Sri **SAI RAM ENGINEERING COLLEGE**

An Autonomous Institution | Affiliated to Anna University & Approved by AICTE, New Delhi

Accredited by NBA and NAAC "A+" | BIS/EOMS ISO 21001 : 2018 and BVQI 9001 : 2015 Certified and NIRF ranked institution

Sai Leo Nagar, West Tambaram, Chennai - 600 044. www.sairam.edu.in

PROSPERITY THROUGH TECHNOLOGY

24AMPT401 – WEB DEVELOPMENT FRAMEWORK LABORATORY WITH THEORY

IV Semester CSE (AI&ML)

Academic year: 2025 - 2026

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

LAB MANUAL

PREFACE

“PRACTICE LEADS TO PERFECTION”

Practical work encourages and familiarizes students with the latest tools that will be required to become experts. Taking this into consideration, this manual is compiled as a preparatory note for the Database Management Systems Laboratory programs. Sufficient details have been included to impart self- learning.

This manual is intended for the III semester CSE (AI&ML) students under Autonomous Regulations 2020. Introductory information and procedure to perform is detailed in this manual.

It is expected that this will help the students develop a broad understanding and learn quick adaptations to real-time problems.

INSTITUTE VISION

To emerge as a “Centre of excellence” offering Technical Education and Research opportunities of very high standards to students, develop the total personality of the individual and instill high levels of disciple and strive to set global standards, making our students technologically superior and ethically stronger, who in turn shall contribute to the advancement of society and humankind.

INSTITUTION MISSION

We dedicate and commit ourselves to achieve, sustain and foster unmatched excellence in Technical Education. To this end, we will pursue continuous development of infra-structure and enhance state-of-art equipment to provide our students a technologically up-to date and intellectually inspiring environment of learning, research, creativity, innovation and professional activity and inculcate in them ethical and moral values.

INSTITUTE POLICY

We at Sri Sai Ram Engineering College are committed to build a better Nation through Quality Education with team spirit. Our students are enabled to excel in all values of Life and become Good Citizens. We continually improve the System, Infrastructure and Service to satisfy the Students, Parents, Industry and Society.

DEPARTMENT VISION

To emerge as a “Centre of Excellence” in the field of Artificial Intelligence and Machine Learning by providing required skill sets, domain expertise and interactive industry interface for students and shape them to be a socially conscious and responsible citizen.

DEPARTMENT MISSION

Computer Science and Engineering (Artificial Intelligence and Machine Learning), Sri Sairam Engineering College is committed to

M1: Nurture students with a sound understanding of fundamentals, theory and practice of Artificial Intelligence and Machine Learning.

M2: Develop students with the required skill sets and enable them to take up assignments in the field of AI & ML

M3: Facilitate Industry Academia interface to update the recent trends in AI &ML

M4: Create an appropriate environment to bring out the latent talents, creativity and innovation among students to contribute to the society

Program Educational Objectives (PEOs)

To prepare the graduates to:

1. Graduates imbibe fundamental knowledge in Artificial Intelligence, Programming, Mathematical modelling and Machine Learning.
2. Graduates will be trained to gain domain expertise by applying the theory basics into practical situation through simulation and modelling techniques.
3. Graduates will enhance the capability through skill development and make them industry ready by inculcating leadership and multitasking abilities

4. Graduates will apply the gained knowledge of AI & ML in Research & Development, Innovation and contribute to the society in making things simpler.

Program Outcomes (PO's)

- PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

- PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

- PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

- PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

- PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

- PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

- PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
- PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
- PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences
- PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
- PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

Program Specific Outcomes (PSOs)

Computer Science and Engineering (Artificial Intelligence and Machine Learning), graduates will be able to:

- The graduates will be in a position to design, develop, test and deploy appropriate mathematical and programming algorithms required for practical applications
- The graduates will have the required skills and domain expertise to provide solutions in the field of Artificial Intelligence and Machine Learning for the Industry and society at large.

COURSE OUTCOMES

Upon completion of the course, the students will be able to

CO1	Apply HTML and CSS to create interactive and dynamic websites. (K3)
CO2	Use JavaScript to develop interactive and dynamic website features. (K3)
CO3	Design and implement dynamic server-side applications using PHP. (K3)
CO4	Analyze Node.js and its features for efficient application development.(K4)
CO5	Develop applications using MongoDB for database management and storage. (K3)
CO6	Analyze and evaluate the features of Express and React for building web applications. (K4)

24AMPT401 – WEB DEVELOPMENT FRAMEWORK LABORATORY WITH THEORY

Syllabus

OBJECTIVES:

- To apply HTML and CSS to design and develop interactive web pages.
- To use scripting languages like JavaScript to design interactive and dynamic web pages.
- To analyze the features and applications of Node.js for efficient web application development.
- To develop applications using MongoDB for efficient data storage and retrieval.
- To understand the role of Express and React in the development of modern web applications.
- To design and implement full-stack web applications using the MERN stack.

LIST OF EXPERIMENTS:

- How to create a simple user registration form using HTML form elements. What are the common form elements used, and provide an example with Code?
- Explain how to style a web page using the CSS Box Model and background properties. Describe the components of the box model and list various CSS background properties with examples. Provide sample CSS code to demonstrate their usage.
- How to implement CSS animations and create a multiple column layoutusing CSS. Describe the key properties used for animations and the methods to divide content into columns. Provide suitable examples with HTML and CSS code.
- Write a JavaScript program that declares and initializes variables using different data types, including string, number, Boolean, array, and object. Explain each data type briefly and provide example values.
- Write a JavaScript function that accepts an array of numbers as its parameter. The function should iterate through the array, identify all even numbers, and return the sum of those even numbers. Provide the complete code and explain how the function works.
- Design and create an HTML form that includes input fields for Name, Email, and Age. Ensure each field uses the appropriate input type and includes basic validation. Provide the complete HTML code.
- Write a PHP script that declares and initializes variables using different data types, including integer, string, float, and Boolean. Also, define a constant using the appropriate PHP function. Explain each data type with example values used in your script.
- Write a PHP function that accepts an array of numbers as input and returns the

maximum value from the array. Explain how the function works and test it with a sample array.

- Write a PHP script to handle the file upload process, ensuring that only image files (JPEG, PNG) are allowed. Display a message confirming successful upload and save the file in a specific directory.
- Write a simple "Hello, World!" program using Node.js. Save the program in a .js file and run it using the command line. Describe the steps involved in writing, saving, and executing the Node.js script.
- Write a Node.js program that creates an EventEmitter, defines a custom event called "welcome", and triggers it to display "Welcome to Node.js!" after 3 seconds using setTimeout().
- Write a Node.js function that accepts two numbers and a callback function as arguments. The callback function should return the sum of the two numbers. Explain how the function works, how the callback is used, and provide a sample program to demonstrate its functionality.
- Write a Node.js script that reads the contents of a text file (e.g., sample.txt) and displays the content in the console. Use the fs module to handle file operations. Explain how the fs module works in Node.js and describe the steps involved in reading a file asynchronously.
- Install MongoDB on your system and start the MongoDB server. Then, use the MongoDB shell to create a new database named studentDB. Describe the steps for installation, starting the server, and creating the database.
- In the database studentDB, create a collection called students and insert a document with the following details: {“name": "John Doe", "age": 20, "course": "Computer Science”}. Describe the steps for creating a collection and inserting the document using MongoDB shell.
- Write a simple Node.js script to connect to a MongoDB database and retrieve all documents from the student’s collection using the MongoDB Node.js driver. Explain the steps involved in establishing the connection and retrieving data from the collection.
- Install Express.js and write a Node.js script to create a basic Express server that listens on port 3000. The server should respond with "Hello, Express!" when accessed at the root URL (/). Explain the steps for setting up Express and creating the server.
- Create a simple React application with a component named Counter. The component should have a button that, when clicked, increases a count state and displays the updated count. Explain the steps to set up the React application and implement the Counter component.
- Set up React Router in a React application and create two pages: "Home" and "About". Use navigation links to switch between these pages without refreshing the page. Explain the process of installing React Router, setting up routes, and handling navigation between pages.
- Create a React component named ToggleText that initially displays the text "Hello!". When a button is clicked, the component should toggle between displaying "Hello!" and "Goodbye!". Explain how to implement the functionality using React's state and event handling.

INDEX

Ex No 1 Creation of User Registration Form using HTML

Aim

To create a simple user registration form using basic HTML form elements to collect user details such as name, email, password, gender, and date of birth

Procedure

An HTML form is used to collect user input and send it to a server for processing. Forms are created using the <form> tag and consist of various form elements like text boxes, radio buttons, checkboxes, and buttons.

Common HTML Form Elements

- <form> – Defines the form container
- <input> – Used for different input types (text, password, email, radio, checkbox, submit)
- <label> – Describes an input field
- <select> – Creates a drop-down list
- <option> – Defines options inside a drop-down
- <textarea> – Allows multi-line text input
- <button> – Creates a clickable button

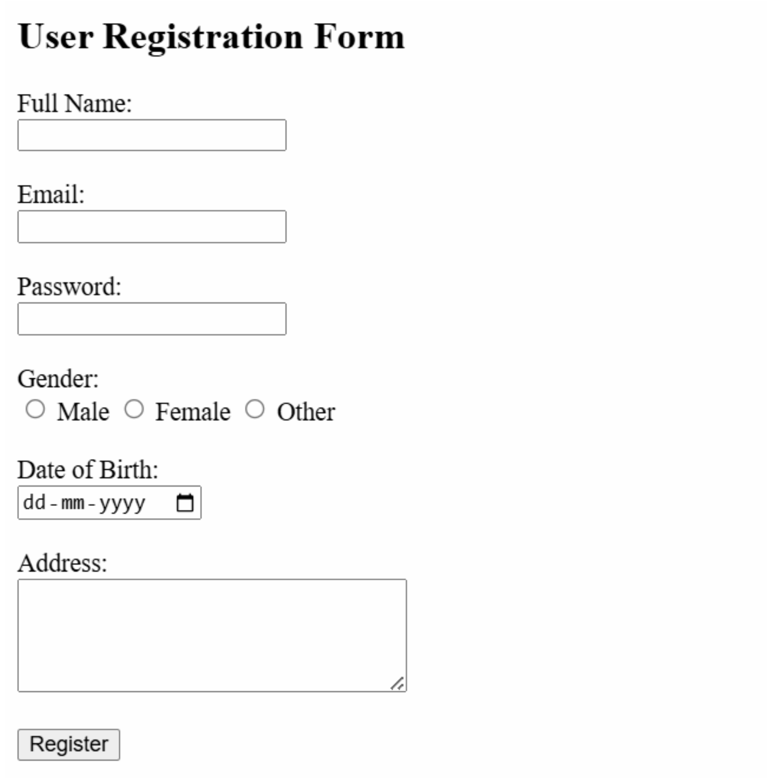
Steps to be followed

1. Open a text editor (Notepad / VS Code).
2. Create a new file and save it as registration.html.
3. Write the basic HTML structure.
4. Use the <form> tag to create a registration form.
5. Add required input fields such as name, email, password, gender, and date of birth.
6. Include a submit button.
7. Save the file and open it in a web browser to view the form.

Program

```
<!DOCTYPE html>
<html>
<head>
<title>User Registration Form</title>
</head>
<body>
<h2>User Registration Form</h2>
<form>
<label>Full Name:</label><br>
<input type="text" name="fullname" required><br><br>
<label>Email:</label><br>
<input type="email" name="email" required><br><br>
<label>Password:</label><br>
<input type="password" name="password" required><br><br>
<label>Gender:</label><br>
<input type="radio" name="gender" value="Male"> Male
<input type="radio" name="gender" value="Female"> Female
<input type="radio" name="gender" value="Other"> Other<br><br>
<label>Date of Birth:</label><br>
<input type="date" name="dob"><br><br>
<label>Address:</label><br>
<textarea rows="4" cols="30"></textarea><br><br>
<input type="submit" value="Register">
</form>
</body>
</html>
```

OUTPUT



User Registration Form

Full Name:

Email:

Password:

Gender:
Male Female Other

Date of Birth:
dd-mm-yyyy

Address:

Result
Thus, a simple user registration form using HTML form elements was successfully created and executed.

EX NO 2 Styling Web Pages using CSS Box Model and Background Properties

Aim

To understand and apply CSS Box Model and background properties to style a web page by controlling layout, spacing, and background appearance of HTML elements.

Procedure

CSS Box Model

The CSS Box Model is a fundamental concept used to design and layout web pages. Every HTML element is considered as a rectangular box, and the box model defines how the size and spacing of elements are calculated.

Components of the CSS Box Model

1. Content
   The actual content of the element such as text or images. The size is controlled using

width and height.

2. Padding
   The space between the content and the border. It increases the inner spacing of the element.
3. Border
   A line that surrounds the padding and content. It can be styled using color, width, and style.
4. Margin
   The outermost space around the element. It controls the distance between elements.

Box Model Structure:
Margin → Border → Padding → Content

CSS Background Properties

Background properties are used to define the background effects of elements.

Common CSS Background Properties

- background-color – Sets the background color
- background-image – Sets an image as background
- background-repeat – Controls repetition of the background image
- background-position – Positions the background image
- background-size – Defines the size of the background image
- background-attachment – Fixes or scrolls the background
- background – Shorthand property for all background settings

Examples of Background Properties

```
background-color: lightgray;
background-image: url("bg.jpg");
background-repeat: no-repeat;
background-position: center;

background-size: cover;
background-attachment: fixed;
```

Program

HTML CODE

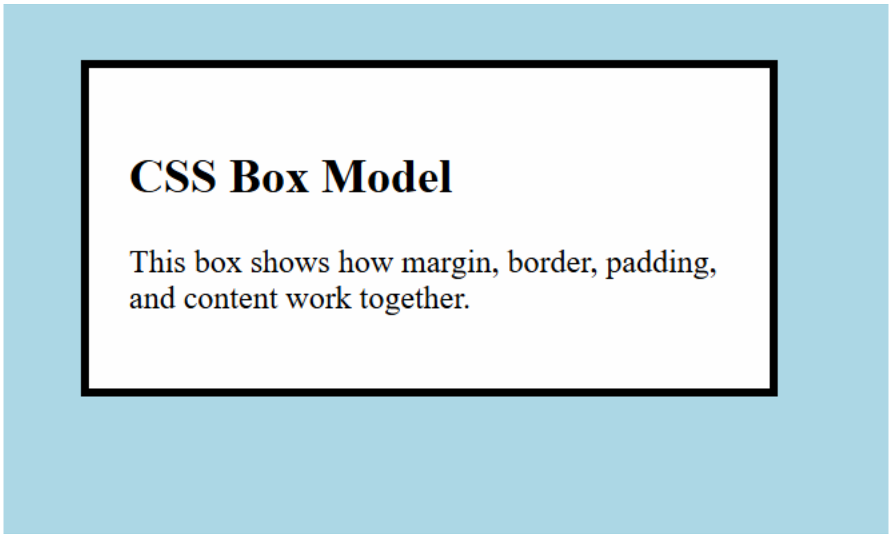
```
<!DOCTYPE html>
<html>
<head>
   <title>CSS Box Model and Background</title>
   <link rel="stylesheet" href="style.css">
```

```
</head>
<body>
   <div class="box">
      <h2>CSS Box Model</h2>
      <p>
         This box shows how margin, border, padding, and content work together.
      </p>
   </div>
</body>
</html>
```

CSS CODE

```
/* Background styling for the page */
body {
   background-color: lightblue;
   background-image: url("background.jpg");
   background-repeat: no-repeat;
   background-position: center;
   background-size: cover;
}
/* Box Model demonstration */
.box {
   width: 300px;              /* Content width */
   padding: 20px;             /* Space inside border */
   border: 4px solid black;   /* Border around content */
   margin: 40px;              /* Space outside border */
   background-color: white;   /* Background color */
   background-image: url("pattern.png");
   background-repeat: no-repeat;
   background-position: right bottom;
}
```

OUTPUT



CSS Box Model

This box shows how margin, border, padding, and content work together.

Result

Thus, the CSS Box Model and background properties were successfully used to style a web page and control layout and appearance.

Ex No 3 Implementation of CSS Animations and Multi-Column Layout

Aim

To implement CSS animations using animation properties and to create a multiple column layout using CSS column properties for dividing content effectively.

Procedure

1. CSS Animations

CSS animations allow HTML elements to gradually change their style over time without using JavaScript. Animations are created using **@keyframes** and controlled using animation properties.

Key Animation Properties:

- **@keyframes** – Defines the animation steps
- **animation-name** – Name of the animation
- **animation-duration** – Total time taken for one animation cycle
- **animation-delay** – Delay before animation starts
- **animation-iteration-count** – Number of times animation repeats
- **animation-direction** – Direction of animation (normal, reverse, alternate)
- **animation-timing-function** – Speed curve (ease, linear, ease-in-out)

2. Multiple Column Layout in CSS

CSS multiple columns allow content to be split into columns like a newspaper layout.

Key Column Properties:

- **column-count** – Number of columns
- **column-gap** – Space between columns
- **column-rule** – Line between columns
- **column-width** – Width of each column
- **column-span** – Element spans across columns

Procedure

1. Create an HTML file and define the page structure.
2. Use CSS to apply animation effects using @keyframes.
3. Apply animation properties to HTML elements.
4. Use CSS column properties to divide text into multiple columns.
5. Open the file in a web browser to view the animation and column layout output.

Program

HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>CSS Animation and Multiple Columns</title>
   <link rel="stylesheet" href="style.css">
</head>
```

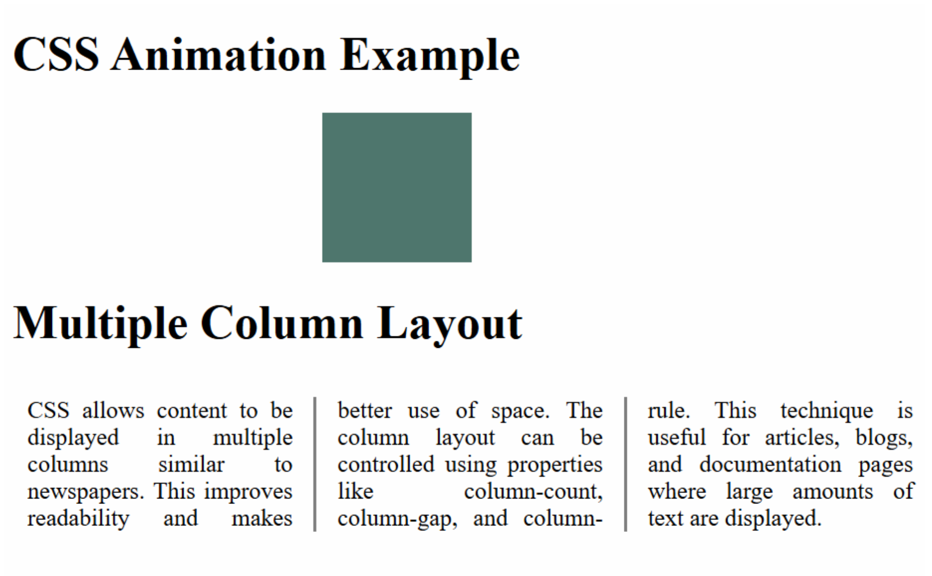
```
<body>
   <h1>CSS Animation Example</h1>
   <div class="box"></div>
   <h1>Multiple Column Layout</h1>
   <div class="columns">
      CSS allows content to be displayed in multiple columns similar to
      newspapers. This improves readability and makes better use of space.
      The column layout can be controlled using properties like column-count,
      column-gap, and column-rule. This technique is useful for articles,
      blogs, and documentation pages where large amounts of text are displayed.
   </div>
</body>
</html>
```

CSS Code (style.css)

```
/* Animation Example */
.box {
   width: 100px;
   height: 100px;
   background-color: tomato;
   position: relative;
   animation-name: moveBox;
   animation-duration: 3s;
   animation-iteration-count: infinite;
   animation-direction: alternate;
}
@keyframes moveBox {
   from {
      left: 0px;
      background-color: tomato;
   }
   to {
      left: 300px;
      background-color: teal;
   }
}
```

```
.columns {
   column-count: 3;
   column-gap: 30px;
   column-rule: 2px solid gray;
   font-size: 16px;
   text-align: justify;
   padding: 10px;
}
```

Output



CSS Animation Example

Multiple Column Layout

CSS allows content to be displayed in multiple columns similar to newspapers. This improves readability and makes better use of space. The column layout can be controlled using properties like column-count, column-gap, and column-rule. This technique is useful for articles, blogs, and documentation pages where large amounts of text are displayed.

Result

Thus, CSS animations were successfully implemented using animation properties, and a multiple column layout was created using CSS column properties.

Ex No 4 JavaScript Program Demonstrating Different Data Types

Aim

To write a JavaScript program that declares and initializes variables using different data types such as string, number, Boolean, array, and object, and to understand their usage.

Procedure

JavaScript is a **loosely typed** programming language, meaning variables can hold values of different data types without explicit declaration.

Common JavaScript Data Types:

1. **String**
   - Used to store text.
   - Enclosed within single (' ') or double (" ") quotes.
   - Example: "Hello World"
2. **Number**
   - Used to store numeric values (integers or decimals).
   - Example: 25, 3.14
3. **Boolean**
   - Represents logical values.
   - Can have only two values: true or false.
4. **Array**
   - Used to store multiple values in a single variable.
   - Values are stored in square brackets [].
5. **Object**
   - Used to store data in key–value pairs.
   - Defined using curly braces {}.

Steps to be followed

1. Create an HTML file.
2. Write JavaScript code inside the <script> tag.
3. Declare variables using var, let, or const.
4. Assign values of different data types to variables.
5. Display the output using document.write() or console.log().
6. Open the file in a web browser to view the output.

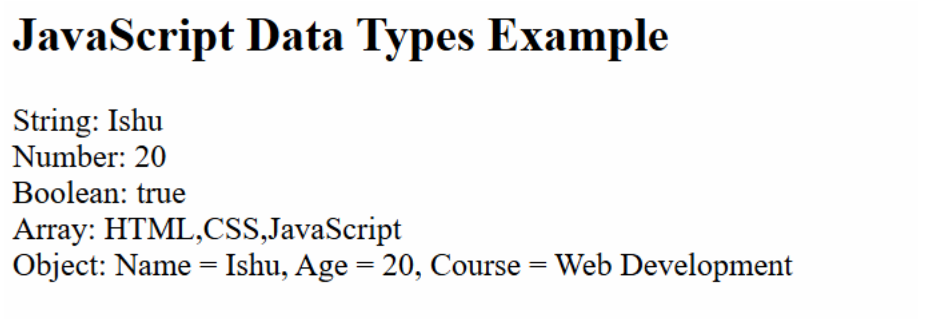
Program

HTML + JavaScript Code

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>JavaScript Data Types</title>
</head>
```

```
<body>
   <h2>JavaScript Data Types Example</h2>
   <script>
      // String
      let studentName = "Ishu";
      // Number
      let age = 20;
      // Boolean
      let isStudent = true;
      // Array
      let subjects = ["HTML", "CSS", "JavaScript"];
      // Object
      let student = {
         name: "JOHN",
         age: 20,
         course: "Web Development"
      };
      document.write("String: " + studentName + "<br>");
      document.write("Number: " + age + "<br>");
      document.write("Boolean: " + isStudent + "<br>");
      document.write("Array: " + subjects + "<br>");
      document.write("Object: Name = " + student.name +
                  ", Age = " + student.age +
                  ", Course = " + student.course);
   </script>
</body>
</html>
```

Output



```
JavaScript Data Types Example

String: Ishu
Number: 20
Boolean: true
Array: HTML,CSS,JavaScript
Object: Name = Ishu, Age = 20, Course = Web Development
```

Result

Thus, the JavaScript program successfully demonstrated the declaration and initialization of variables using different data types such as string, number, Boolean, array, and object.

Ex No 5 JavaScript Function to Find Sum of Even Numbers in an Array

Aim

To write a JavaScript function that accepts an array of numbers, identifies all even numbers, and returns the sum of those even numbers.

Description

A **function** in JavaScript is a reusable block of code that performs a specific task.
An **array** is used to store multiple values in a single variable.

In this program:

- The function receives an array as a parameter.
- It iterates through each element using a loop.
- Even numbers are identified using the modulus operator (%).
- The sum of even numbers is calculated and returned.

Procedure

1. Create an HTML file.
2. Write JavaScript code inside the <script> tag.
3. Define a function that accepts an array as an argument.
4. Use a loop to traverse the array.
5. Check whether each number is even.
6. Add even numbers to a sum variable.
7. Display the result in the browser.

Program

HTML + JavaScript Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Sum of Even Numbers</title>
</head>
<body>
  <h2>Sum of Even Numbers in an Array</h2>
  <script>
    // Function to calculate sum of even numbers
    function sumOfEvenNumbers(numbers) {
      let sum = 0;
      for (let i = 0; i < numbers.length; i++) {
        if (numbers[i] % 2 === 0) {
```

```
                    sum += numbers[i];
                }
            }
            return sum;
        }
        // Array of numbers
        let numArray = [10, 15, 20, 25, 30, 35];
        // Function call
        let result = sumOfEvenNumbers(numArray);
        document.write("Array Elements: " + numArray + "<br>");
        document.write("Sum of Even Numbers: " + result);
    </script>
</body>
</html>
```

Output

```
Array Elements: 10,15,20,25,30,35
Sum of Even Numbers: 60
```

Result

Thus, the JavaScript function successfully identified all even numbers in the given array and returned their sum.

Ex No 6 HTML Form Design with Validation for Name, Email, and Age

Aim

To design and create an HTML form that collects user details such as Name, Email, and Age using appropriate input types with basic validation.

Description

An **HTML form** is used to collect user input and send it to a server or process it on the client side.
HTML5 provides various **input types** and **validation attributes** that help ensure correct data entry without using JavaScript.

Input Fields Used:

- **Name** – Text input
- **Email** – Email input with format validation
- **Age** – Number input with range validation

Validation Attributes:

- **required** – Ensures the field is not left empty
- **type** – Validates input format
- **min / max** – Restricts numeric range
- **placeholder** – Provides hint text

Procedure

1. Create an HTML file.
2. Use the <form> tag to define the form structure.
3. Add input fields for Name, Email, and Age.
4. Use appropriate input types (text, email, number).
5. Apply validation attributes such as required, min, and max.
6. Save and open the file in a web browser.
7. Enter invalid and valid data to observe validation.

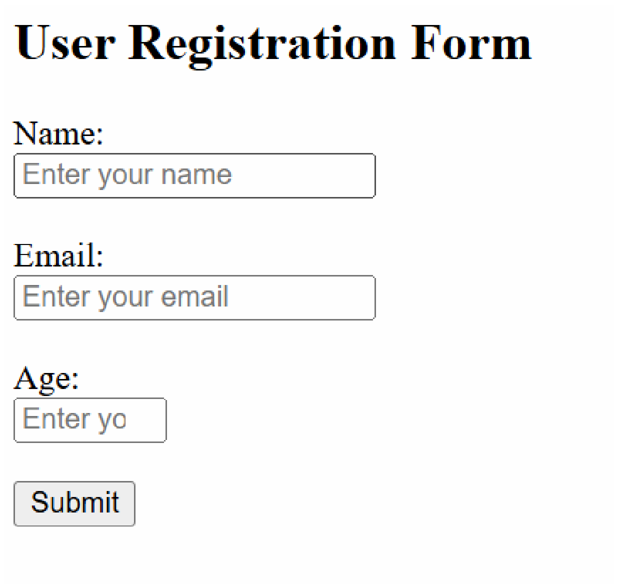
Program

HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>User Registration Form</title>
</head>
<body>

   <h2>User Registration Form</h2>
   <form>
```

```
        <label for="name">Name:</label><br>
        <input type="text" id="name" name="name"
            placeholder="Enter your name" required>
        <br><br>
        <label for="email">Email:</label><br>
        <input type="email" id="email" name="email"
            placeholder="Enter your email" required>
        <br><br>
        <label for="age">Age:</label><br>
        <input type="number" id="age" name="age"
            placeholder="Enter your age" min="1" max="120" required>
        <br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Output



User Registration Form

Name:
Enter your name

Email:
Enter your email

Age:
Enter yo

Submit

Result

Thus, an HTML form was successfully designed using appropriate input types and basic validation features.

Ex No 7 PHP Program Demonstrating Variables, Data Types, and Constants

AIM

To Write a PHP script that declares and initializes variables using different data types, including integer, string, float, and Boolean. Also, define a constant using the appropriate PHP function. Explain each data type with example values used in your script.

ALGORITHM

Integer

Description: Used to store whole numbers without decimal points.

Example in script:

$age = 25;

Explanation: The variable $age stores a whole number representing a person’s age.

String

Description: Used to store text or a sequence of characters.

Example in script:

$name = "John Doe";

Explanation: The variable $name stores textual data such as a person’s name.

Float (Double)

Description: Used to store numbers with decimal points.

Example in script:

$salary = 45000.75;

Explanation: The variable $salary stores a decimal value representing a salary amount.

Boolean

Description: Used to store only two possible values: true or false.

Example in script:

$isEmployed = true;

Explanation: The variable $isEmployed indicates whether the person is employed.

Constant

Description: A constant is a value that cannot be changed during script execution.

Defined using: define() function

Example in script:

define("COMPANY_NAME", "Tech Solutions Ltd");

Explanation: COMPANY_NAME stores the company name and remains unchanged throughout the program.

PROGRAM

```
  <?php
// Integer data type
$age = 25;
// String data type
$name = "John Doe";
// Float (double) data type
$salary = 45000.75;
// Boolean data type
$isEmployed = true;
// Defining a constant
define("COMPANY_NAME", "Tech Solutions Ltd");
// Displaying values
echo "Name: $name <br>";
echo "Age: $age <br>";
echo "Salary: $salary <br>";
echo "Employed: " . ($isEmployed ? "Yes" : "No") . "<br>";
echo "Company: " . COMPANY_NAME;
?>
```

OUTPUT

```
Name: John Doe
Age: 25
Salary: 45000.75
Employed: Yes
Company: Tech Solutions Ltd
```

RESULT

The program executed successfully and demonstrated the use of variables, data types, and constants in PHP.

Ex No 8 PHP Function to Find Maximum Value in an Array

AIM

To Write a PHP function that accepts an array of numbers as input and returns the maximum value from the array. Explain how the function works and test it with a sample array.

ALGORITHM

Function Definition:

The function findMaxValue accepts an array called $numbers.

Empty Array Check:

The first thing the function does is check if the array is empty with empty($numbers). If it is empty, the function returns null because there's no maximum value in an empty array.

Initialization of Maximum:

The function assumes the first element ($numbers[0]) is the maximum value and assigns it to the variable $max.

Iterating through the Array:

The function uses a foreach loop to go through each element of the array. For each element $number, it checks if it's greater than the current $max. If it is, the $max variable is updated.

Return the Maximum Value:

After the loop completes, the function returns the largest value found in the array.

How the Function works:

Initially, the max value is set to the first element of the array, 12.

The loop then compares each element:

45 is greater than 12, so max is updated to 45.

7 is less than 45, so max remains 45.

99 is greater than 45, so max is updated to 99.

34 is less than 99, so max remains 99.

Thus, the function returns 99, which is the largest value in the array.

PROGRAM

```
<?php
function findMaxValue($numbers) {
   // Check if the array is empty
   if (empty($numbers)) {
      return null; // Return null if the array is empty
   }
   // Initialize the max variable to the first value in the array
   $max = $numbers[0];

   // Loop through the array starting from the second element
```

```
    foreach ($numbers as $number) {
        // If the current number is greater than the current max, update the max
        if ($number > $max) {
            $max = $number;
        }
    }
    // Return the maximum value found
    return $max;
}
// Test the function with a sample array
$sampleArray = [12, 45, 7, 99, 34];
echo "The maximum value is: " . findMaxValue($sampleArray);
?>
```

Output:

```
$sampleArray = [12, 45, 7, 99, 34];
The maximum value is: 99
```

RESULT

The program executed successfully and correctly displayed the maximum value from the given array.

Ex No 9 PHP Script for Image File Upload Handling

AIM

To Write a PHP script to handle the file upload process, ensuring that only image files (JPEG, PNG) are allowed. Display a message confirming successful upload and save the file in a specific directory.

ALGORITHM

HTML Form:

The form allows users to select an image file from their local machine and upload it.

The form uses the multipart/form-data encoding type, which is necessary for uploading files.

PHP Script:

The $_FILES["fileToUpload"] array holds information about the uploaded file, including the temporary file name, the original file name, and its type.

The script checks if the file is an actual image using the getimagesize() function. If it’s not an image, the upload is denied.

It restricts the file types to only JPEG and PNG images by checking the file extension.

If the file passes validation, it is moved to the specified directory (uploads/), and a success message is displayed.

Upload Directory:

The uploaded files are saved in a folder named uploads/. You should create this folder in your project directory and ensure it has appropriate write permissions.

Error Handling:

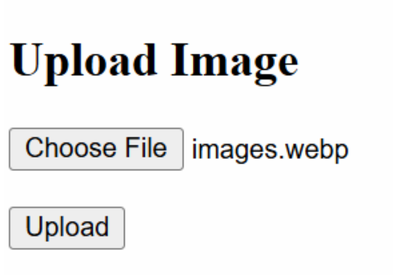
If the file type is not allowed or if there’s an error with the upload, an error message is shown to the user.

PROGRAM

```
<!DOCTYPE html>
<html>
<head>
   <title>Image Upload</title>
</head>
<body>
   <h2>Upload Image</h2>
   <form action="upload.php" method="post" enctype="multipart/form-data">
      <input type="file" name="image" required>
      <br><br>
      <input type="submit" name="upload" value="Upload">
   </form>
</body>
```

```
</html>
<?php
if (isset($_POST['upload'])) {

    $uploadDir = "uploads/";
    $fileName = $_FILES["image"]["name"];
    $fileTmp = $_FILES["image"]["tmp_name"];
    $fileType = $_FILES["image"]["type"];
    $fileError = $_FILES["image"]["error"];

    // Allowed image types
    $allowedTypes = array("image/jpeg", "image/png");

    if ($fileError === 0) {
        if (in_array($fileType, $allowedTypes)) {

            // Create uploads directory if not exists
            if (!is_dir($uploadDir)) {
                mkdir($uploadDir, 0777, true);
            }

            $targetFile = $uploadDir . basename($fileName);

            if (move_uploaded_file($fileTmp, $targetFile)) {
                echo "Image uploaded successfully!";
            } else {
                echo "Error uploading file.";
            }

        } else {
            echo "Only JPEG and PNG images are allowed.";
        }
    } else {
        echo "File upload error.";
    }
}
?>
```

OUTPUT



RESULT

The program executed successfully and uploaded only valid image files with proper validation.

Ex No: 10 Node.js “Hello World” Program

Aim

To write and execute a simple Node.js program that displays the message **“Hello, World!”** using the command line.

Description

Node.js is a JavaScript runtime environment that allows JavaScript code to be executed outside the web browser. It is commonly used for server-side programming and command-line applications.

In this experiment:

- A JavaScript file (.js) is created.
- The program prints a message to the console.
- The file is executed using the Node.js command-line interface.

Procedure

1. Install Node.js on the system.
2. Open a text editor (Notepad / VS Code).
3. Write a JavaScript program to print “Hello, World!”.
4. Save the file with a .js extension.
5. Open Command Prompt / Terminal.
6. Navigate to the folder where the file is saved.
7. Run the program using the node command.

Program

JavaScript Code (hello.js)

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

Steps to Save and Run the Program

1. Save the file as **hello.js**.
2. Open the Command Prompt.
3. Use the cd command to go to the directory containing hello.js.
4. cd path_to_folder
5. Execute the program using:
6. node hello.js

Output

```
Hello, World!
```

Result

Thus, a simple Node.js program was successfully written, saved, and executed using the command line.

Ex No 11 Node.js Program using EventEmitter and setTimeout

Aim

To create a Node.js program that uses the EventEmitter class, defines a custom event named **"welcome"**, and triggers it after a delay of 3 seconds to display a message.

Description

Node.js provides the **events** module to handle asynchronous events.
The **EventEmitter** class allows objects to emit and listen for named events.

In this program:

- An EventEmitter object is created.
- A custom event called "welcome" is defined.
- The setTimeout() function is used to trigger the event after 3 seconds.
- When the event occurs, a message is displayed in the console.
- require('events') imports the built-in events module.
- new EventEmitter() creates an event emitter instance.
- emitter.on('welcome', callback) registers an event listener.
- setTimeout() delays execution by 3000 milliseconds (3 seconds).
- emitter.emit('welcome') triggers the custom event.
- When the event is emitted, the callback function executes and prints the message

Procedure

1. Open a text editor (VS Code / Notepad).
2. Create a new JavaScript file.
3. Import the events module.
4. Create an instance of EventEmitter.
5. Define the "welcome" event using .on().
6. Use setTimeout() to emit the event after 3 seconds.
7. Save and run the file using Node.js.

Program

JavaScript Code (eventEmitter.js)

```
// Import events module
const EventEmitter = require('events');
// Create an EventEmitter object
const emitter = new EventEmitter();
// Define the custom event "welcome"
```

```
emitter.on('welcome', () => {
   console.log("Welcome to Node.js!");
});
// Trigger the event after 3 seconds
setTimeout(() => {
   emitter.emit('welcome');
}, 3000);
```

Output

After 3 seconds, the console displays:

```
Welcome to Node.js!
```

Result

Thus, the Node.js program successfully created and triggered a custom event using EventEmitter after a time delay.

Ex No 12 Node.js Callback Function for Sum of Two Numbers

Aim

To write a Node.js function that accepts two numbers and a callback function, and to use the callback to return the sum of the two numbers.

Theory / Description

In Node.js, a **callback function** is a function passed as an argument to another function and executed after the completion of a task. Callbacks are commonly used to handle asynchronous and synchronous operations.

In this program:

- A function accepts two numeric values and a callback.
- The callback function performs the addition.
- The result is displayed using the callback mechanism.
- calculateSum(a, b, callback) is the main function.
- It receives two numbers (a and b) and a callback function.
- The callback function addNumbers() is passed as an argument.
- Inside calculateSum(), the callback is executed using callback(a, b).
- The callback function calculates the sum and prints the result.
- This demonstrates how functions can be passed and executed dynamically in Node.js.

Procedure

1. Open a text editor (VS Code / Notepad).
2. Create a new JavaScript file.
3. Define a function that accepts two numbers and a callback.
4. Define the callback function to return the sum.
5. Call the main function and display the result.
6. Run the program using Node.js from the command line.

Program

JavaScript Code (callbackSum.js)

```
// Function that accepts two numbers and a callback
function calculateSum(a, b, callback) {
   callback(a, b);
}
// Callback function to return the sum
function addNumbers(x, y) {
    console.log("Sum of the two numbers:", x + y);
}
// Function call
calculateSum(10, 20, addNumbers);
```

Output

```
Sum of the two numbers: 30
```

Result

Thus, the Node.js program successfully demonstrated the use of a callback function to compute and return the sum of two numbers.

Ex No 13 Node.js Program to Read a File using fs Module

Aim

To write a Node.js script that reads the contents of a text file using the **fs** module and displays the content in the console.

Description

The **fs (File System)** module in Node.js is a built-in module that allows interaction with the file system. It provides methods to read, write, update, delete, and manage files.

Key Points about fs Module:

- Supports **asynchronous** and **synchronous** file operations.
- Asynchronous methods prevent blocking of program execution.
- Uses **callbacks** to handle results after the operation completes.

In this experiment, the asynchronous method fs.readFile() is used to read a text file.

Steps Involved in Asynchronous File Reading

1. The readFile() function is called.
2. Node.js continues executing other code without waiting.
3. Once the file is read, the callback function is executed.
4. The file content or error is returned via the callback.

Procedure

1. Create a text file named sample.txt and add some content.
2. Open a text editor and create a new JavaScript file.
3. Import the fs module.
4. Use fs.readFile() to read the file asynchronously.
5. Display the file contents in the console.
6. Run the program using Node.js.

Program

Text File (sample.txt)

```
Welcome to Node.js File System Module.
This is a sample text file.
```

JavaScript Code (readFile.js)

```
// Import fs module
const fs = require('fs');
// Read the file asynchronously
fs.readFile('sample.txt', 'utf8', (error, data) => {
    if (error) {
        console.log("Error reading file:", error);
        return;
    }
    console.log("File Content:\n" + data);
});
```

Output

```
File Content:
Welcome to Node.js File System Module.
This is a sample text file.
```

Result

Thus, the Node.js program successfully read and displayed the contents of a text file using the fs module asynchronously.

Ex No 14 MongoDB Installation and Database Creation

Aim

To install MongoDB on the system, start the MongoDB server, and create a new database named **studentDB** using the MongoDB shell.

Description

MongoDB is a popular **NoSQL database** that stores data in flexible, JSON-like documents. It is document-oriented, scalable, and widely used for modern web applications.

Key Components:

- **MongoDB Server (mongod)** – Runs the database service
- **MongoDB Shell (mongosh)** – Command-line interface to interact with MongoDB
- **Database** – A container for collections

Procedure

1. Download and install MongoDB Community Edition.
2. Start the MongoDB server.
3. Open the MongoDB shell.
4. Create a new database named studentDB.
5. Verify the database creation.

Steps for Installation and Execution

Step 1: Install MongoDB

1. Download **MongoDB Community Server** from the official MongoDB website.
2. Run the installer.
3. Choose **Complete Installation**.
4. Select **Install MongoDB as a Service**.
5. Finish the installation.

Step 2: Start MongoDB Server

Option 1: Using Command Prompt

```
mongod
```

Option 2: If Installed as a Service

```
net start MongoDB
```

If MongoDB starts successfully, it listens on port **27017**.

Step 3: Open MongoDB Shell

Open a new Command Prompt and type:

```
mongosh
```

If connected successfully, the MongoDB shell prompt appears.

Step 4: Create a Database (studentDB)

In MongoDB shell:

```
use studentDB
```

MongoDB creates the database when data is first inserted.

Insert a sample document:

```
db.students.insertOne({ name: "Ishu", age: 20, course: "Web Development" })
```

Step 5: Verify Database Creation

List all databases:

```
show dbs
```

You will see **studentDB** in the list.

Output

```
switched to db studentDB
{
  acknowledged: true,
  insertedId: ObjectId("...")
}
```

Result

Thus, MongoDB was successfully installed, the MongoDB server was started, and a new database named **studentDB** was created using the MongoDB shell.

Ex No 15 MongoDB Collection Creation and Document Insertion

Aim

To create a database named **studentDB**, create a collection called **students**, and insert a document containing student details using the **MongoDB shell**.

Procedure

MongoDB is a NoSQL, document-oriented database that stores data in **collections** and **documents** instead of tables and rows.
In this experiment, the MongoDB shell is used to:

1. Create and switch to a database.
2. Create a collection implicitly.
3. Insert a document into the collection.
4. Verify the inserted document.

Step 1: Start MongoDB Shell

Open Command Prompt / Terminal and start the MongoDB shell.

```
Mongosh
```

This command opens the MongoDB interactive shell where database commands can be executed.

Step 2: Create and Switch to Database

```
use studentDB
```

- If studentDB already exists, MongoDB switches to it.
- If it does not exist, MongoDB creates it when data is inserted.

Step 3: Create Collection and Insert Document

```
db.students.insertOne({
  name: "John Doe",
  age: 20,
  course: "Computer Science"
})
```

- students is the collection name.
- insertOne() inserts a single document.
- MongoDB automatically creates the collection if it does not already exist.

Step 4: Display Inserted Document

```
db.students.find()
```

This command retrieves and displays all documents stored in the students collection.

Output

```
{
  "_id" : ObjectId("64f8a3c9b7c1a8d123456789"),
  "name" : "John Doe",
  "age" : 20,
  "course" : "Computer Science"
}
```

Result

Thus, the database **studentDB** was successfully created, the **students** collection was created, and a document with student details was inserted and displayed using the MongoDB shell.

Ex No 16 Node.js Program to Connect MongoDB and Retrieve Data

Aim

To write a simple **Node.js script** to connect to a **MongoDB database** and retrieve all documents from the **students** collection using the **MongoDB Node.js driver**.

Procedure

Node.js is a JavaScript runtime that allows execution of JavaScript code outside the browser. MongoDB provides an official **Node.js driver** to interact with MongoDB databases programmatically.

In this experiment:

1. The MongoDB Node.js driver is installed.
2. A connection is established between Node.js and MongoDB.
3. The required database and collection are accessed.
4. All documents from the students collection are retrieved and displayed.

Step 1: Install MongoDB Node.js Driver

Run the following command in the project directory:

```
npm install mongodb
```

This command installs the official MongoDB driver required to connect Node.js applications with MongoDB databases.

Step 2: Import MongoDB Client

```
const { MongoClient } = require("mongodb");
```

MongoClient is used to establish a connection between the Node.js application and the MongoDB server.

Step 3: Define MongoDB Connection URL and Database Name

```
const url = "mongodb://localhost:27017";
const dbName = "studentDB";
```

- url specifies the MongoDB server address.
- dbName specifies the database to be accessed.

Step 4: Establish Connection and Retrieve Documents

```
async function fetchStudents() {
  const client = new MongoClient(url);
  try {
    await client.connect();
    console.log("Connected to MongoDB");
    const db = client.db(dbName);
    const collection = db.collection("students");
    const students = await collection.find({}).toArray();

    console.log("Students Collection Data:");
    console.log(students);
  } catch (error) {
    console.error("Error:", error);
  } finally {
    await client.close();
  }
}

fetchStudents();
```

Complete Node.js Program

```
const { MongoClient } = require("mongodb");
const url = "mongodb://localhost:27017";
const dbName = "studentDB";
async function fetchStudents() {
  const client = new MongoClient(url);
  try {
    await client.connect();
    console.log("Connected to MongoDB");
    const db = client.db(dbName);
    const collection = db.collection("students");
    const students = await collection.find({}).toArray();
    console.log(students);
  } catch (error) {
    console.error(error);
  } finally {
    await client.close();
  }
}
fetchStudents();
```

Output

```
Connected to MongoDB
[
  {
    "_id": "64f8a3c9b7c1a8d123456789",
    "name": "John Doe",
    "age": 20,
    "course": "Computer Science"
  }
]
```

Result

Thus, a Node.js script was successfully created to connect to the MongoDB database **studentDB**, retrieve all documents from the **students** collection using the MongoDB Node.js driver, and display them in the console.

Ex No 17 Express.js Server Creation and Routing

Aim

To install **Express.js** and create a basic **Express server** using Node.js that listens on **port 3000** and responds with **“Hello, Express!”** when accessed at the root URL (/).

Procedure

Express.js is a lightweight and flexible **Node.js web application framework** used to build web servers and APIs easily. It provides features such as routing, middleware support, and simplified HTTP request handling.

In this experiment:

1. Express.js is installed using npm.
2. A simple Express application is created.
3. A server is configured to listen on port 3000.
4. A route is defined to handle requests at the root URL.

Step 1: Initialize Node.js Project

```
npm init -y
```
This command creates a package.json file which stores project details and dependencies.

Step 2: Install Express.js

```
npm install express
```
This command installs the Express.js framework and adds it as a dependency in the project.

Step 3: Import Express Module

```
const express = require("express");
```
This statement imports the Express module so that it can be used to create a server.

Step 4: Create Express Application

```
const app = express();
```
The express() function creates an Express application object.

Step 5: Define Root Route

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

- app.get() handles HTTP GET requests.
- / represents the root URL.
- res.send() sends the response to the browser.

Step 6: Start the Server

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

This starts the server and makes it listen for requests on port 3000.

Complete Node.js Express Program

```
const express = require("express");
const app = express();
app.get("/", (req, res) => {
  res.send("Hello, Express!");
});
app.listen(3000, () => {
  console.log("Server running on port 3000");
});
```

Output

When the server is running and the browser accesses:

http://localhost:3000/

Output displayed in browser:

```
Hello, Express!
```

Console Output:

```
Server running on port 3000
```

Result

Thus, Express.js was successfully installed, and a basic Express server was created using Node.js. The server listens on **port 3000** and responds with **“Hello, Express!”** when accessed at the root URL.

Ex No 18 **React Application with Counter Component**

Aim

To create a simple **React application** with a component named **Counter** that displays a count value and increments the count when a button is clicked.

Procedure

React is a JavaScript library used to build **user interfaces** using reusable components. React components can manage data using **state**, and the UI automatically updates when the state changes.

In this experiment:

1. A React application is created.
2. A functional component named Counter is implemented.
3. React useState hook is used to manage the count value.
4. A button click event updates and displays the count.

Step 1: Create a React Application

Open terminal / command prompt and run:

```
npx create-react-app counter-app
```

This command creates a new React project named counter-app with all required dependencies and configuration.

Step 2: Navigate to Project Directory

```
cd counter-app
```

Moves into the newly created React project folder.

Step 3: Start the React Application

```
npm start
```

Starts the development server and opens the React application in the browser.

Step 4: Create Counter Component

Open src/App.js and replace the existing code with the following:

```
import React, { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h2>Counter Value: {count}</h2>
      <button onClick={() => setCount(count + 1)}>
```

```
        Increment
      </button>
    </div>
  );
}

export default Counter;
```

- useState(0) initializes the count state with value 0.
- count stores the current value.
- setCount() updates the state.
- Clicking the button increases the count and re-renders the component.

Step 5: Display Counter Component

Ensure index.js renders the App component (default):

```
import React from "react";
import ReactDOM from "react-dom/client";
import Counter from "./App";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Counter />);
```

This renders the Counter component inside the root DOM element.

Complete Counter Component Code

```
import React, { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h2>Counter Value: {count}</h2>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
export default Counter;
```

Output

- Initial display:

Counter Value: 0

[Increment]

- After clicking the button:

Counter Value: 1

Counter Value: 2

Counter Value: 3

(The value increases on every button click.)

Result

Thus, a simple React application was successfully created with a **Counter component**. The component uses React state to update and display the count value dynamically whenever the button is clicked.

Ex No 19 React Router Implementation with Home and About Pages

Aim

To set up **React Router** in a React application and create two pages named **Home** and **About**, enabling navigation between them without refreshing the page.

Procedure

React Router is a standard library used for **client-side routing** in React applications. It allows navigation between different components (pages) without reloading the entire web page, providing a smooth **Single Page Application (SPA)** experience.

In this experiment:

1. React Router is installed.
2. Routes are defined for Home and About pages.
3. Navigation links are created.
4. Page switching is handled dynamically without page refresh.

Step 1: Create a React Application

```
npx create-react-app router-app
```

Creates a new React project with all required dependencies and folder structure.

Step 2: Navigate to Project Folder

```
cd router-app
```

Moves into the project directory.

Step 3: Install React Router

```
npm install react-router-dom
```

Installs react-router-dom, which provides routing components like BrowserRouter, Routes, Route, and Link.

Step 4: Create Home and About Components

Home.js

```
function Home() {
  return <h2>Welcome to Home Page</h2>;
}
export default Home;
```

About.js

```
function About() {
  return <h2>About Us Page</h2>;
}
export default About;
```

Two simple functional components are created to represent different pages.

Step 5: Set Up Routing in App.js

```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Home from "./Home";
import About from "./About";
function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

- BrowserRouter wraps the application and enables routing.
- Routes contains all route definitions.
- Route maps URLs to components.
- Link enables navigation without page reload.

Step 6: Render App Component

Ensure index.js contains:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root = ReactDOM.createRoot(document.getElementById("root"));
```

```
root.render(<App />);
```

Complete Routing Code (App.js)

```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Home from "./Home";
import About from "./About";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

Output

- When accessing:

http://localhost:3000/

Output:

```
Welcome to Home Page
```

- When clicking **About** link:

http://localhost:3000/about

Output:

```
About Us Page
```

(Navigation occurs without page refresh.)

Result

Thus, React Router was successfully installed and configured. Two pages, **Home** and **About**, were created, and navigation between them was achieved using React Router links without refreshing the page.

Ex No 20 **React ToggleText Component using State and Events**

Aim

To create a React component named **ToggleText** that initially displays **“Hello!”** and toggles between **“Hello!”** and **“Goodbye!”** when a button is clicked using React state and event handling.

Procedure

React allows components to manage dynamic data using **state**. The **useState** hook is used in functional components to store and update values. Event handling in React enables user interactions such as button clicks to update the component state, which in turn updates the UI automatically.

In this experiment:

1. A functional component named ToggleText is created.
2. React state is used to store the displayed text.
3. A button click event toggles the text between two values.

Step 1: Create a React Application

```
npx create-react-app toggle-text-app
```

Creates a new React project with required dependencies and configuration.

Step 2: Navigate to Project Directory

```
cd toggle-text-app
```

Moves into the project folder.

Step 3: Implement ToggleText Component

Open src/App.js and write the following code:

```
import React, { useState } from "react";
function ToggleText() {
  const [text, setText] = useState("Hello!");

  const toggleMessage = () => {
    setText(text === "Hello!" ? "Goodbye!" : "Hello!");
  };
  return (
    <div>
```

```
      <h2>{text}</h2>
      <button onClick={toggleMessage}>
        Toggle Text
      </button>
    </div>
  );
}

export default ToggleText;
```

Step 4: Render the Component

Ensure index.js renders the component:

```
import React from "react";
import ReactDOM from "react-dom/client";
import ToggleText from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<ToggleText />);
```

Complete ToggleText Component Code

```
import React, { useState } from "react";
function ToggleText() {
  const [text, setText] = useState("Hello!");
  const toggleMessage = () => {
    setText(text === "Hello!" ? "Goodbye!" : "Hello!");
  };
  return (
    <div>
      <h2>{text}</h2>
      <button onClick={toggleMessage}>
        Toggle Text
      </button>
    </div>
  );
}
export default ToggleText;
```

Output

- Initial display:

Hello!

[Toggle Text]

- After clicking the button:

Goodbye!

- On next click:

Hello!

(Text toggles on every button click.)

Result

Thus, a React component named **ToggleText** was successfully created. The component uses React state and event handling to toggle the displayed text between **“Hello!”** and **“Goodbye!”** when the button is clicked.