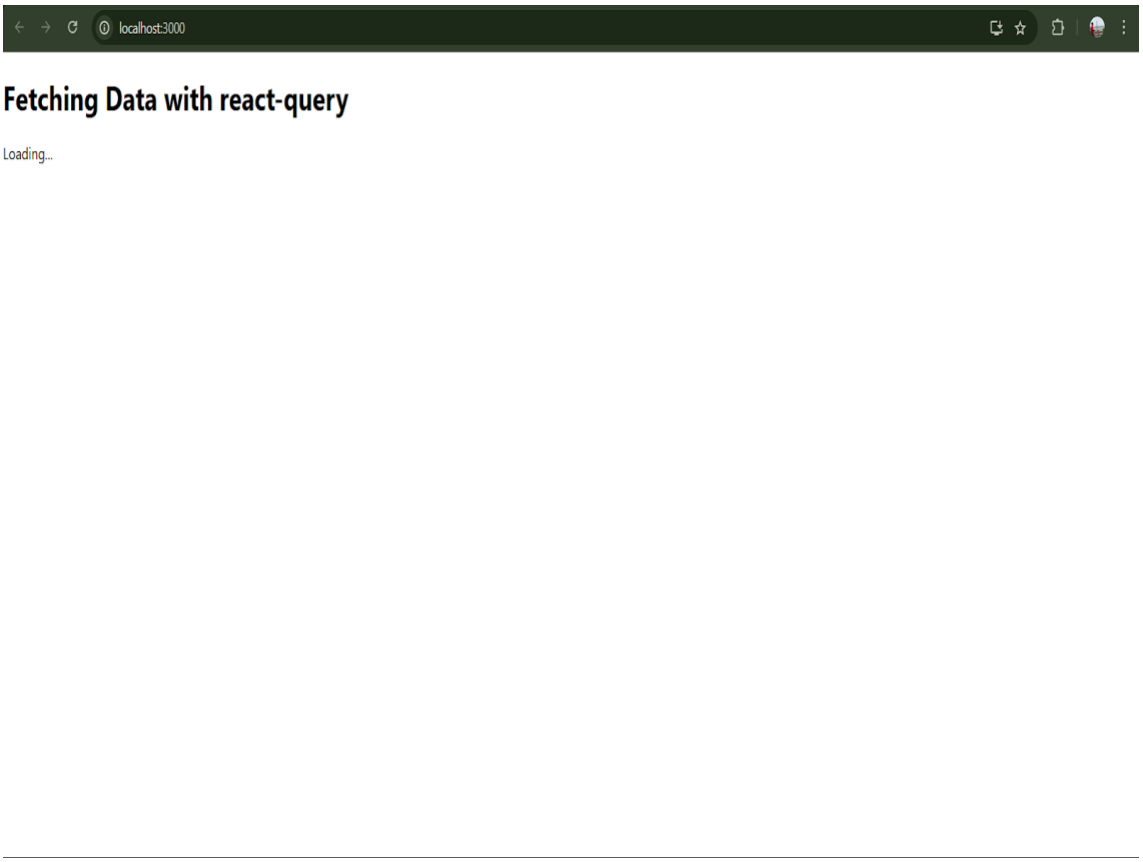
```
import React from 'react';
import useSWR from 'swr';

const fetcher = async (url) => {
  const response = await fetch(url);
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
```

```
  return response.json();
};

function Posts() {
  const { data, error } = useSWR('https://jsonplaceholder.typicode.com/posts', fetcher);

  if (error) return <div>Error fetching data</div>;
  if (!data) return <div>Loading...</div>;

  return (
    <div>
      <h2>Posts</h2>
      <ul>
        {data.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
export default Posts;
```

Posts component:

```
import React from 'react';
import Posts from './Posts';

function App() {
  return (
    <div className="App">
      <h1>Fetching Data with SWR</h1>
      <Posts />
    </div>
  );
}

export default App;
```

OUTPUT:



RESULT:

Thus, a rest API with react-query and SWR has been developed and executed successfully.

Ex.No: 12 Developing full stack application using ReactJS and MongoDB.

AIM:

To develop full stack application using ReactJS and MongoDB

ALGORITHM:

Step 1: Set Up the Backend (Node.js & Express)

Step 2: Connect to MongoDB

Step 3: Create REST API Endpoints

Step 4: Set Up the Frontend with ReactJS

Step 5: Build Frontend Components and Connect to API

Step 6: Run Both Frontend and Backend

PROGRAM:

To develop a full-stack application using React.js and MongoDB, you'll need to create both the frontend (using React.js) and the backend (using Node.js with Express) components. MongoDB will serve as the database to store and retrieve data.
Here's a basic outline of the steps involved:

i. Set Up the Backend (Node.js with Express and MongoDB)
ii. Install Node.js if you haven't already.
iii. Set up a new Node.js project:

mkdir backend
cd backend
npm init -y

iv. Install necessary dependencies:

npm install express mongoose body-parser cors

v. Create a MongoDB Atlas account (or use a local MongoDB installation).
vi. Create a MongoDB database and get the connection URI.

Backend server using Express and MongoDB:

// server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const port = process.env.PORT || 5000;
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/myapp', { useNewUrlParser: true,
useUnifiedTopology: true });
const connection = mongoose.connection;
connection.once('open', () => {
  console.log('MongoDB database connection established successfully');
});

// Define routes
const todoRouter = require('./routes/todo');
app.use('/todos', todoRouter);

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port: ${port}`);
});
```

Create a models directory to define MongoDB schema and a routes directory to define API routes.

i. **Create a new React.js project:**

```
npx create-react-app frontend
cd frontend
```

ii. **Install Axios for making HTTP requests:**

```
npm install axios
```

iii. **Replace the default src folder with your React components.**

iv. **Start your backend server:**

```
node server.js
```

v. Start your React frontend:

npm start

OUTPUT:

RESULT:

Thus, the full-stack application with React.js handling the frontend and communicating with the Express backend, which in turn interacts with the MongoDB database has been developed successfully.

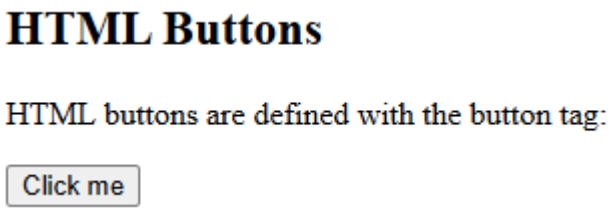
Additional Experiments

1. Write simple HTML code to create buttons and its functions

PROGRAM

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Buttons</h2>
<p>HTML buttons are defined with the button tag:</p>
<button>Click me</button>
</body>
</html>
```

OUTPUT:



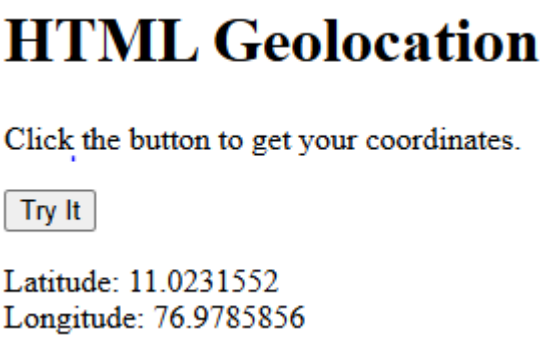
2. Write HTML Geolocation API to return the latitude and longitude of the user's current location

PROGRAM

```
<!DOCTYPE html>
<html>
<body>
<h1>HTML Geolocation</h1>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>
<script>
const x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(success, error);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
```

```
function success(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
function error() {
  alert("Sorry, no position available.");
}
</script>
</body>
</html>
```

OUTPUT:

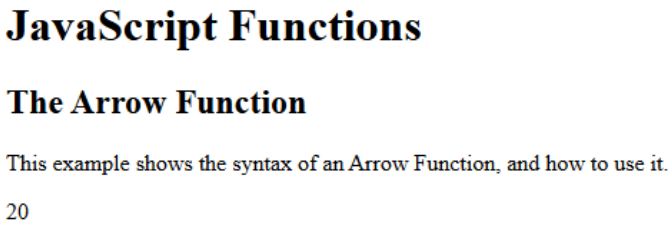


3. Write a simple JavaScript code to use arrow function

PROGRAM:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>
<h2>The Arrow Function</h2>
<p>This example shows the syntax of an Arrow Function, and how to use
it.</p>
<p id="demo"></p>
<script>
let myFunction = (a, b) => a * b;
document.getElementById("demo").innerHTML = myFunction(4, 5);
</script>
</body>
</html>
```

OUTPUT:

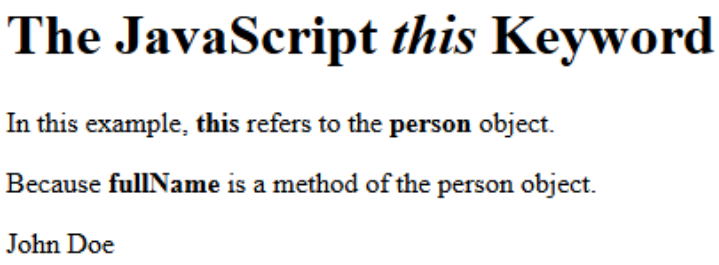


4. Write simple JavaScript code to illustrate the purpose of this keyword

PROGRAM:

```
<!DOCTYPE html>
<html>
<body>
<h1>The JavaScript <i>this</i> Keyword</h1>
<p>In this example, <b>this</b> refers to the <b>person</b> object.</p>
<p>Because <b>fullName</b> is a method of the person object.</p>
<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```

OUTPUT:



5. Write a javascript code for exploring the purpose of JavaScript Object Notation

PROGRAM:

```
<!DOCTYPE html>
<html>
<body>
<h2>Create Object from JSON String</h2>
<p id="demo"></p>
<script>
let text = '{"employees":[' +
```

```
'{"firstName":"John","lastName":"Doe" },' +
'{"firstName":"Anna","lastName":"Smith" },' +
'{"firstName":"Peter","lastName":"Jones" }]}';
const obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

OUTPUT:

Create Object from JSON String

Anna Smith

6. Write HTML code to define various colors

PROGRAM:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="background-color:Tomato;">Tomato</h1>
<h1 style="background-color:Orange;">Orange</h1>
<h1 style="background-color:DodgerBlue;">DodgerBlue</h1>
<h1style="background-color:MediumSeaGreen;">MediumSeaGreen</h1>
<h1 style="background-color:Gray;">Gray</h1>
<h1 style="background-color:SlateBlue;">SlateBlue</h1>
<h1 style="background-color:Violet;">Violet</h1>
<h1 style="background-color:LightGray;">LightGray</h1>
</body>
</html>
```

OUTPUT:

Tomato

Orange

DodgerBlue

MediumSeaGreen

Gray

SlateBlue

Violet

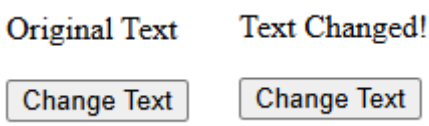
LightGray

7. Create a webpage that changes the text of a paragraph when a button is clicked

PROGRAM:

```
<!DOCTYPE html>
<html>
<head><title>Change Text</title></head>
<body>
  <p id="myText">Original Text</p>
  <button onclick="changeText()">Change Text</button>
  <script>
    function changeText() {
      document.getElementById('myText').innerText = 'Text Changed!';
    }
  </script>
</body>
</html>
```

OUTPUT:

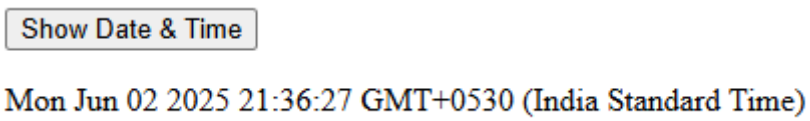


8. Create a webpage that displays the current date and time when a button is clicked

PROGRAM:

```
<!DOCTYPE html>
<html>
<head><title>Show Date</title></head>
<body>
  <button onclick="showDate()">Show Date & Time</button>
  <p id="dateTime"></p>
  <script>
    function showDate() {
      document.getElementById('dateTime').innerText = new Date().toString();
    }
  </script>
</body>
</html>
```

OUTPUT:

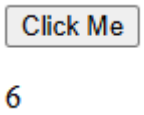


9. Create a webpage that counts the number of times a button is clicked

PROGRAM:

```
<!DOCTYPE html>
<html>
<head><title>Click Counter</title></head>
<body>
  <button onclick="countClicks()">Click Me</button>
  <p id="counter">0</p>
  <script>
    let count = 0;
    function countClicks() {
      count++;
      document.getElementById('counter').innerText = count;
    }
  </script>
</body>
</html>
```

OUTPUT:



10. Write simple JavaScript code to describe Operator Precedence

PROGRAM:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arithmetic</h1>
<h2>Operator Precedence</h2>
<p>Multiplication has precedence over addition.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 100 + 50 * 3;
</script>
</body>
</html>
```

OUTPUT: