Understanding JavaScript

1. Introduction

- JavaScript is a programming language used to make web pages interactive.
- It executes on the client-side, meaning it runs directly in the user's browser, enabling dynamic and responsive user experiences.
- Additionally, JavaScript can be used on the server-side with environments like Node.js, allowing developers to build complete applications using a single language.
- JavaScript is an interpreted language, meaning its code is executed line by line by the browser's JavaScript engine without the need for prior compilation.
- This allows developers to write and test code quickly, as changes can be seen immediately in the browser.

2. DOM

- The DOM (Document Object Model) in JavaScript is a programming interface for HTML and XML documents.
- It represents the structure of a web page as a tree of objects,
- Where each node corresponds to a part of the document (e.g., an element, attribute, or text).
- Document: The root of the DOM tree. Represents the entire web page.
- Elements: Represent HTML tags (e.g., <div>, <p>).

- Attributes: Represent HTML attributes (e.g., id, class).
- Nodes: Everything in the DOM (elements, text, comments) is a node.

Accessing and Modifying Elements

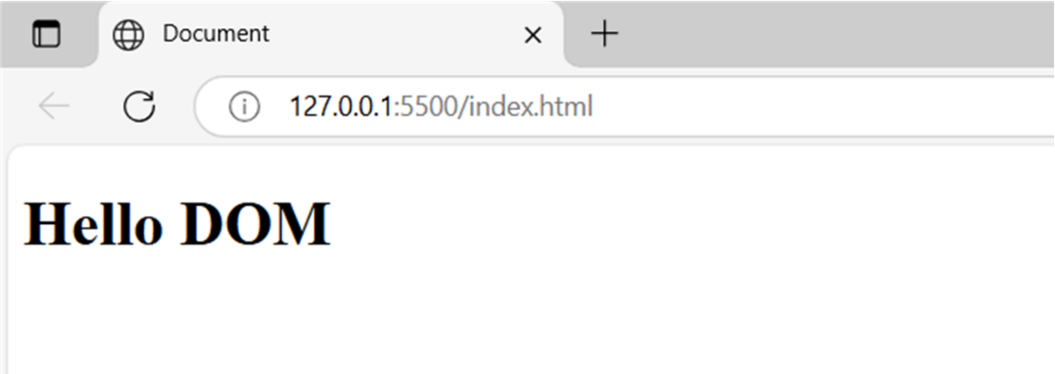
1) getElementById

index.html

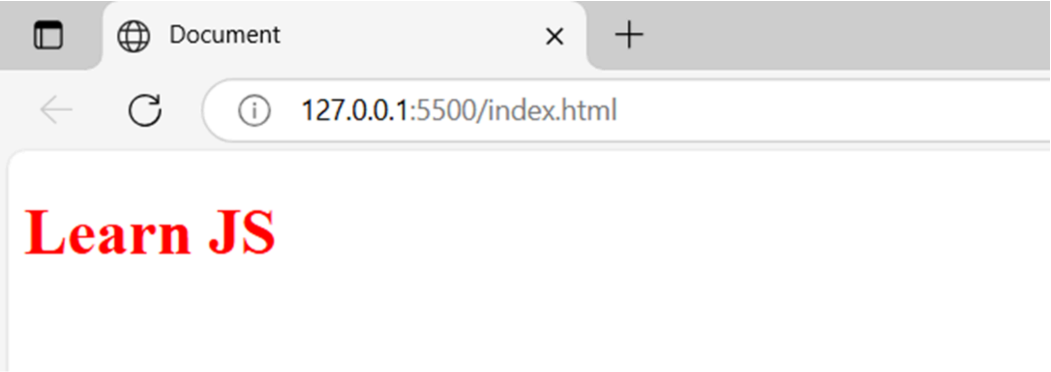
```
<h1 id="head1">Hello DOM</h1>
```

example.js

```
const value = document.getElementById(‘head1’)
value.innerHTML = 'Learn JS’
value.style.color=‘red’
console.log(value);
```



Hello DOM



Learn JS

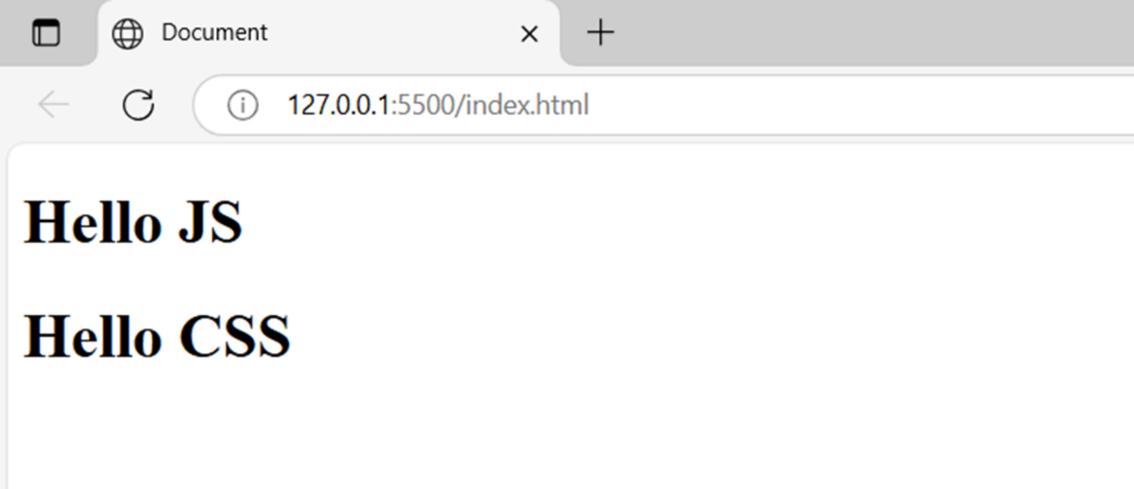
2) getElementsByClass

example.html

```
<h1 class="head">Hello JS</h1>
<h1 class="head">Hello CSS</h1>
```

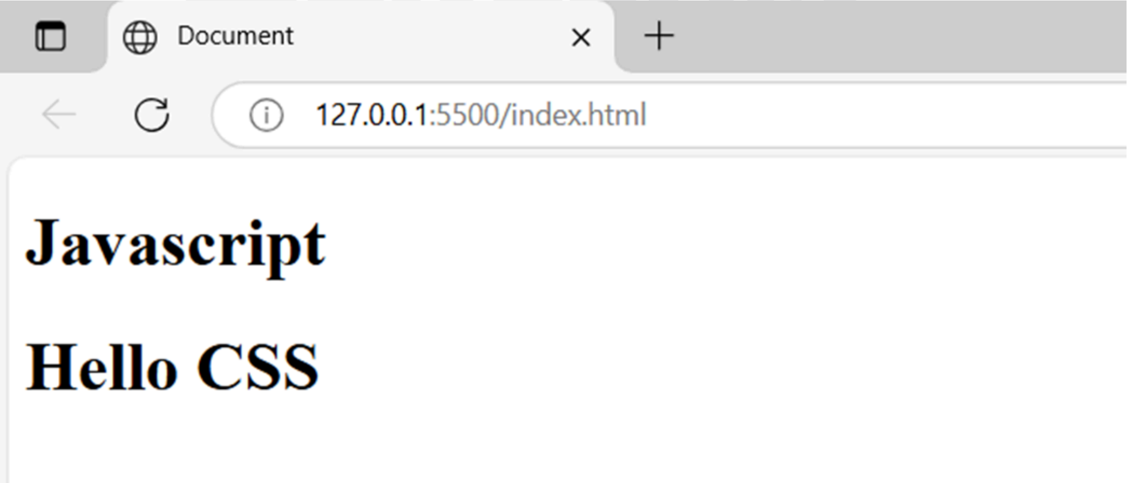
example.js

```
const res = document.getElementsByClassName('head’)
res[0].innerHTML = 'Javascript’
console.log(res[0]);
```



Hello JS

Hello CSS



Javascript

Hello CSS

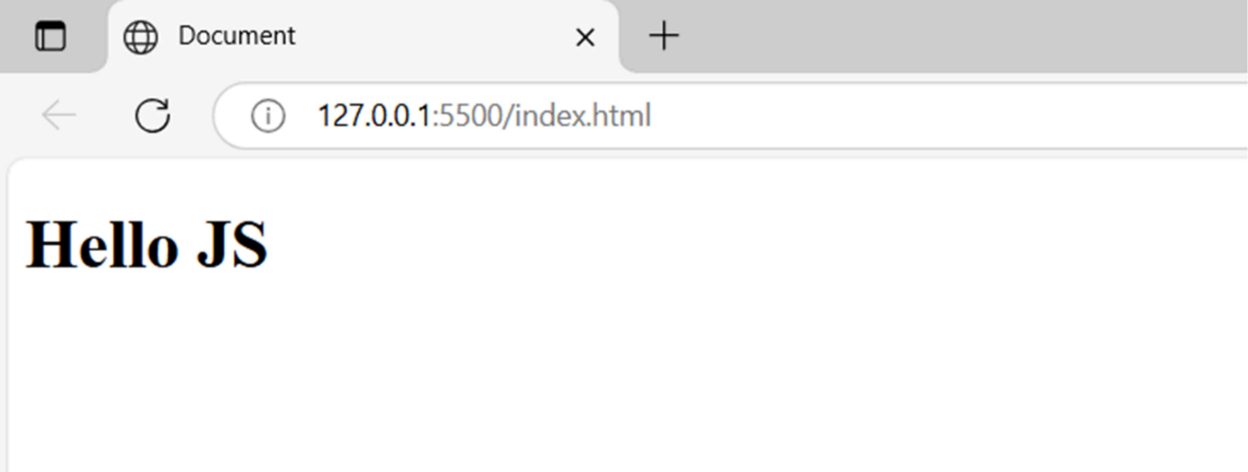
3) getElementsByTagName

example.html

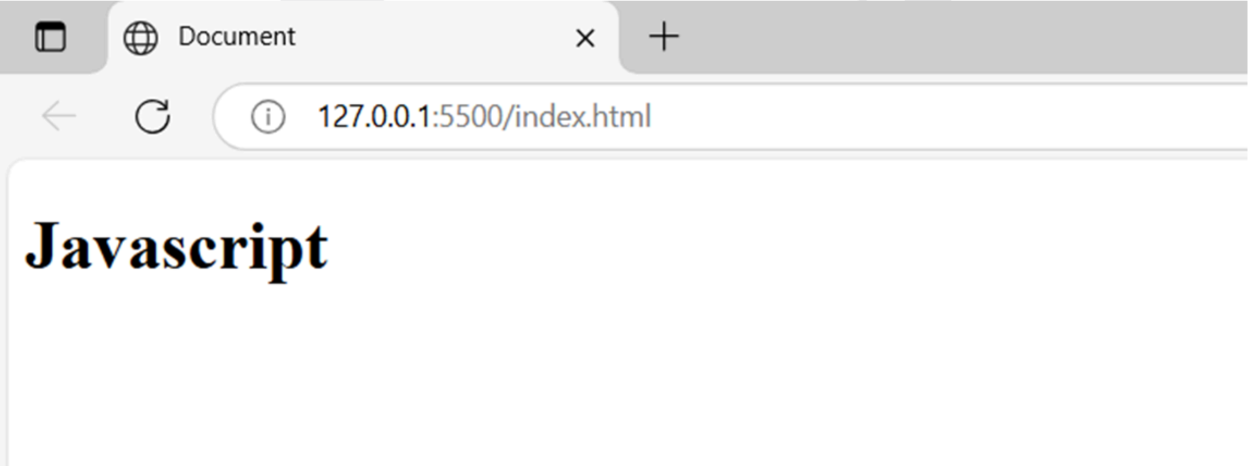
```
<h1 name=“doc">Hello JS</h1>
```

example.js

```
const res = document.getElementsByName(‘doc’)
res[0].innerHTML = 'Javascript’
console.log(res[0]);
```



Hello JS



Javascript

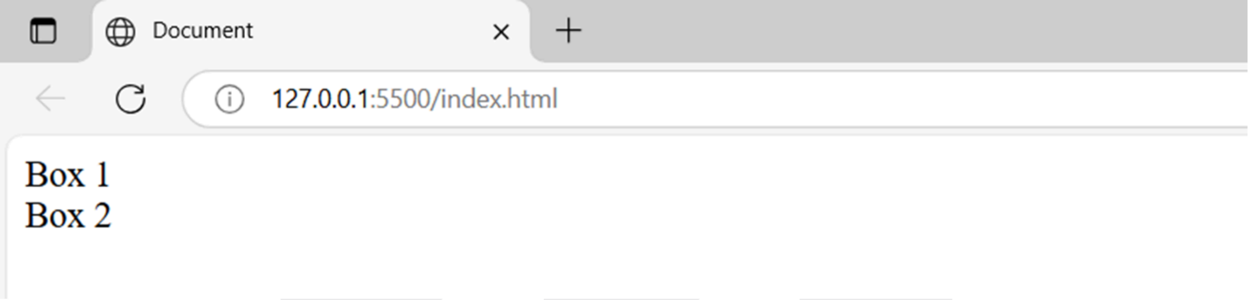
4) querySelector

example.html

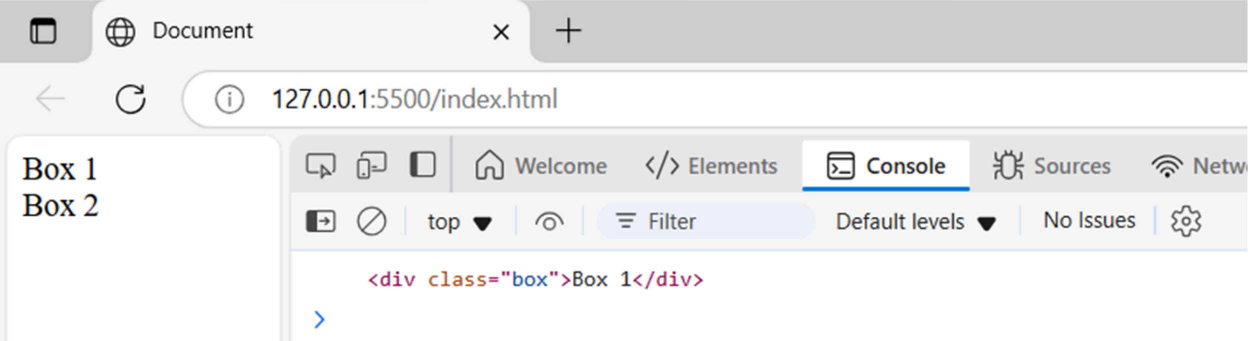
```
<div class="box">Box 1</div>
<div class="box">Box 2</div>
```

example.js

```
let firstBox = document.querySelector('.box’);
console.log(firstBox.textContent); // Output: "Box 1"
```

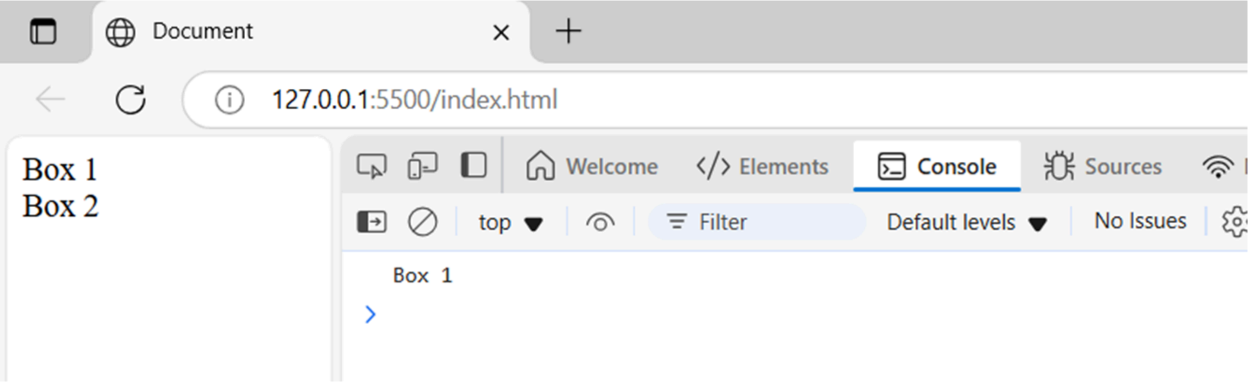


Box 1
Box 2



Box 1
Box 2

```
<div class="box">Box 1</div>
```



Box 1
Box 2

```
Box 1
```

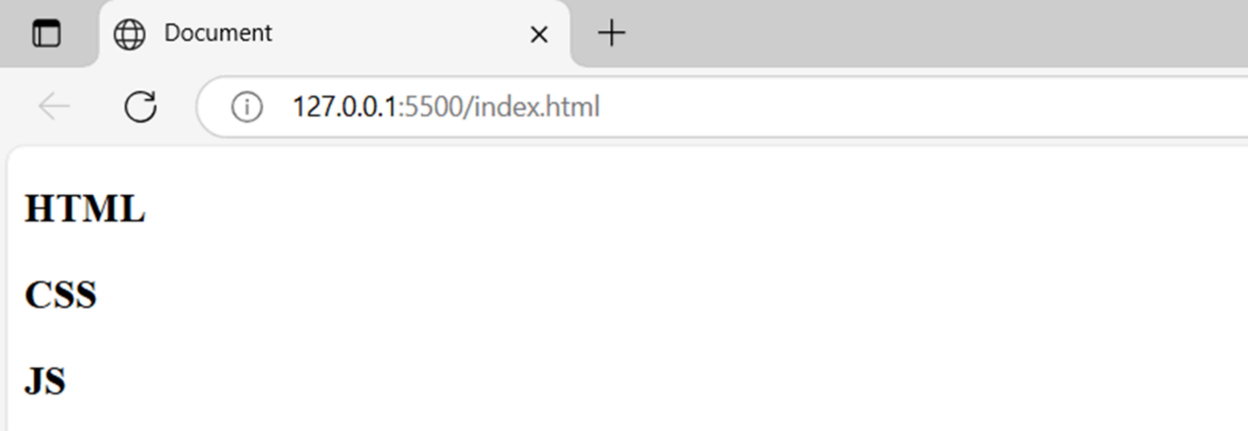
5) querySelectorAll

example.html

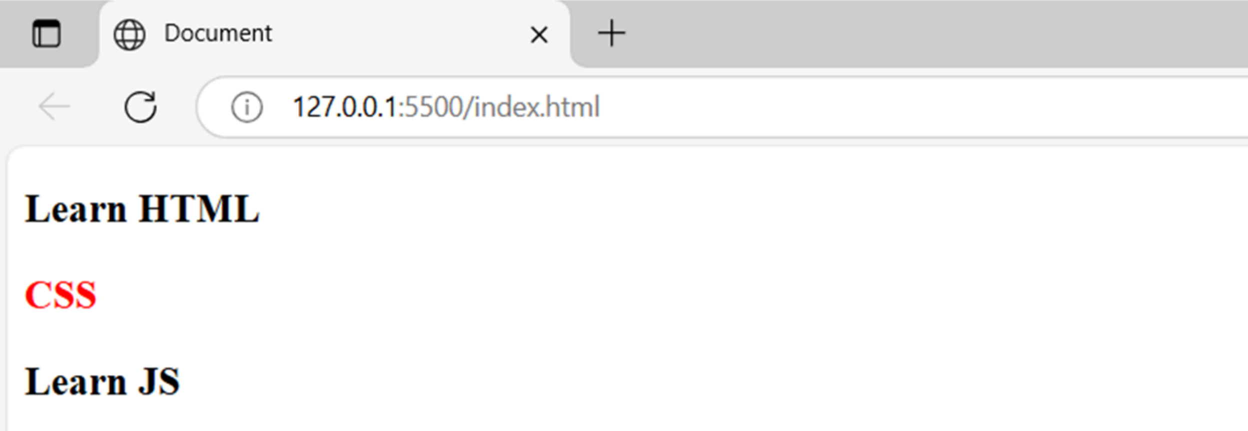
```
<h3 id="htm" class="htmlclass">HTML</h3>
<h3 class="htmlclass">CSS</h3>
<h3 class="htmlclass">JS</h3>
```

example.js

```
const res= document.querySelectorAll('h3’)
res[0].innerHTML = 'Learn HTML’
res[1].style.color = 'red’
res[2].innerHTML = 'Learn JS’
console.log(res);
```



```
HTML

CSS

JS
```



```
Learn HTML

CSS

Learn JS
```

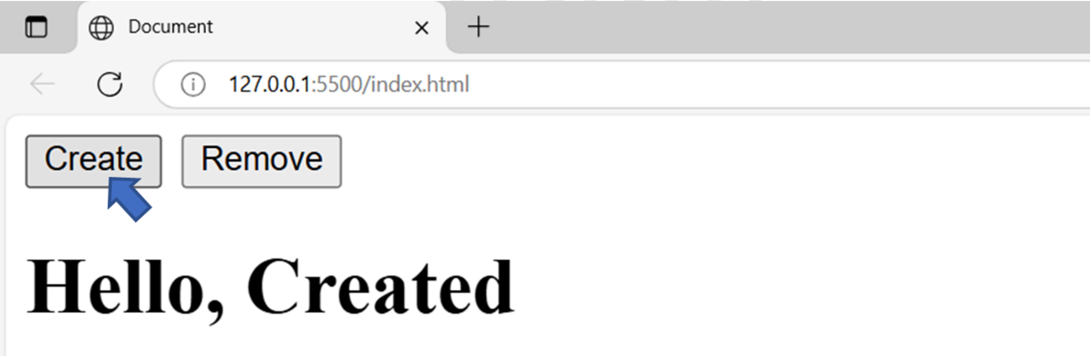
6) createAndRemoveElement

example.html

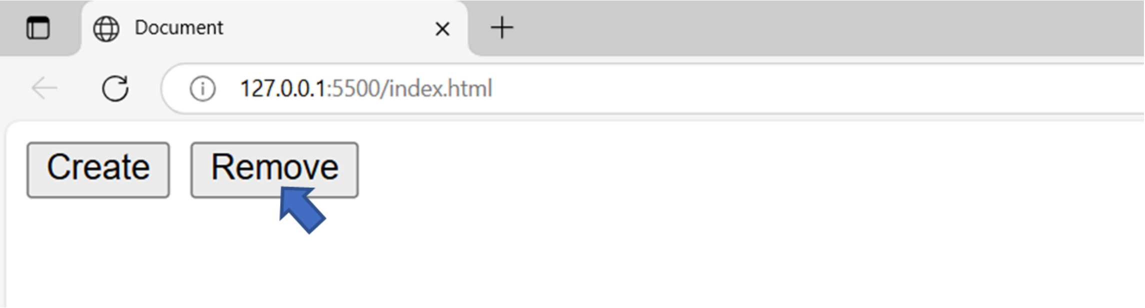
```
<input type="button" value="Create" onclick="create()">
<input type="button" value="Remove" onclick="remove()">
```

example.js

```
const heading = document.createElement('h1')
function create() {
        heading.innerHTML = 'Hello, Created'
        document.body.appendChild(heading)
}
function remove() {
        heading.remove()
}
```



Hello, Created



3. The Three Core Divisions of JavaScript

- Core (ECMAScript) –
    - The foundational part of JavaScript, defining basic syntax, data types, operators, control structures, and functions.
    - It forms the standard upon which other features are built.
- Client-Side JavaScript.
    - Used to create interactive and dynamic web pages by manipulating the DOM and handling user events.
    - Runs directly in the browser to enhance user experience.
    - Examples:
        - Form validation,
        - Animations, and
        - Real-time content updates.
- Server-Side JavaScript
    - Executes on the server using environments like Node.js.
    - Examples:
        - Handling HTTP requests,
        - Connecting to databases, and
        - Server-side rendering.

4. Uses of JavaScript

- Dynamic and Interactive Web Pages
    - Enables real-time updates, animations, and interactivity on websites.
    - Examples: Image sliders, dropdown menus, and modals.
- Manipulating HTML and CSS
    - Dynamically changes the structure, style, and content of a

webpage using the DOM.
    - Example : Changing the color of a button when clicked.
- Event Handling
    - Responds to user actions like clicks, hovers, keypresses, etc.
    - Example : Displaying a tooltip when hovering over an element.
- Backend Development
    - Used on the server-side with environments like Node.js to build web servers, APIs, and database applications.
- Game Development
    - Builds browser-based games using libraries like Phaser.js.
- Mobile App Development
    - Creates cross-platform mobile apps using frameworks like React Native.
- Building Web Applications
    - Develops single-page applications (SPAs) using frameworks like Angular, React, and Vue.js.

5. Java vs. JavaScript: Key Differences

- Type & Purpose
    - Java: A general-purpose, object-oriented programming language used for building standalone applications, enterprise software, and Android apps.
    - JavaScript: A scripting language primarily used for web development to create interactive and dynamic web pages.
- Execution Environment
    - Java: Runs in the Java Virtual Machine (JVM) and requires

compilation before execution.

- JavaScript: Runs in web browsers via an interpreter (JavaScript engine) like V8 (Chrome), SpiderMonkey (Firefox), or JavaScriptCore (Safari).

- Syntax & Paradigm
  - Java: Statically typed (variables must have a declared type), class-based, and strongly object-oriented.
  - JavaScript: Dynamically typed (variables can hold different types), prototype-based, and follows a mix of functional, procedural, and object-oriented programming.
- Usage
  - Java: Used in backend development, Android apps, desktop apps, and large-scale systems (e.g., banking, enterprise applications).
  - JavaScript: Mainly used in frontend development for making web pages interactive, but can also be used on the backend (Node.js).
- Concurrency Model
  - Java: Uses multi-threading for handling multiple tasks simultaneously.
  - JavaScript: Uses an event-driven, non-blocking (asynchronous) model with the single-threaded event loop.
- Platform Dependency
  - Java: "Write once, run anywhere" (WORA) due to the JVM.
  - JavaScript: Runs in browsers but can be used on servers with Node.js.

6. HTML And JavaScript: Understanding The Relationship

- Event-driven computation in JavaScript supports user interactions through HTML form elements on the client display.
- It allows developers to check the values provided in forms by users and perform input checks directly on the client side.
- By validating input on the client side, event-driven computation reduces the need for unnecessary server requests, saving both server processing time and internet bandwidth.
- HTML (Hyper Text Markup Language) and JavaScript work together to create modern, interactive web applications.
- When a JavaScript script is encountered in the HTML document, the browser uses its JavaScript interpreter to “execute” the script.

7. Embedding of JavaScript in HTML

- Explicit Embedding (Internal JavaScript)
  - In explicit embedding, the JavaScript code is directly written within the <script> tag in the HTML document.
  - This ensures that the script is visibly defined and isolated, making it easier to locate and manage.

```
<!DOCTYPE html>
<html>
        <head>
                <title>Explicit Embedding Example</title>
                <script>
                        function showMessage() {
                                alert("Explicit Embedding: Hello, World!");
                        }
```

```
            </script>
        </head>
        <body>
            <button onclick="showMessage()">
                Click Me
            </button>
        </body>
    </html>
```

- Implicit Embedding / Inline JavaScript
  - In implicit embedding, JavaScript is included directly within HTML attributes such as onclick, onmouseover, onload, etc.
  - It combines the script logic with the HTML structure, which can be convenient for quick actions but may make the code harder to maintain.

```
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <title>Implicit Embedding Example</title>
        </head>
        <body>
            <button onclick="alert('Implicit Embedding: Hello, World!’)”>Click Me
            </button>
        </body>
</html>
```

    - Explicit embedding is generally preferred for maintainability and scalability, while implicit embedding can be used for quick and simple tasks.

- External JavaScript
    - The script is stored in an external file and linked to the HTML using the src attribute of the <script> tag.
    - This method promotes code reuse and cleaner HTML.
    - External files help separate content from behavior, making the code more maintainable and modular.
    - Inline JavaScript is discouraged in modern development for reasons of maintainability and security.
    - Internal and external JavaScript remain common, with external being the preferred method.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>External JavaScript Example</title>
    </head>
    <body>
        <h1>Welcome to External JavaScript Example</h1>
        <button id="greetButton">Click Me</button>
        <script src="example.js"> </script>
    </body>
</html>
```

8. Object Orientation And JavaScript

- JavaScript is an object-based, prototype-driven language supporting encapsulation and inheritance.
- Unlike class-based languages, it uses prototypes for inheritance.
- While it lacks native polymorphism, dynamic typing and method overriding enable polymorphic behaviour, offering flexibility for object-oriented development.

9. JavaScript Reserved Words

- In JavaScript, reserved words are keywords that have special meaning in the language's syntax and cannot be used as identifiers (e.g., variable names, function names, or object properties).
- These words are predefined by the language and are used to define its structure and behavior.

  let, new, import, delete, finally, null, break, default, in, enum, for, return, case, do, instanceof, false, function, switch, catch, else, continue, class, this, const, try, typeof, var, void, while,

10. Declaring A Variable In JavaScript

- In JavaScript, variables can be declared using var, let, or const. Each has specific use cases and behaves differently in terms of scope, hoisting, and reassignment.
- Rules for Declaring Variables:
  - Must start with a letter, underscore (_), or dollar sign ($).
  - Cannot start with a number.
  - Can contain letters, digits, underscores, and dollar signs.

    - Cannot use reserved words (e.g., class, return, let).
- Best Practices:
    - Use const for variables that do not change.
    - Use let for variables that will be reassigned, and it is better to avoid using var in modern JavaScript development

Declaring a Variable with var

```
var x = 10;
var y;
y = 20;
console.log(x);
console.log(y);
```

Declaring a Variable with let

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

Declaring a Variable with const

```
const x = 10;
console.log(x); // 10
x = 20; // Error: Assignment to constant variable
```

11. Primitives, Operations, And Expressions

- JavaScript primitive types are the basic data types
- That is immutable (cannot be changed).
- Five primitive types: Number, String, Boolean, Undefined, and Null.

1. Number:

- Represents numeric values and includes both integers and floating-point numbers.
- Special values: Infinity, -Infinity, NaN (Not-a-Number).

```
let age = 25;
let price = 19.99;
let invalid = 0 / 0;
```

2. String:

- Represents a sequence of characters.
- Strings can be enclosed in single quotes ('), double quotes ("), or template literals (` `).

```
let name = 'Alice’;
let greeting = "Hello, World!";
let template = `Hi, ${name}`;
```

3. Boolean:

- Represents logical values, true or false.
- Boolean is often used in conditional statements.

```
let isAvailable = true;
let isEmpty = false
```

4. Undefined:

- A variable is undefined if it has been declared but not assigned a value.

```
let notAssigned;
console.log(notAssigned); // undefined
```

5. Null:

- Represents the intentional absence of any object value.
- It is often used to indicate that a variable has no value.

```
let emptyValue = null;
```

Type Checking:

- Use typeof to check the type of a variable:

```
console.log(typeof 42); //number
console.log(typeof 'Hello'); //string
console.log(typeof true); // boolean
console.log(typeof undefined); // undefined
console.log(typeof null); // "object"
                              (this is a known quirk of JavaScript)
```

Non Primitives Data types

- That is mutable (can be changed).
- Non primitive types:
  - Object,
  - Array
  - Functions

12. Numeric And String Literals

- In JavaScript, literals are fixed values that are directly written into the code, representing a specific type of data.
- There are two common types of literals in JavaScript: Numeric literals and String literals

1. Numeric Literals

- Numeric literals represent numbers in JavaScript, such as integers or floating-point values.

```
let integer = 42;

let float = 3.14;

let hex = 0xFF;

let binary = 011010;

let octal = 112;
```

2. String Literals

- A string literal is a sequence of characters enclosed in quotes.
- String literals represent text enclosed in single (‘), double ("), or backticks (` `).

```
let singleQuote = 'Hello';

let doubleQuote = "World";

let templateLiteral = `Hi ${singleQuote}`;
```

13. Numeric Operators

- Used to perform numeric operations on values.
- These operators work on numbers (either primitive values or numeric objects).
- Can be used with both integers and floating-point numbers.

1. Arithmetic Operators

- Arithmetic operators are used to perform basic mathematical operations.
    - Addition (+): Adds two operands.

      ```
      let sum = 5 + 3;
      console.log(sum);
      ```

    - Subtraction (-): Subtracts the second operand from the first.

      ```
      let difference = 5 - 3;
      console.log(difference);
      ```

    - Multiplication (*): Multiplies two operands.

      ```
      let product = 5 * 3;
      console.log(product);
      ```

    - Division (/): Divides the first operand by the second. If the divisor is 0, the result is Infinity or -Infinity.

      ```
      let quotient = 6 / 3;
      console.log(quotient); // 2
      let zeroDivision = 6 / 0;
      console.log(zeroDivision); // NaN
      ```

    - Modulo (%): Returns the remainder when the first operand is divided by the second. Often used to determine if a number is even or odd.

      ```
      let remainder = 5 % 2;
      console.log(remainder); //1
      let evenCheck = 4 % 2;
      console.log(evenCheck);
      ```

    - Exponentiation (**): Raises the first operand to the power of the second operand

      ```
      let power = 3 ** 2;
      console.log(power); //9
      ```

2. Assignment Operators

- Assignment operators are used to assign values to variables, with some performing operations as well.
    - Assignment (=): Assigns the value of the right operand to the left operand.

      ```
      let x = 10;
      console.log(x); // 10
      ```

    - Addition Assignment (+=): Adds the right operand to the left operand and assigns the result to the left operand.

      ```
      let x = 5;
      x += 3; // Equivalent to x = x + 3
      console.log(x); // 8
      ```

    - Subtraction Assignment (-=): Subtracts the right operand from the left operand and assigns the result to the left operand

```
let x = 5;

x -= 2; // Equivalent to x = x - 2

console.log(x); // 3
```

- Multiplication Assignment (*=): Multiplies the left operand by the right operand and assigns the result to the left operand.

```
let x = 5;

x *= 2; // Equivalent to x = x * 2

console.log(x); // 10
```

- Division Assignment (/=): Divides the left operand by the right operand and assigns the result to the left operand.

```
let x = 10;

x /= 2; // Equivalent to x = x / 2

console.log(x); // 5
```

- Modulo Assignment (%=): Divides the left operand by the right operand and assigns the remainder to the left operand

```
let x = 10;

x %= 3; // Equivalent to x = x % 3

console.log(x); // 1 (remainder of 10 divided by 3)
```

3. Increment and Decrement Operators

- These operators are used to increase or decrease a variable's value by 1.
  - Increment (++): Increases the operand's value by 1.
    - Prefix (++x): Increments x and then returns the value of x.

      ```
      let x = 5;
      console.log(++x); // 6 (increments first)
      console.log(x); // 6
      ```

    - Postfix (x++): Returns the value of x and then increments x.

      ```
      let x = 5;
      console.log(x++); // 5 (returns value first)
      console.log(x); // 6 (after increment)
      ```

  - Decrement (--): Decreases the operand's value by 1.
    - Prefix (--x): Decrements x and then returns the value of x.

      ```
      let x = 5;
      console.log(--x); // 4 (decrements first)
      console.log(x); // 4
      ```

    - Postfix (x--): Returns the value of x and then decrements x

      ```
      let x = 5;
      console.log(x--); // 5 (returns value first)
      console.log(x); // 4 (after decrement
      ```

4. Comparison Operators (Related to Numeric Values)

- Comparison operators are used to compare two values, often for control flow and decision-making.
    - Equal (==): Compares two values for equality, after type coercion.

      ```
      console.log(5 == "5"); //TRUE
      ```

    - Strict Equal (===): Compares two values for equality, without type coercion.

      ```
      console.log(5 === "5");  //FALSE
      ```

    - Not Equal (!=): Compares two values for inequality, after type coercion.

      ```
      console.log(5 != "5");  //FALSE
      ```

    - Strict Not Equal (!==): Compares two values for inequality, without type coercion.

      ```
      console.log(5 !== "5");  //TRUE
      ```

    - Greater Than (>): Checks if the left operand is greater than the right operand.

      ```
      console.log(5 > 3);  //TRUE
      ```

    - Greater Than or Equal (>=): Checks if the left operand is greater than or equal to the right operand.

      ```
      console.log(5 >= 5);  //TRUE
      ```

    - Less Than (<): Checks if the left operand is less than the right operand.

```
console.log(5 < 3);  //FALSE
```

  - Less Than or Equal (<=): Checks if the left operand is less than or equal to the right operand.

```
console.log(5 <= 3);  //FALSE
```

5. Unary Numeric Operators

- Unary operators work with a single operand.
  - Unary Plus (+): Converts the operand to a number.

```
let num = "5";
let result = +num; // Converts string to number
console.log(result); // 5
```

  - Unary Minus (-): Converts the operand to a number and negates it.

```
let num = 5;
let negNum = -num; // Converts number to negative
console.log(negNum); // -5
```

6. Ternary Operators

- A shorthand for if-else.
- Syntax: (condition)?'true’ : 'false'

```
let x = 10;
let y = 20;
let maxValue = (x > y) ? x : y;
let strOutput = (x > y) ? 'True Value' : 'Value is True';
console.log(maxValue);
console.log(strOutput);
```

7. Logical Operators

- Used to combine conditional statements.

```
const number1= 10;
const number2=6;
const isAnd = 10 > 8 && 8 < number2;
console.log(isAnd); // false
const isOr = number1 > 8 || 8 < number2;
console.log(isOr); // true
const isNot = !number1 > 8;
console.log(!isNot); // true
```

14. The Math Object

- The Math object in JavaScript provides a collection of built-in methods and properties for performing mathematical operations.
- All of the Math methods are referenced through the Math object, as in Math.sin(x).
  - Math.abs(x) Returns the absolute value of x.

    ```
    Math.abs(-3) → 3
    ```

  - Math.ceil(x) Rounds x upwards to the nearest integer.

    ```
    Math.ceil(3.2) → 4
    ```

  - Math.floor(x) Rounds x downwards to the nearest integer.

    Math.floor(3.8) → 3

  - Math.round(x) Rounds x to the nearest integer.

    Math.round(3.5) → 4

  - Math.max(x, y, ...) Returns the largest of the values.

    Math.max(1, 2, 3) → 3

  - Math.min(x, y, ...) Returns the smallest of the values.

    Math.min(1, 2, 3) → 1

  - Math.random() Returns a random number between 0 (inclusive) and 1.

    Math.random() → 0.423

  - Math.pow(x, y) Returns x raised to the power of y.

    Math.pow(2, 3) → 8

  - Math.sqrt(x) Returns the square root of x.

    Math.sqrt(16) → 4

  - Math.tan(x) Returns the tangent of x (in radians).

    Math.tan(Math.PI/4) → 1

  - Math.trunc(x) Returns the integer part of x.

    Math.trunc(3.9) → 3

15. Type Conversion

- Type conversions in JavaScript refer to the process of converting a value from one data type to another.
- This can happen either implicitly (type coercion) or explicitly (type casting).

1. Implicit Type Conversions

- Implicit Type Conversions in JavaScript refer to the automatic conversion of one data type to another by JavaScript when performing operations.

String + Number: Converts Number to String

let result = “5” + 3 = “53” // string

String - Number: Converts String to Number

let result = “10” - 5 = 5 // number

Boolean to Number: true becomes 1, false become 0

let result = true + 3 = 4

2. Explicit Conversion

- You manually convert types using methods.
- Implicit conversions can lead to unexpected results.
- So explicit conversion is preferred for clarity.
- False values like (e.g., 0, "", null, undefined, NaN) become false when converted to Boolean.

To String

let num = 42;

let str = String(num); "42"

To Number

let str = "3.14";

let num = Number(str); 3.14

To Boolean

let value = 0;

```
let bool = Boolean(value); false
```

16. Screen Output And Keyboard Input

Methods for Screen Output

1. console.log():
   - Outputs to the browser console (for debugging).

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

2. document.write():
   - Writes directly to the webpage.

   ```
   document.write("Hello, Page!");
   ```

3. alert():
   - Displays a popup alert box.

   ```
   alert("Hello, User!");
   ```

4. innerHTML:
   - Updates the content of an HTML element.

   ```
   document.getElementById("output").innerHTML = "Hello, DOM!";
   ```

Methods for Capturing Keyboard Input

JavaScript uses different input methods to capture user data.

1. prompt():
   - Displays a popup to collect user input as a string.

   ```
   let name = prompt("Enter your name:");
   console.log("Hello, " + name);
   ```

2. Event Listeners for Input Fields:
   - Can be used to capture input from HTML form elements

(e.g., text boxes).

17. String Properties And Methods

- In JavaScript, strings are immutable sequences of characters. Strings come with several properties and methods
- that allow for various operations like searching, extracting, and manipulating text.

1. Length Property

- Returns the number of characters in a string (including spaces and special characters).

```
let str = "Hello, World!";
console.log(str.length); // 13
```

2. String Methods

1. toUpperCase():

   - Converts a string to uppercase.

   ```
   let text = "hello";
   console.log(text.toUpperCase()); // "HELLO"
   ```

2. toLowerCase():

   - Converts a string to lowercase.

   ```
   let text = "HELLO";
   console.log(text.toLowerCase()); // "hello“
   ```

3. charAt(index):

   - Returns the character at the specified index.

   ```
   let text = "JavaScript";
   ```

```
console.log(text.charAt(4)); // "S"
```

4. includes(substring):

- Checks if a string contains a substring (returns true or false).

```
let text = "JavaScript is fun";
console.log(text.includes("fun")); // true
```

5. indexOf(substring):

- Returns the position of the first occurrence of a substring, or -1 if not found.

```
let text = "Hello, World!";
console.log(text.indexOf("World")); // 7
```

6. slice(start, end):

- Extracts a part of a string (end is optional).

```
let text = "JavaScript";
console.log(text.slice(0, 4)); // "Java“
```

7. replace(search, replaceWith):

- Replaces the first occurrence of a substring.

```
let text = "I love JavaScript";
console.log(text.replace("JavaScript", "coding")); // "I love coding“
```

8. split(separator):

- Splits a string into an array based on a separator.

```
let text = "Apple, Banana, Cherry";
console.log(text.split(",")); // ["Apple", "Banana", "Cherry"]
```

9. trim():

- Removes whitespace from both ends of a string.

```
let text = " Hello World! ";

console.log(text.trim()); // "Hello World!"
```

10. concat(str1, str2, ...):

- Combines two or more strings.

```
let str1 = "Hello";

let str2 = "World";

console.log(str1.concat(", ", str2, "!")); // "Hello, World!"
```

18. Control Statements

- In JavaScript, control statements are used to control the flow of execution based on conditions or loops.
- They Include;
    - Decision-making statements.
    - Looping statements.
    - Branching statements.

Decision-Making Statements

1. if Statement:

- Executes a block of code if the condition is true.

```
let age = 18;

if (age >= 18) {

        console.log("You are eligible to vote.");

}
```

2. if...else Statement:

- Executes one block of code if the condition is true, and another if false.

```
let age = 16;

if (age >= 18) {

        console.log("You are eligible to vote.");

} else {

        console.log("You are not eligible to vote.");

}
```

3. if...else if...else Statement:

- Tests multiple conditions sequentially.

```
let marks = 85;

if (marks >= 90) {

        console.log("Grade: A+");

} else if (marks >= 75) {

        console.log("Grade: A");

} else {

        console.log("Grade: B");

}
```

4. switch Statement:

- Executes code based on the value of an expression.

```
let day = 3;
switch (day) {
    case 1:
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
        break;
    case 3:
        console.log("Wednesday");
        break;
    default:
        console.log("Invalid day");
}
```

Looping Statements

- These statements are used to repeat code.

1. for Loop:

- Repeats a block of code a fixed number of times.

```
for (let i = 1; i <= 5; i++) {
    console.log("Iteration:", i);
}
```

2. for..in Loop:

- Iterates over the properties of an object.

```
let person = { name: "Alice", age: 25 };
for (let key in person) {
    console.log(key + ":", person[key]);
}
```

3. for..of Loop:

- Iterates over the values of an iterable (e.g., arrays, strings).

```
let arr = [10, 20, 30];
for (let value of arr) {
    console.log(value);
}
```

4. while Loop:

- Repeats as long as the condition is true.

```
let i = 1;
while (i <= 5) {
    console.log("Iteration:", i);
    i++;
}
```

5. do..while Loop:

- Executes at least once, repeats while the condition is true.

```
let i = 1;
do {
        console.log("Iteration:", i);
        i++;
    }
    while (i <= 5);
```

Branching Statements

- Alter the flow of execution.

1. break Statement:

- Exits a loop or switch statement.

```
for (let i = 1; i <= 5; i++) {
    if (i === 3) break;
    console.log(i); // Stops at 2
}
```

2. continue Statement:

- Skips the current iteration and moves to the next one.

```
for (let i = 1; i <= 5; i++) {
    if (i === 3) continue;
    console.log(i); // Skips 3
}
```

3. Conditional (Ternary) Operator:

- A shorthand for if..else.

```
let age = 18;
let result = (age >= 18) ? "Adult" : "Minor";
console.log(result); // "Adult"
```

19. Event Handling in JavaScript

Event Listener

- The most common way to handle events in JavaScript is by using addEventListener().
- Syntax:
    - element.addEventListener(event, function, useCapture);
- event: The type of event (e.g., "click", "keydown").
- function: The function to be called when the event occurs.
- useCapture: Optional parameter

```
const btn = document.getElementById('myButton');
btn.addEventListener('click', () => {
        alert("Button clicked!");
});
```

Event Types

- Common event types include:
    - Mouse Events: click, dblclick, mouseover, mouseout
    - Keyboard Events: keydown, keyup, keypress
    - Form Events: submit, focus, blur, input
    - Window Events: resize, scroll, load

Inline Event Handlers

- You can also handle events inline within HTML attributes,
- although using addEventListener() is preferred.

```
<button onclick="alert('Button clicked!')">Click me</button>
```

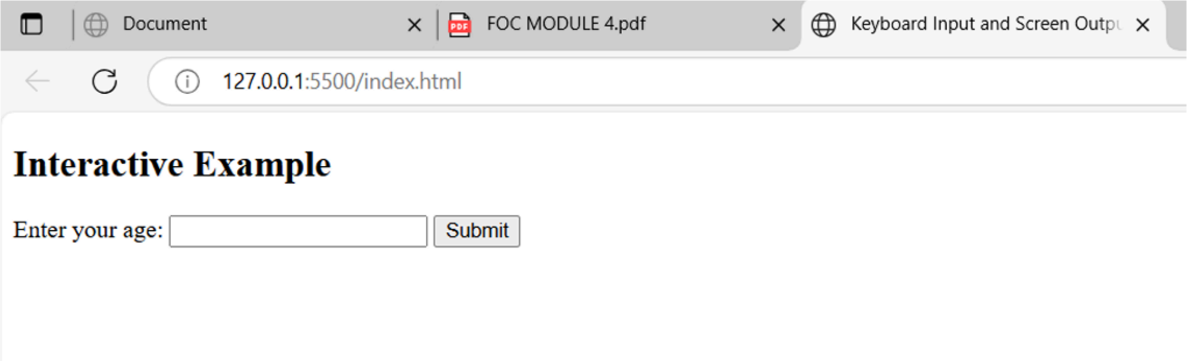
Removing Event Listeners

- You can remove an event listener using removeEventListener().

```
const btn = document.getElementById('myButton’);
function handleClick() {
        alert('Button clicked!');
}
btn.addEventListener('click', handleClick);
btn.removeEventListener('click', handleClick);
```

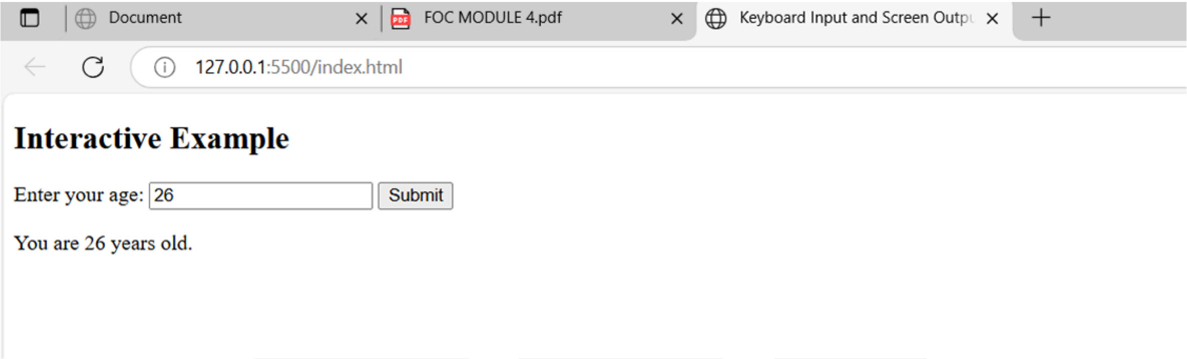
Example- 1

```
<!DOCTYPE html>
<html>
        <head>
                <title>Keyboard Input and Screen Output</title>
        </head>
        <body>
                <h2>Interactive Example</h2>
                <label for="age">Enter your age:</label>
                <input type="number" id="age">
                <button onclick="document.getElementById('result').innerText =
                'You are ' + document.getElementById('age').value + ' years old.'">
                Submit </button>
                <p id="result"></p>
        </body>
</html>
```



Interactive Example

Enter your age:



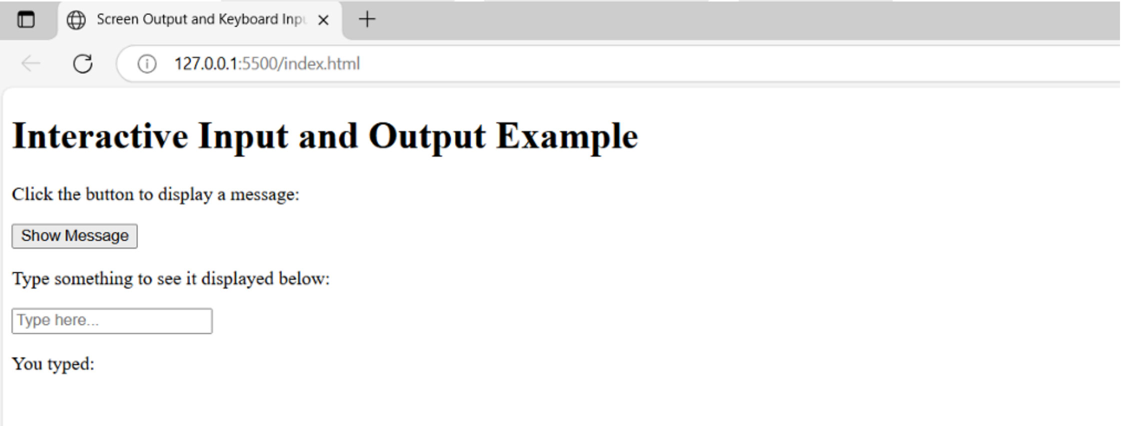
Interactive Example

Enter your age: 26

You are 26 years old.

Example- 2

```
!DOCTYPE html>
<html>
        <head>
                <title>Screen Output and Keyboard Input</title>
        </head>
        <body>
                <h1>Interactive Input and Output Example</h1>
                <!-- Screen Output Section →
                <p id="outputMessage">
                        Click the button to display a message:
                </p>
                <button
                onclick="document.getElementById('outputMessage').innerText =
                'Hello! This is a message displayed using JavaScript.'">

                Show Message </button>

                <!-- Keyboard Input Section →

                <p>Type something to see it displayed below:</p>

                <input type="text" id="userInput" placeholder="Type here..."
                onkeyup="document.getElementById('displayText').innerText =
                'You typed: ' + this.value;">

                <p id="displayText">You typed: </p>

        </body>

</html>
```

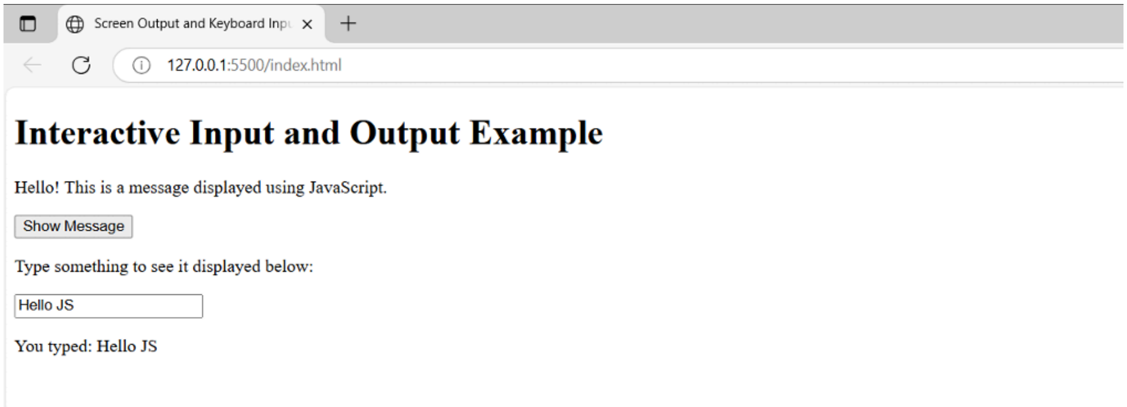


Interactive Input and Output Example

Click the button to display a message:

Type something to see it displayed below:

You typed:



Interactive Input and Output Example

Hello! This is a message displayed using JavaScript.

Type something to see it displayed below:

Hello JS

You typed: Hello JS