JAVASCRIPT

# Introduction

à JS is light weight interpreted and Just in time compiled programming language.

* Light weight : memory occupied & how heavy the application is
* Interpreted : line by line translation of language
* Compiled – translating entire program at a time
* JIT [Just in time] JS code is loaded into browser & complied in browser

JavaScript is a language which is used at

* Client side: (with HTML)
* Server side: (with Node JS)
* Database: (with Mongo DB)
* Animation Tools: (Flash, 3D Max, etc.)

JavaScript supports various programming techniques and approaches:

* Structural programming
* Functional programming
* Imperative programming
* Object Oriented programming etc.

What is the role of JS with HTML:

* For DOM Manipulation (Adding, removing, updating elements from page)
* Client side validations
* Client side interactions
* Handling plugins & extensions

What is the role of JS Server side:

* Server side Interactions(Requests & Response)
* Server side Objects(Managing Memory, OS, Files on server)
* Configuring API’s
* Handling communication b/w Client & Server

# Environment Setup

1. Install “Node JS”

Node JS: Server side programming

Node Compiler: Used to compile & run JS program

NPM : Node Package Manager

Package Manger (Software tools used by developer to install, update, remove libraries from project)

<http://nodejs.org/en/>

Download & install LTS (18x Version)

Check NODE JS Version (node -v)

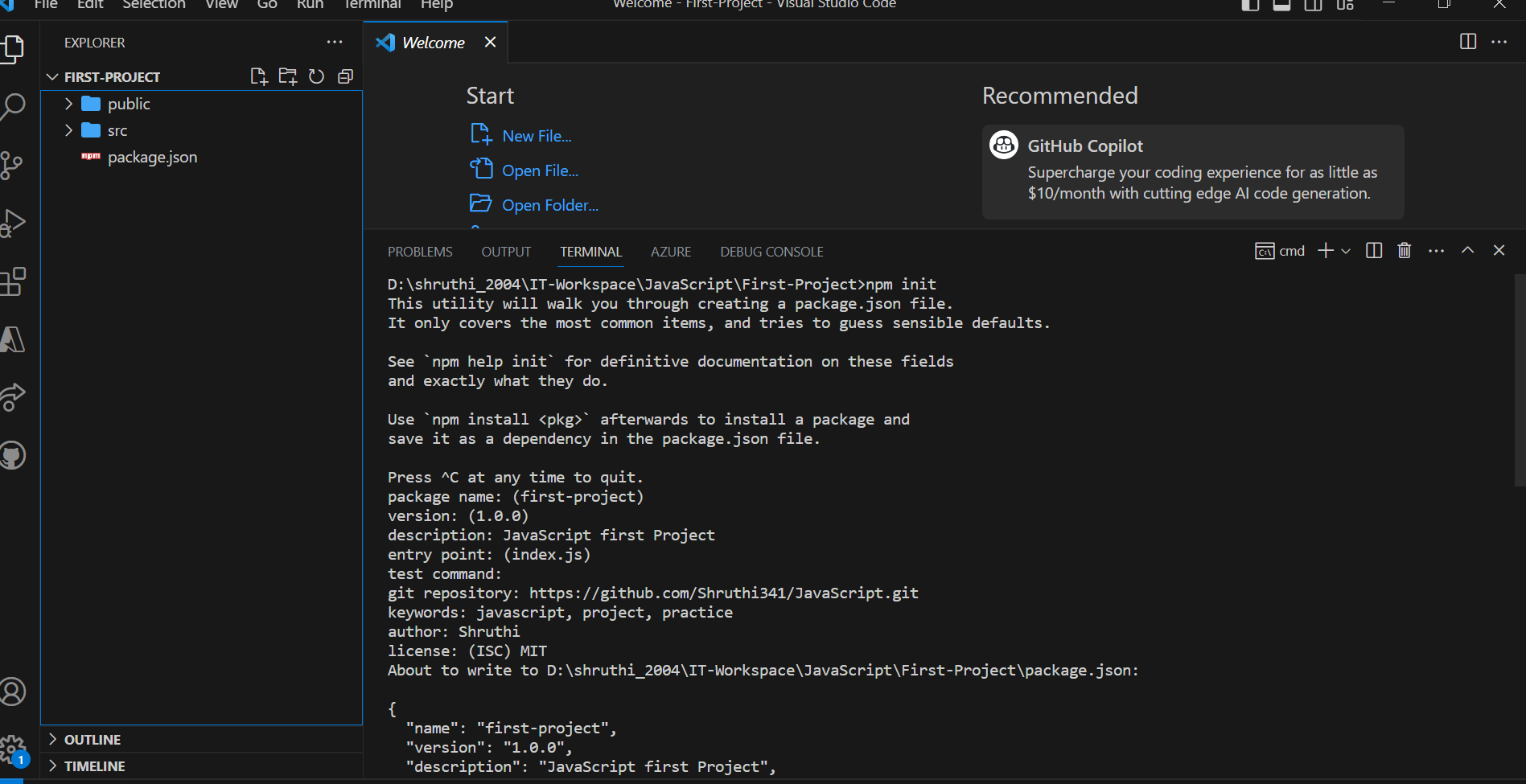
Check package manager version (npm -v)

1. Download “VS Code” (Extendions: Live Server, VsCode Icons)

# Project Setup

**METHOD - 1**

1. Create a js folder for project
2. Open in VS Code
3. Add following folders into project
4. Public: It is used to keep all static resources (html,images,text,docx,mp4,etc)
5. Src: It is used to keep all dynamic resources( js,ts, css, saas, less, etc.)
6. Open Terminal in Vs Code
7. Give (npm init) and give answers to questions



1. Package.json file will be generated

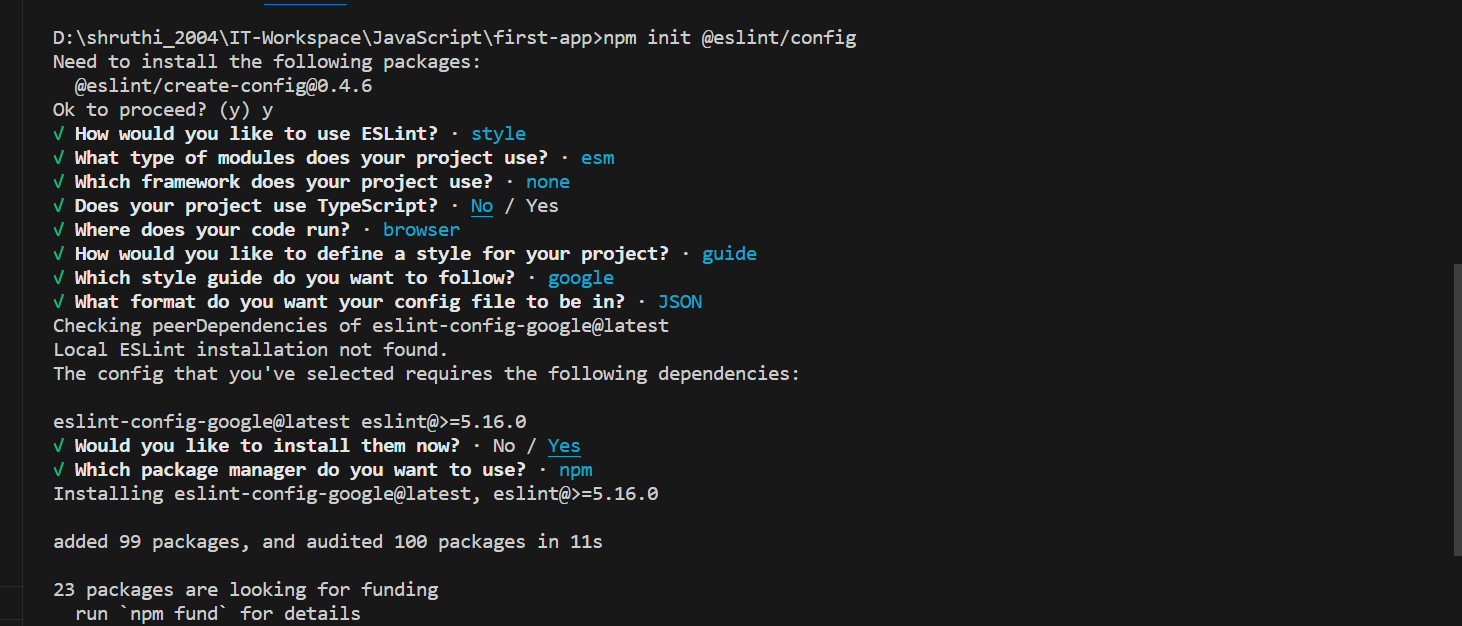
**METHOD – 2**

Open cmd and give (npm init -y)

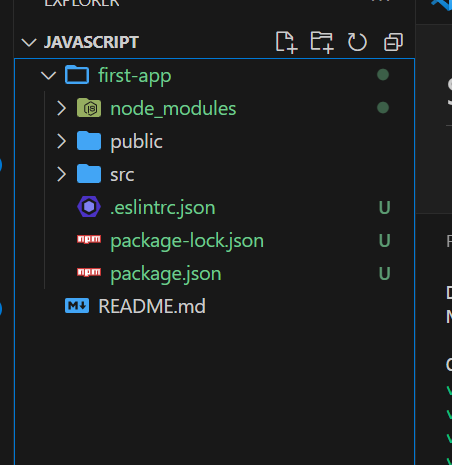
By default it will take yes to all & give package.json

1. Install Eslint configuration (verify code standards)

npm init @eslint/config



1. Project Structure



How to convert static DOM into dynamic DOM ?

By integrating JS functions into page, JS converts static DOM into Dynamic DOM.

3 Ways to embed JS into Page:

* Inline: functions written inside html elements start tag

<button onclick=”window.print()”>Print </button>

It is faster in responding

Not good for reusability

* Embedded: will use functions present in <script> container

Can reuse

But slower compared to inline

Script container can be in <head> or <body> section

<script>

Function PrintPage(){

Window.print()

}

</script>

<button onclick=”PrintPage()”>Print</button>

...

<button onclick=”PrintPage()”>Printer</button>

* Script in head: intended to load into browser memory and later accessed by the page.
* Script in body; intended to load into page directly, it is not in memory of browser.
* External

<html>

<head>

<title>

Embedded JavaScript Example

</title>

<script src="myscript.js"></script>

</head>

<body>

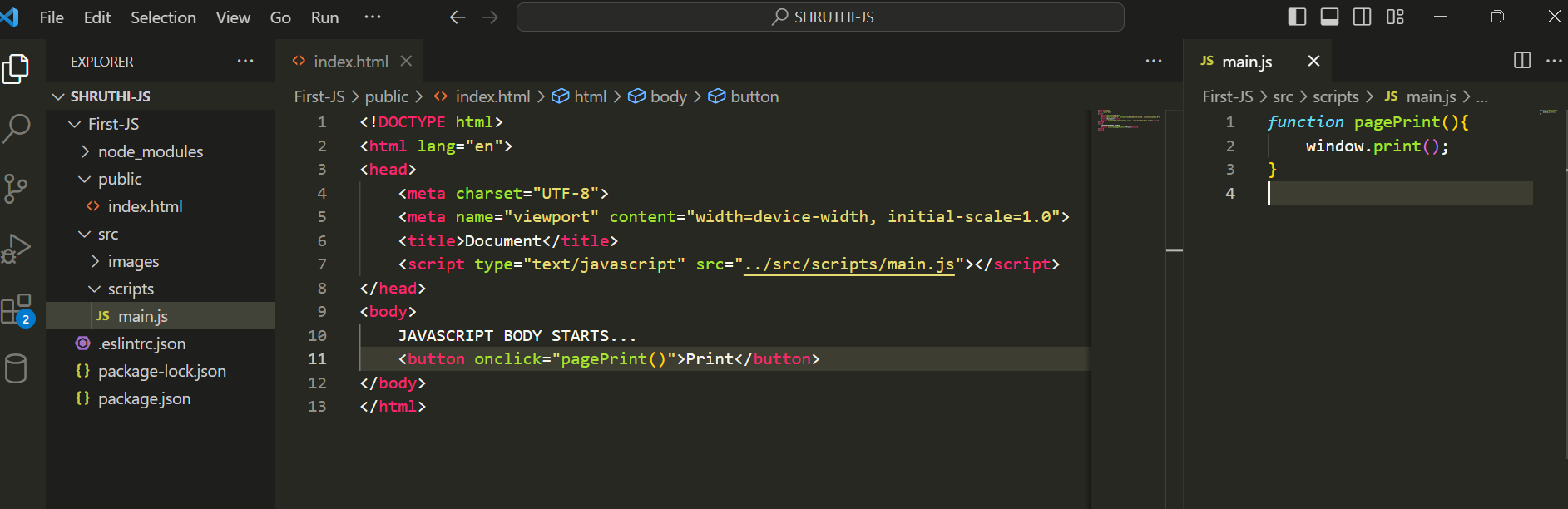
<!-- Body content here -->

</body>

</html>

myscript.js

alert("Hello, World!");



# Strictly Typed?

* Not implicitly strictly typed.
* Can manually turn on strictly type mode

Example:

<script>

x =10;

document.write("x=",x);

<script>

Output:

X = 10

But,

<script>

“use script”

x =10;

document.write("x=",x);

<script>

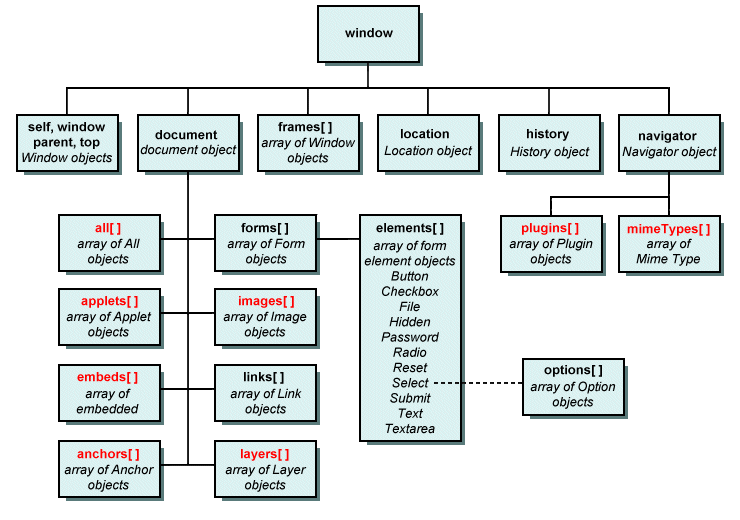
Output:

Errror (x is not defined)

# Minification

* Coding technique used to reduce size of file[compress].
* Minified files are used in Production
* Un-minified files are used in Development
* <https://www.toptal.com/developers/javascript-minifier>

# DOM Heirarchy



Example: How javascript takes control over HTML. Dynamic Content

project-root/

|-- src/

| |-- images/

| | |-- profile.jpg

| |-- scripts/

| | |-- dom-content-load.js

|-- public/

| |-- index.html

|-- .gitignore

|-- README.md

Index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script type="text/javascript" src="../src/scripts/dom-content-load.js"></script>

</head>

<body onload="contentload()">

<div>

<form>

<h2>Register</h2>

Your Email <input type="email"/> <input type="button"/>

</form>

</div>

<div>

<form>

<h2>Login</h2>

Your Mobile <input type="text"/> <input type="button"/>

</form>

</div>

</body>

</html>

Dom-content-load.js:

function contentload(){

window.document.forms[0].elements[1].value="Register";

window.document.forms[1].elements[1].value="Login";

}

Output:



Javascript can also refer dom elements by name:

--> Every element defined with a name

--> Can’t access any child element directly

-->Need to use parent name for child elements

<body onload="contentloadByname()">

<div>

<form name="formRegister">

<h2>Register</h2>

Your Email <input type="email"/> <input type="button" name="btn1"/>

</form>

</div>

<div>

<form name="formLogin">

<h2>Login</h2>

Your Mobile <input type="text"/> <input type="button" name="btn2"/>

</form>

</div>

</body>

Js file:

function contentloadByname(){

formRegister.btn1.value = "Button1";

formLogin.btn2.value = "Button2";

}

Output:



# Output Techniques

In JavaScript, there are several techniques for producing output, and the choice often depends on the context in which you're working. Here are some common techniques:

* **Console Output:**

**console.log():** This method is widely used for logging information to the browser's console. It's commonly used for debugging purposes.

* console.log("Hello, World!");
* **Alerts:**
* **alert():** This method displays a dialog box with a specified message and an OK button. It's often used for simple user notifications, but it can be intrusive.
* alert("Hello, World!");
* **Writing to HTML:**

**document.write():** This method writes HTML expressions or JavaScript code to a document. Be cautious with its use, as it can overwrite the entire document if called after the document has fully loaded.

* document.write("Hello, World!");
* **DOM Manipulation:**

**innerHTML:** You can manipulate the HTML content of an element using the **innerHTML** property.

* document.getElementById("outputElement").innerHTML = "Hello, World!";
* **Creating Elements:**

You can dynamically create elements and append them to the document.

var newDiv = document.createElement("div");  
newDiv.innerHTML = "Hello, World!";  
document.body.appendChild(newDiv);

# Input Techniques

* **Input Elements:**
* Use HTML input elements like textboxes, checkboxes, radio buttons, etc., and then access their values using JavaScript.

<input type="text" id="textInput" />  
<button onclick="showInput()">Show Input</button>

* javascriptCopy code
* function showInput() {  
   var userInput = document.getElementById("textInput").value;  
   alert("User Input: " + userInput);  
  }
* **Event Listeners:**

Attach event listeners to HTML elements to respond to user actions (e.g., clicks, keypresses).

* htmlCopy code

<button id="myButton">Click Me</button>

* javascriptCopy code

document.getElementById("myButton").addEventListener("click", function() {  
 alert("Button Clicked!");  
});

* **Forms:**
* Use HTML forms and handle form submissions to gather user input.
* htmlCopy code

<form id="myForm">  
 <input type="text" name="username" />  
 <input type="submit" value="Submit" />  
</form>

* javascriptCopy code

document.getElementById("myForm").addEventListener("submit", function(event) {  
 event.preventDefault(); // Prevents the form from submitting and refreshing the page  
 var username = event.target.elements.username.value;  
 alert("Username: " + username);  
});

* **Prompt Dialog:**
* Use the **prompt** function to get input directly from the user via a dialog box.
* javascriptCopy code

var userInput = prompt("Enter something:");  
alert("User Input: " + userInput);

* **Confirm Dialog:**
* Use the **confirm** function to get a yes/no response from the user via a dialog box.
* javascriptCopy code

var userConfirmation = confirm("Do you want to proceed?");  
if (userConfirmation) {  
 alert("User confirmed!");  
} else {  
 alert("User cancelled.");  
}

* **Reading from URL:**
* Access query parameters or hash fragments from the URL to get input.

// Assuming the URL is something like "index.html?param=value"

var urlParams = new URLSearchParams(window.location.search);

var paramValue = urlParams.get("param");

alert("Parameter Value: " + paramValue);

# Bootstrap

Install bootstrap in project

> npm install bootstrap --save

> npm install bootstrap-icons --save

>npm install jquery –save

Link bootstrap css:

Node\_module:

|-bootstrap/dist/css/boostrap.css

|-bootstrap-icons/font/bootstrao-icons.css

Link js:

Jquery/dist/jquery.js

Example:-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="../node\_modules/bootstrap//dist/css/bootstrap.css">

<link rel="stylesheet" href="../node\_modules/bootstrap-icons/font/bootstrap-icons.css">

</head>

<body class="container-fluid">

<h2>Bootstrap JS</h2>

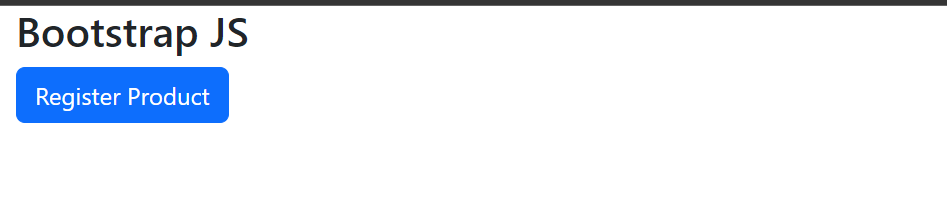
<button class="btn btn-primary">Register Product</button>

<script src="../node\_modules/jquery/dist/jquery.js"></script>

<script src="../node\_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>

</body>

</html>



# Example: Input Form & Display

Lets

* create a simple button to open Register form
* When form is Submitted display Form Details & Disable Register Form.
* When clicked on Edit button in details Open Register Form Again.

Html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="../node\_modules/bootstrap//dist/css/bootstrap.css">

<link rel="stylesheet" href="../node\_modules/bootstrap-icons/font/bootstrap-icons.css">

<script type="text/javascript" src="../src/scripts/input-demo-boostrap.js"></script>

</head>

<body class="container-fluid">

<button class="btn btn-primary mt-1" onclick="getRegisterForm() " id="initialButton">Register Product</button>

<div id="RegisterContainer" style="display: none;">

<h3>Register Form</h3>

<div class="form-group col-md-4">

Product Name: <input type="text" id="Name" class="form-control">

</div>

<div class="form-group col-md-4">

Price: <input type="number" id="Price" class="form-control">

</div>

<div class="form-group col-md-4">

City:

<select id="City" class="form-control">

<option>Delhi</option>

<option>Hyderabad</option>

<option>Mumbai</option>

</select>

</div>

<div class="form-group col-md-4">

Stock: <br>

<input type="checkbox" id="Stock" class="form-check-input"> Available

</div>

<button class="btn btn-success" onclick="RegisterClick()">Register</button>

</div>

<div id="DetailsContainer" style="display: none;">

<h3>Product Details</h3>

<b>Name : </b><span id="lblName"></span><br>

<b>Price : </b><span id="lblPrice"></span><br>

<b>City : </b><span id="lblCity"></span><br>

<b>Stock : </b><span id="lblStock"></span><br>

<button class="btn btn-warning" onclick="onEdit()">Edit</button>

</div>

<script src="../node\_modules/jquery/dist/jquery.js"></script>

<script src="../node\_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>

</body>

</html>

JavaScript:

function getRegisterForm(){

document.getElementById("DetailsContainer").style.display = "none";

document.getElementById("RegisterContainer").style.display = "block";

document.getElementById("initialButton").style.display = "none";

}

function RegisterClick(){

document.getElementById("DetailsContainer").style.display = "block";

document.getElementById("RegisterContainer").style.display = "none";

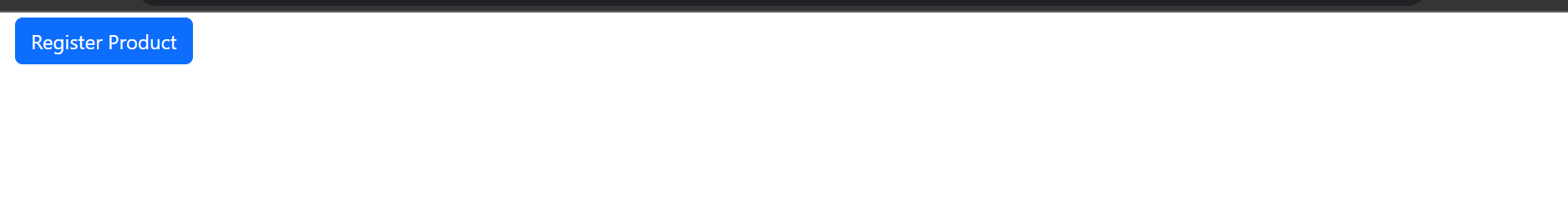
}

function onEdit(){

document.getElementById("RegisterContainer").style.display = "block";

document.getElementById("DetailsContainer").style.display = "none";

}



On Register Product Button Click:



On Register Click:



On Edit Again it displays Register Page.

To display form Details Update function of Submit,

function RegisterClick(){

document.getElementById("DetailsContainer").style.display = "block";

document.getElementById("RegisterContainer").style.display = "none";

//Display Details

document.getElementById("lblName").innerHTML = document.getElementById("Name").value;

document.getElementById("lblPrice").innerHTML = document.getElementById("Price").value;

document.getElementById("lblCity").innerHTML = document.getElementById("City").value;

stockStatus = "";

stockCheck = document.getElementById("Stock");

if(stockCheck.checked){

stockStatus ="Available";

document.getElementById("lblStock").innerHTML = stockStatus;

}

else{

stockStatus ="Out of Stock";

document.getElementById("lblStock").innerHTML = stockStatus;

}

}





# JavaScript Basics

-> Variables

-> Data Types

-> Operators

-> Statements

-> Functions

### 1. Variables:

Variables are storage locations in memory, where we can store a value & use it.

Variables in JavaScript are used to store and manage data in a program.

They are like containers that hold values, and these values can be of various types such as numbers, strings, objects, etc.

--> Variables have 3 phases

> **Declaration**

Var x;

> **Assignment**

X = 10;

> **Initialization**

Var X = 10;

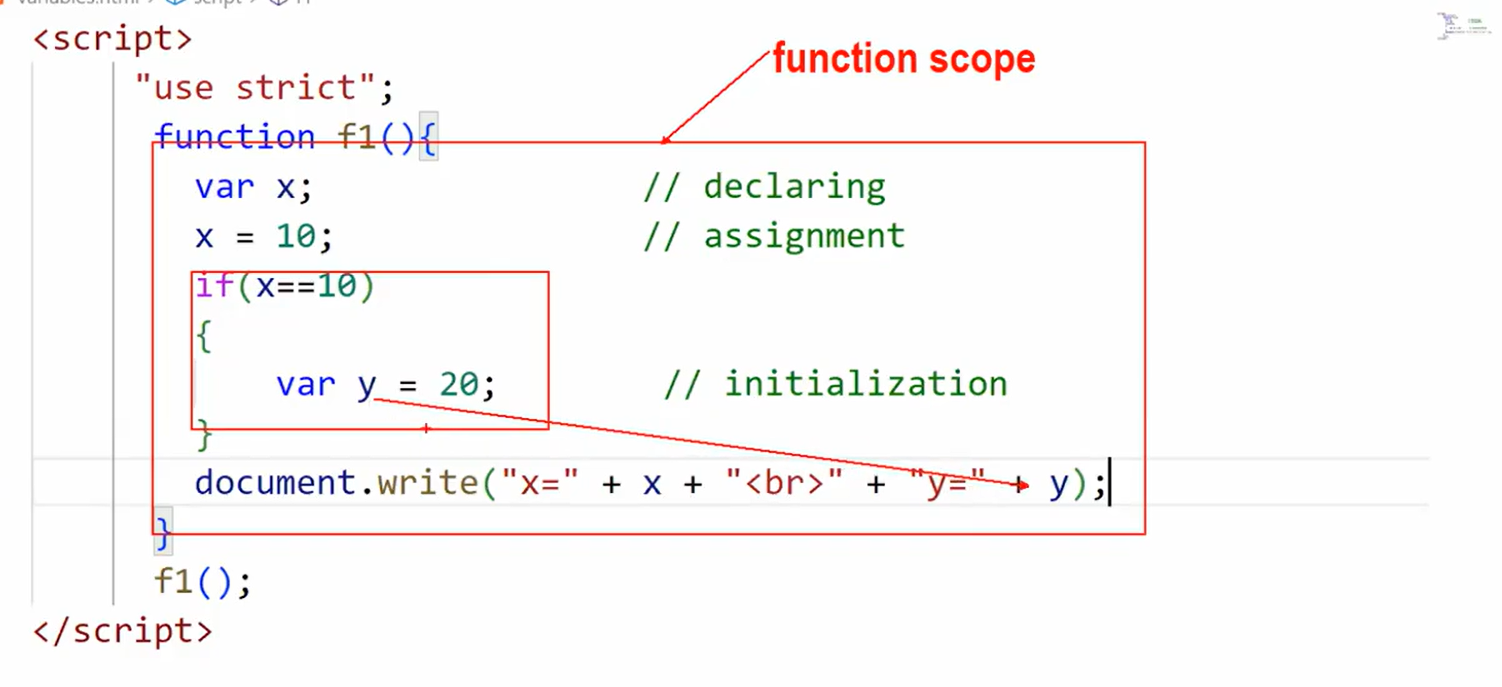
In JavaScript, you can declare variables using the **var**, **let**, or **const** keyword.

**Var:**

This was the original way to declare variables in JavaScript, and it has function scope (not block scope).

If you declare a variable using **var** inside a function, it will be accessible throughout the entire function, even if it's declared inside a block.

**var** variables are also hoisted, which means they are moved to the top of their scope during the compilation phase.



function example() {

if (true) {

var x = 10;

}

console.log(x); // 10

}

**Let:**

Introduced in ECMAScript 6 (ES6), **let** allows you to declare variables with block scope.

This means the variable is only accessible within the block, statement, or expression where it is defined.

**let** variables are not hoisted to the top of their scope.

function example() {

if (true) {

let y = 20;

console.log(y); // 20

}

// console.log(y); // Error: y is not defined

}

**Const:**

Also introduced in ES6, **const** is used to declare constants, and like **let**, it has block scope.

Once a value is assigned to a **const** variable, it cannot be reassigned.

However, note that if the value is an object or an array, the properties (or elements) of the object or array can still be modified.

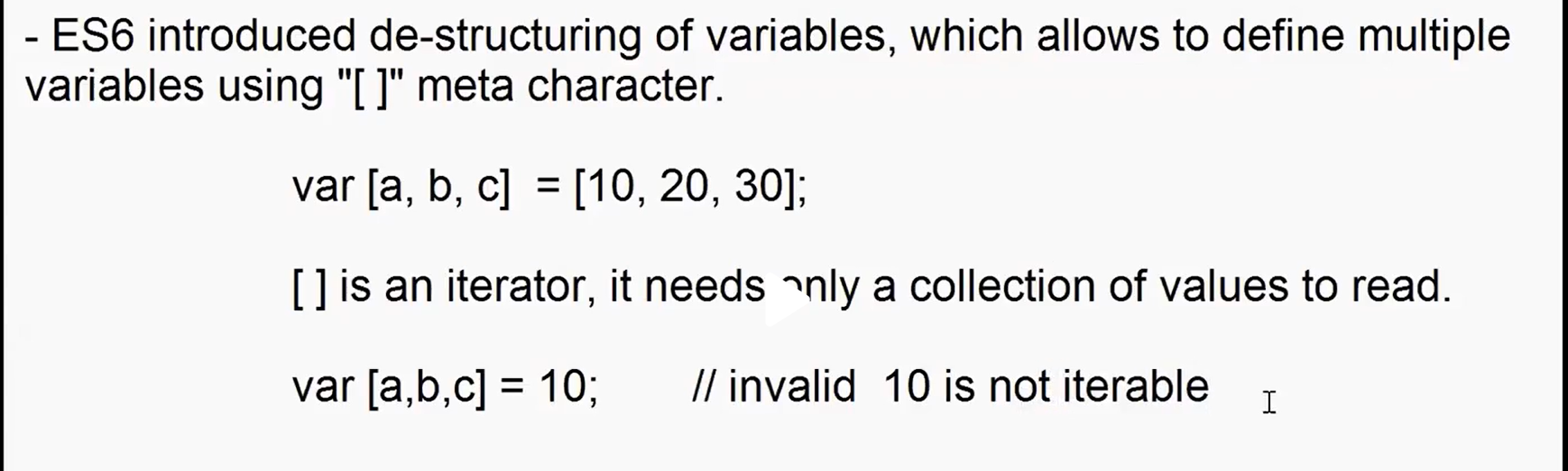
Example:

const PI = 3.14;

// PI = 3.14159; // Error: Assignment to a constant variable

const myArray = [1, 2, 3];

myArray.push(4); // This is allowed



# Data Types

JavaScript is a dynamically typed language, which means that variables can hold values of any type, and the type of a variable can change during runtime.

JavaScript has several built-in data types, which can be broadly categorized into primitive types and object types.

**Primitive Data Types:**

Primitive data types in JavaScript are basic data types that are not objects and do not have methods. They are the most fundamental building blocks for storing and manipulating data. Here are the primitive data types in JavaScript:

1. **Number:**

* Represents numeric values, including integers and floating-point numbers.
* Signed Integer, Unsigned Integer, float, Double, Decimal, Exponent, Binary, etc.
* Not strongly typed, so we explicitly verify the type of number using “isNaN()”.

<script>

var age = prompt(“Enter Age”);

if(isNan(age)){

document.write(“Age must be number”);

}else{

document.write(“Your Age: ”+age);

}

<script>

1. **String:**

* String is a literal with group of characters enclosed in a double quote, single quote, backticks.

<script>

var username = prompt("Enter Name");

var age = parseInt(prompt("Enter Age"));

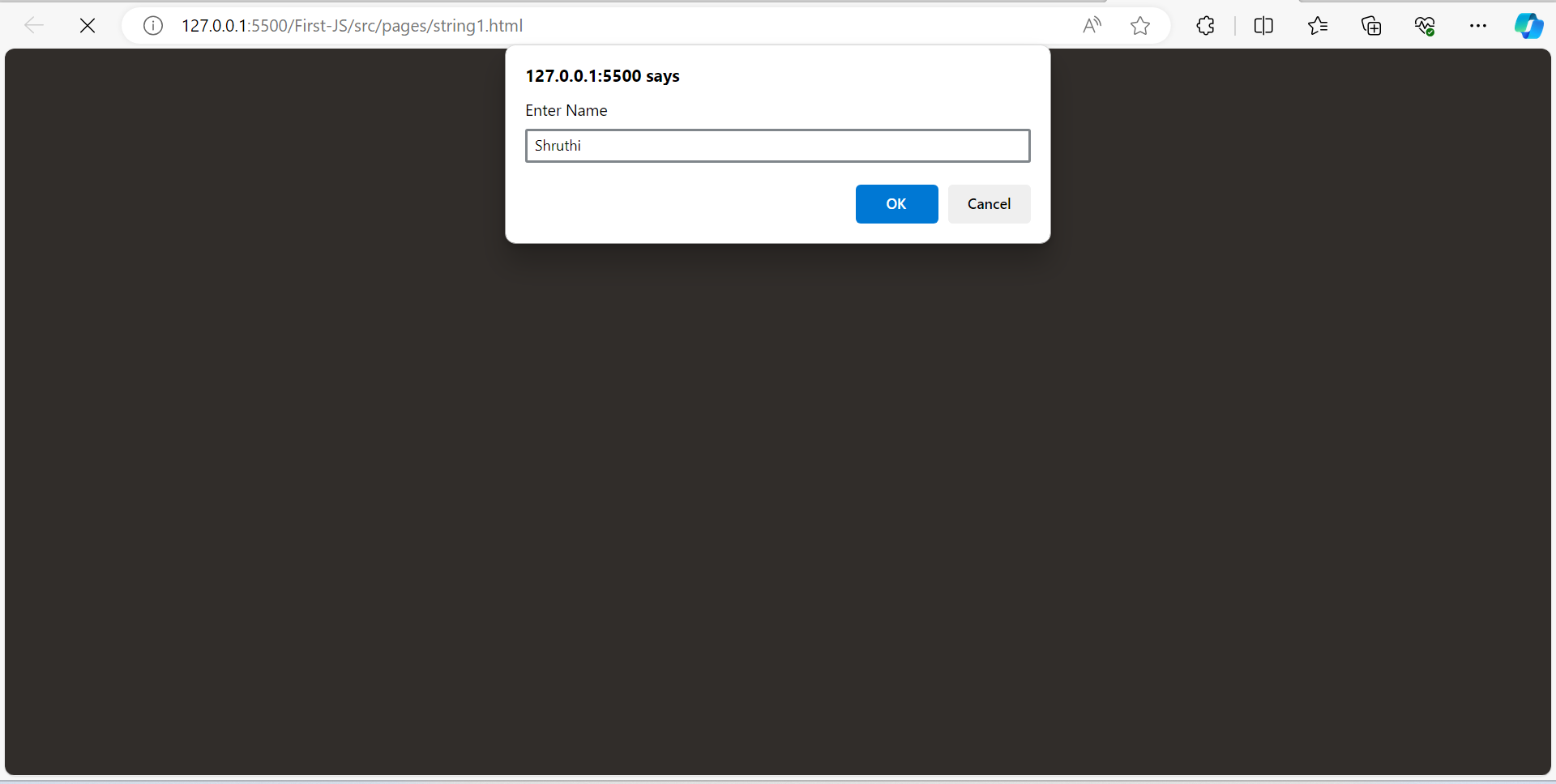
var msg1 = "Hello!"+" "+username+","+" "+"you are "+age+" years old.<br>";

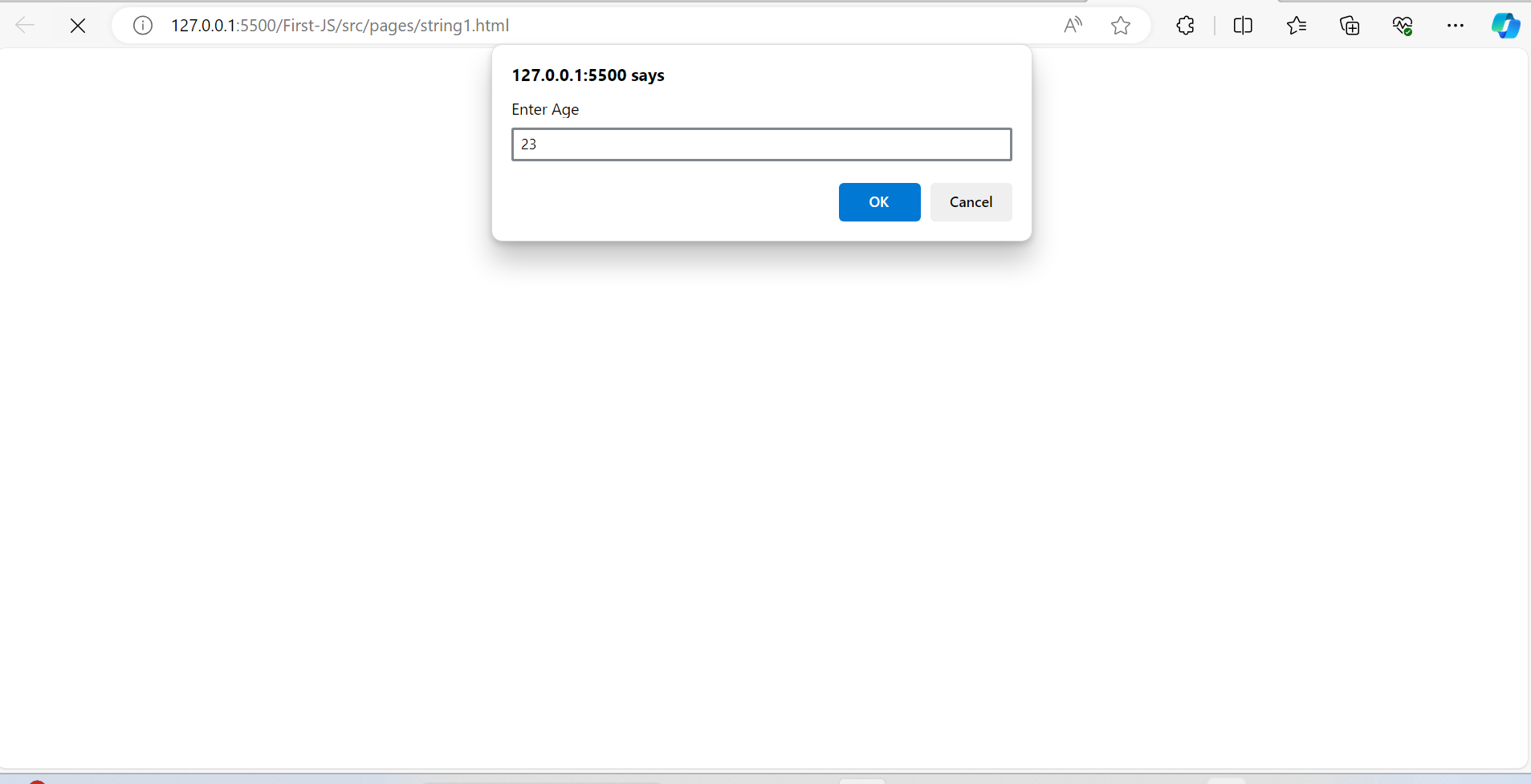
var msg2 = `Hello! ${username}, you are ${age} years old.<br>`;

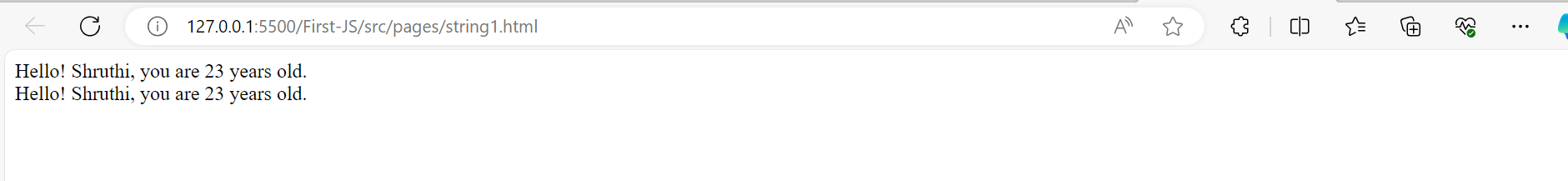
document.write(msg1);

document.write(msg2);

</script>







* Backtick allows a string which can be embed expression
* Example 2:

<script>

var username = prompt("Enter Username");

var inputField = prompt("Enter Type","Email|Password")

var login =

`

<h2>Login Info</h2>

<dl>

Username: <dt>${username}</dt>

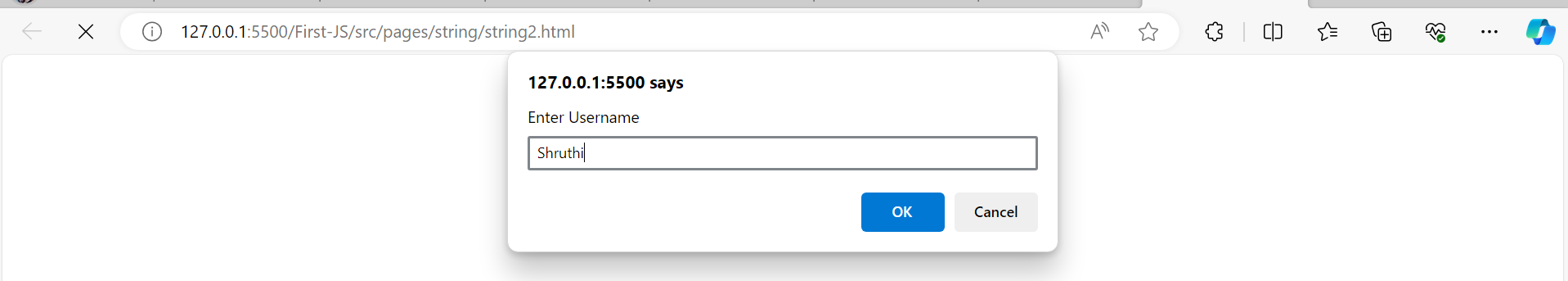
<dt><input type=${inputField}></dt>

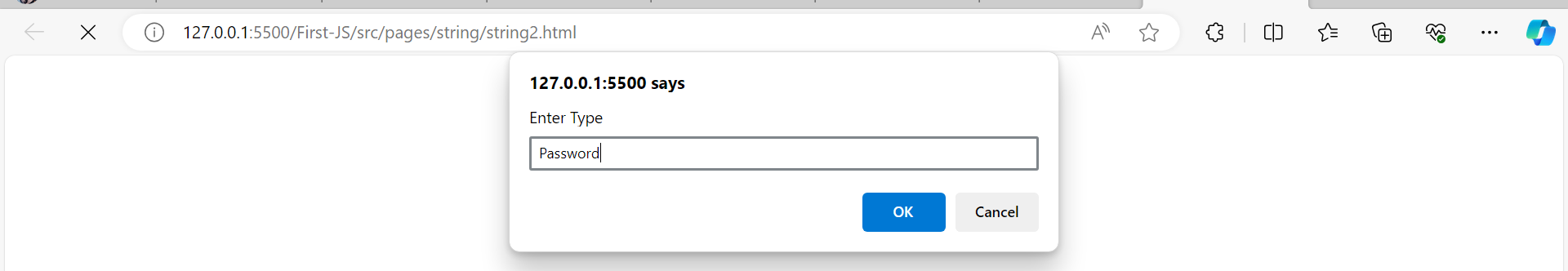
</dl>

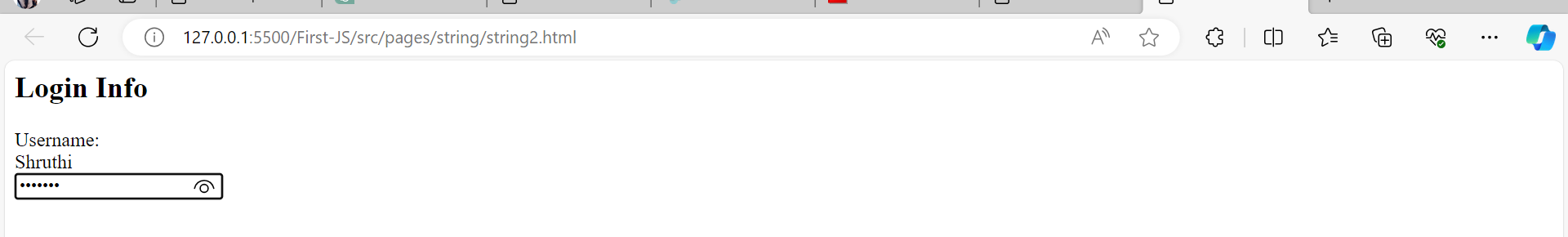
`;

document.write(login);

</script>







* Example 3: Print error message on submit

Step1:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>String</title>

</head>

<body>

<dl>

<dt>User Name</dt>

<dd><input type="text" id="userName"></dd>

<dd id="userError"></dd>

</dl>

<button onclick="RegisterClick()">Register</button>

</body>

</html>

Step2:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>String</title>

<script type="text/javascript">

function RegisterClick(){

var userName = document.getElementById("userName").value;

var userError = document.getElementById("userError");

if(userName == ""){

userError.innerHTML = "Required\*";

}

else{

document.write("Registered Successfully!")

}

}

</script>

</head>

<body>

<dl>

<dt>User Name</dt>

<dd><input type="text" id="userName"></dd>

<dd id="userError"></dd>

</dl>

<button onclick="RegisterClick()">Register</button>

</body>

</html>

Step3: Add css

<script type="text/javascript">

function RegisterClick(){

var userName = document.getElementById("userName").value;

var userError = document.getElementById("userError");

if(userName == ""){

userError.innerHTML = `<font color="red">Required\*</font>`;

}

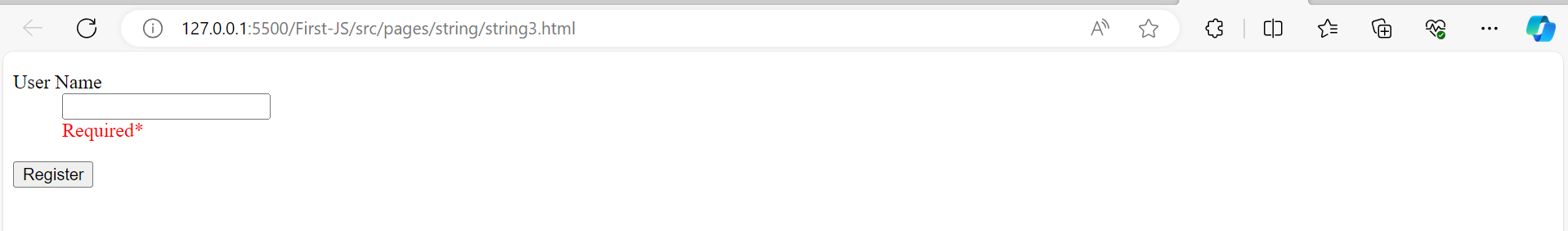
else{

document.write("Registered Successfully!")

}

}

</script>

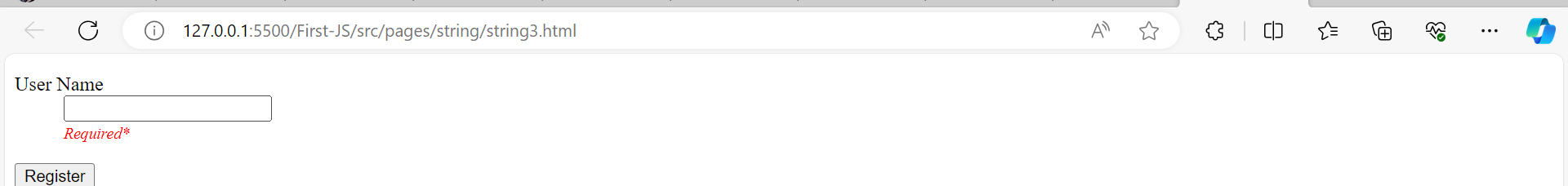


* We have string formatting functions

bold(), italics(), fontcolr(), fontsize(), sup(), sub(), toUpperCase(),etc..

Example

userError.innerHTML = "Required\*".fontcolor('red').fontsize(2).italics();



* These string formatting functions must be used on “non-RC” type.
* Javascript allows you to format a string using “style” and “class”.

element.style.attributeName = “value”

* Note: style attributes are written in camel case.

background-color --> backgroundColor

text-align textAlign

margin-left marginLeft

* Styles are not directly applied to string, they are defined to element that handles string.

Example:

userError.innerHTML = "Required\*";

userError.style.color = 'red';

userError.style.fontSize = '2';

* Javascript allows to configure formats for elements using “css class”.
* CSS class is applied by using “className” property;

Example:

<style>

.error-style{

color: red;

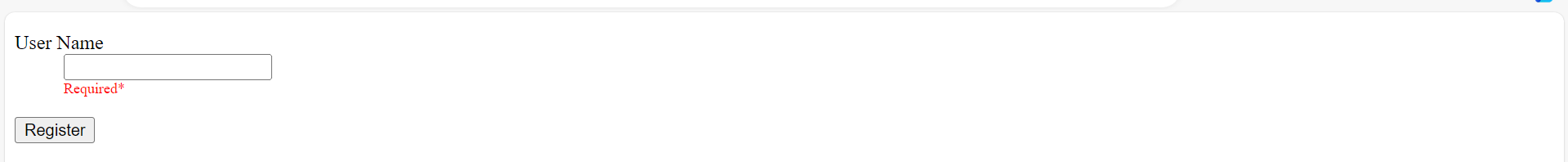
font-size: 12px;

}

</style>

userError.innerHTML = "Required\*";

userError.className = 'error-style';



* **String Manipulation & Methods**

1. Length

Returns number of chars in a variable.

Example:

<script type="text/javascript">

function VerifyName(){

var userName = document.getElementById("UserName").value;

var userError = document.getElementById("UserError");

if(userName.length < 4){

userError.innerHTML = "Minimun 4 chars Required\*".fontcolor('red').fontsize(1);

}

else if(userName.length > 10){

userError.innerHTML = "Maximum 10 char Required\*".fontcolor('red').fontsize(1);

}

else{

userError.innerHTML ="";

}

if(userName == ""){

userError.innerHTML = "Required\*\*".fontcolor('red').fontsize(1);

}

}

function SubmitClick(){

var userName = document.getElementById("UserName").value;

var userError = document.getElementById("UserError");

if(userName == ""){

userError.innerHTML = "Required\*\*".fontcolor('red').fontsize(1);

}else{

document.write("Registered Successfully");

}

}

</script>

<body>

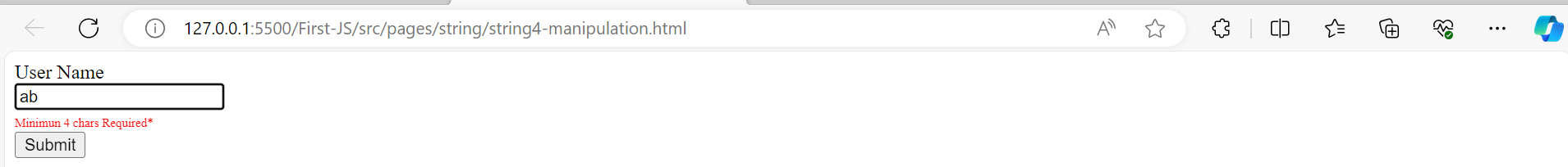
<div>User Name</div>

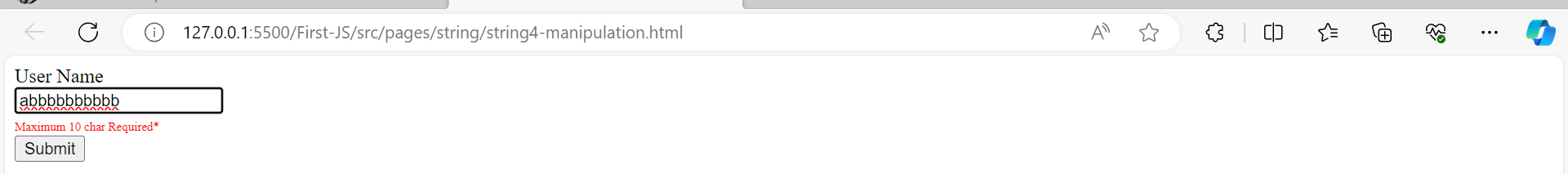
<div><input type="text" id="UserName" onkeyup="VerifyName()"></div>

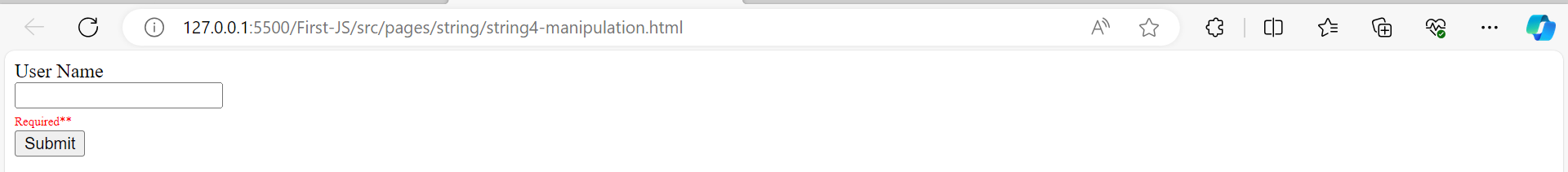
<div id="UserError"></div>

<button onclick="SubmitClick()">Submit</button>

</body>







### **String Methods:**

* **charAt(index):**
  + Returns the character at the specified index.
* const str = "Hello";  
  console.log(str.charAt(1)); // Output: e
* **concat(str1, str2, ...):**
  + Concatenates two or more strings.
* const str1 = "Hello";  
  const str2 = "World";  
  console.log(str1.concat(", ", str2)); // Output: Hello, World
* **indexOf(searchString, position):**
  + Returns the index of the first occurrence of a specified substring.
* const str = "Hello, World!";  
  console.log(str.indexOf("World")); // Output: 7
* **substring(startIndex, endIndex):**
  + Returns a subset of a string based on start and end index.
* const str = "Hello, World!";  
  console.log(str.substring(0, 5)); // Output: Hello
* **toUpperCase() / toLowerCase():**
  + Converts a string to uppercase or lowercase.
* const str = "Hello, World!";  
  console.log(str.toUpperCase()); // Output: HELLO, WORLD!  
  console.log(str.toLowerCase()); // Output: hello, world!
* **trim():**
  + Removes whitespace from both ends of a string.
* const str = " Hello, World! ";  
  console.log(str.trim()); // Output: Hello, World!
* **replace(searchValue, replaceValue):**
  + Replaces a specified value with another value in a string.
* javascriptCopy code
* const str = "Hello, World!";  
  console.log(str.replace("World", "Universe"));
* // Output: Hello, Universe!
* **split(separator):**
  + Splits a string into an array of substrings based on a specified separator.
* const str = "apple,orange,banana";  
  const fruits = str.split(",");  
  console.log(fruits); // Output: ["apple", "orange", "banana"]
* **startsWith(prefix):**
  + Checks if a string starts with a specified prefix.
* const str = "Hello, World!";  
  console.log(str.startsWith("Hello")); // Output: true
* **endsWith(suffix):**
* Checks if a string ends with a specified suffix.

const str = "Hello, World!";  
console.log(str.endsWith("World!")); // Output: true

### **Accessing Characters by Index:**

* **charAt(index):**
  + Returns the character at the specified index.
* const str = "Hello";  
  console.log(str.charAt(1)); // Output: e
* **charCodeAt(index):**
  + Returns the Unicode value of the character at the specified index.
* const str = "Hello";  
  console.log(str.charCodeAt(1)); // Output: 101 (Unicode value of 'e')
* **substring(startIndex, endIndex):**
  + Returns a subset of a string based on start and end index.
* const str = "Hello, World!";  
  console.log(str.substring(0, 5)); // Output: Hello
* **slice(startIndex, endIndex):**
  + Extracts a section of a string and returns it as a new string.
* const str = "Hello, World!";  
  console.log(str.slice(0, 5)); // Output: Hello
* **substr(startIndex, length):**
  + Returns the characters in a string beginning at the specified location through the specified number of characters.
* const str = "Hello, World!";  
  console.log(str.substr(7, 5)); // Output: World

### **Searching by Index:**

* **indexOf(searchString, position):**
  + Returns the index of the first occurrence of a specified substring.
* const str = "Hello, World!";  
  console.log(str.indexOf("World")); // Output: 7
* **lastIndexOf(searchString, position):**
  + Returns the index of the last occurrence of a specified substring.
* const str = "Hello, World!";  
  console.log(str.lastIndexOf("l")); // Output: 10

### **Modifying by Index:**

* **replace(searchValue, replaceValue):**
  + Replaces a specified value with another value in a string.
* const str = "Hello, World!";  
  console.log(str.replace("World", "Universe")); // Output: Hello, Universe!

### **Padding by Index:**

* **padStart(targetLength, padString):**
  + Pads the current string with another string until the resulting string reaches the given length.
* const str = "42";  
  console.log(str.padStart(5, "0")); // Output: "00042"
* **padEnd(targetLength, padString):**
  + Pads the current string with another string until the resulting string reaches the given length. Similar to **padStart** but pads at the end.
* const str = "42";  
  console.log(str.padEnd(5, "0")); // Output: "42000"
* **includes(searchString, position):**
  + Checks if a string contains a specified substring.
* const str = "Hello, World!";  
  console.log(str.includes("World")); // Output: true
* **match(regexp):**
  + Searches a string for a match against a regular expression and returns the matches as an array.
* const str = "The quick brown fox";  
  const matches = str.match(/o/g);  
  console.log(matches); // Output: ["o", "o"]
* **search(regexp):**
  + Searches a string for a specified value or regular expression. Returns the index of the first match.
* const str = "The quick brown fox";  
  console.log(str.search(/brown/)); // Output: 10

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Mobile Pattern Match</title>

<link rel="stylesheet" href="../../../node\_modules/bootstrap/dist/css/bootstrap.css">

<script>

var regExp = / /;

function OnCountryChange(){

var countryName = document.getElementById("lstCountries").value;

var flagImg = document.getElementById("flag");

var mobileNo = document.getElementById("txtMobile");

if(countryName == "INDIA"){

flagImg.src ="../../../src/images/India.jpg";

mobileNo.placeholder = "Eg: +91XXXXXXXXXX";

regExp = /\+91\d{10}/;

} else if(countryName == "US"){

flagImg.src ="../../../src/images/Usa.jpg";

mobileNo.placeholder = "Eg: +(1)-(213)-XXX-XXXX";

regExp = /\+\(1\)\(\d{3}\)\s\d{3}-\d{4}/;

} else if(countryName == "UK"){

flagImg.src ="../../../src/images/UK.jpg";

mobileNo.placeholder = "Eg: +(44)(XX) XXXX XXXX";

regExp = /\+\(44\)\(\d{2}\)\s\d{4}\s\d{4}/;

}

}

function VerifyMobile(){

console.log("Verifying Mobile")

var mobile = document.getElementById("txtMobile").value;

var mobileError = document.getElementById("mobileError");

if(mobile.match(regExp)){

console.log("Matched")

mobileError.innerHTML = "";

}else{

console.log("Not Matched")

mobileError.innerHTML = "Invalid Mobile Number".fontcolor('red');

}

}

</script>

</head>

<body class="container-fluid">

<h2>Verify Mobile</h2>

<dl class="w-25">

<dt>Your Country</dt>

<dd class="input-group">

<select id="lstCountries" class="form-select" onchange="OnCountryChange()">

<option>INDIA</option>

<option>US</option>

<option>UK</option>

</select>

<img width="80" height="40" class="input-group-text" id="flag" alt="" src ="../../../src/images/India.jpg">

</dd>

<dt>Your Mobile</dt>

<dd>

<input type="text" class="form-control" id="txtMobile" placeholder = "Eg: +91XXXXXXXXXX" onblur="VerifyMobile()">

</dd>

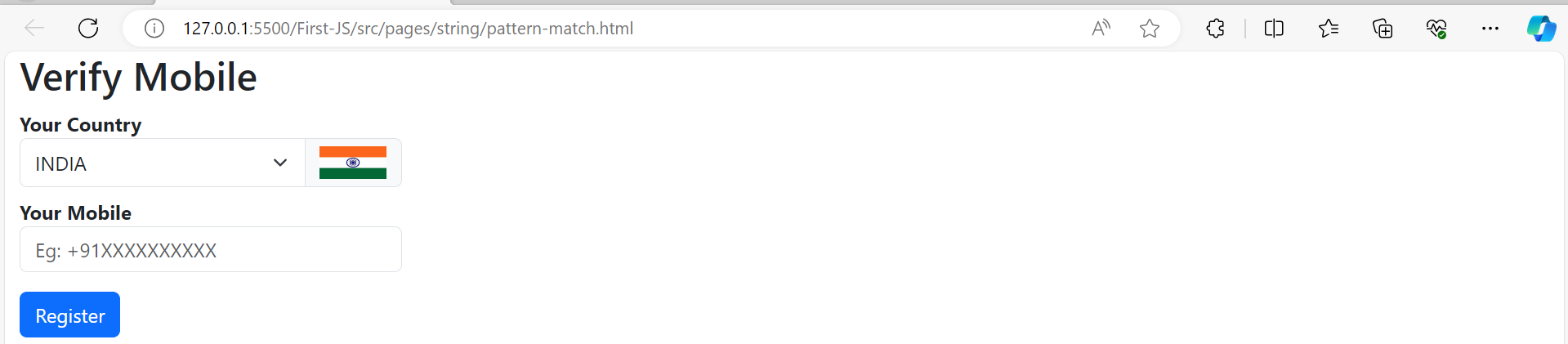
<dd id="mobileError"></dd>

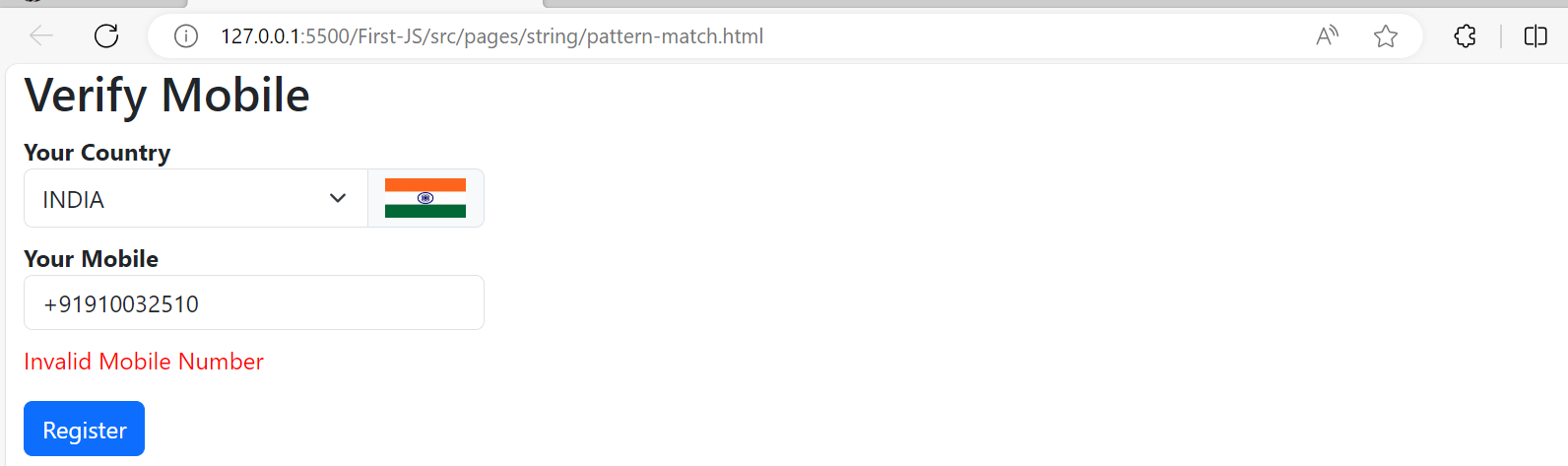
</dl>

<button class="btn btn-primary">Register</button>

</body>

</html>





1. **Boolean:**

* Boolean types are used indecision Making.(true/false)
* The Boolean data type represents a logical entity and can have one of two values: true or false.
* Booleans are often used in conditional statements and logical expressions to control the flow of a program.

Here are some key points about the Boolean data type in JavaScript:

* **Values:** There are only two possible Boolean values: **true** and **false**.

let isTrue = true;  
let isFalse = false;

* **Boolean Expressions:** Boolean values are often the result of a comparison or logical operation. For example:
* let x = 5;  
  let y = 10;  
    
  let isGreaterThan = x > y; // Evaluates to false  
  let isEquals = x === y; // Evaluates to false
* **Logical Operators:** JavaScript provides logical operators that can be used to create complex Boolean expressions. The logical operators include **&&** (logical AND), **||** (logical OR), and **!** (logical NOT).
* let isTrue1 = true && true; // Evaluates to true  
  let isTrue2 = true || false; // Evaluates to true  
  let isFalse1 = !true; // Evaluates to false
* **Truthy and Falsy Values:** In JavaScript, values are not strictly limited to **true** or **false** when it comes to Boolean evaluations. Some values are considered "truthy," meaning they are treated as **true** in a Boolean context, while others are "falsy," meaning they are treated as **false**. For example, the number **0**, an empty string **""**, **null**, **undefined**, and **NaN** are falsy values.
* let truthyValue = "Hello"; // This is truthy  
  let falsyValue = ""; // This is falsy
* **Boolean Objects:** In addition to the primitive Boolean type, there is also a Boolean object. However, it is rarely used and generally not recommended for simple boolean operations.
* let boolObj = new Boolean(true);  
  let boolValue = boolObj.valueOf(); // (true)

**Boolean Type Coercion:**

Type coercion is the automatic conversion of values from one data type to another.

In JavaScript, type coercion can happen in various situations, especially when using operators like **+** with different types. Let's go through examples with boolean values and type coercion:

* **Boolean with Number Type Coercion:**
* let result = true + 5;  
  console.log(result); //6
* In this example, the boolean value **true** is implicitly converted to the number **1**, and then it's added to **5**. So, the result is **1 + 5**, which equals **6**.
* **Boolean with String Type Coercion:**
* let result = true + "A";  
  console.log(result); //trueA
* Here, the boolean **true** is converted to the string **"true"**, and then it's concatenated with the string **"A"**. The result is the string **"trueA"**.
* **Boolean with Implicit String to Number Coercion:**
* let result = true + "5";  
  console.log(result); //true5
* In this case, the boolean **true** is converted to string true, and then it's added to the string **"5"**. The result is the string **"true5"**.
* **Boolean with Explicit String to Number Coercion:**
* let result = true + Number("2");  
  console.log(result); //3
* The boolean **true** is converted to the number **1**, and then it's added to the explicitly converted number **2**. The result is **1 + 2**, which equals **3**.
* **Boolean with Implicit String to Number Coercion and Addition:**
* let result = true + "A" + 5;  
  console.log(result); //trueA5
* Here, the boolean **true** is converted to the string **"true"**, then concatenated with the string **"A"**, and finally, the result is concatenated with the number **5**. The result is the string **"trueA5"**.

Example:

<script>

document.write(`<h2>Boolean with Number Type: <br></h2>`);

let ex1 = true+5;

document.write(`true + 5 = <b> ${ex1} </b><br>`);

let ex2 = true + true;

document.write(`true + true = <b> ${ex2} </b><br>`);

document.write(`<h2>Boolean with String Type: <br></h2>`);

let ex3 = true+"A";

document.write(`true + "2" = <b> ${ex3} </b><br>`);

document.write(`<h2>Boolean with Implicit String to Number: <br></h2>`);

let ex4 = true+"5";

document.write(`true + "5" = <b> ${ex4} </b><br>`);

document.write(`<h2>Boolean with Explicit String to Number: <br></h2>`);

let ex5 = true+Number("2");

document.write(`true + Number("5") = <b> ${ex5} </b><br>`);

document.write(`<h2>Boolean with Mixed Addition: <br></h2>`);

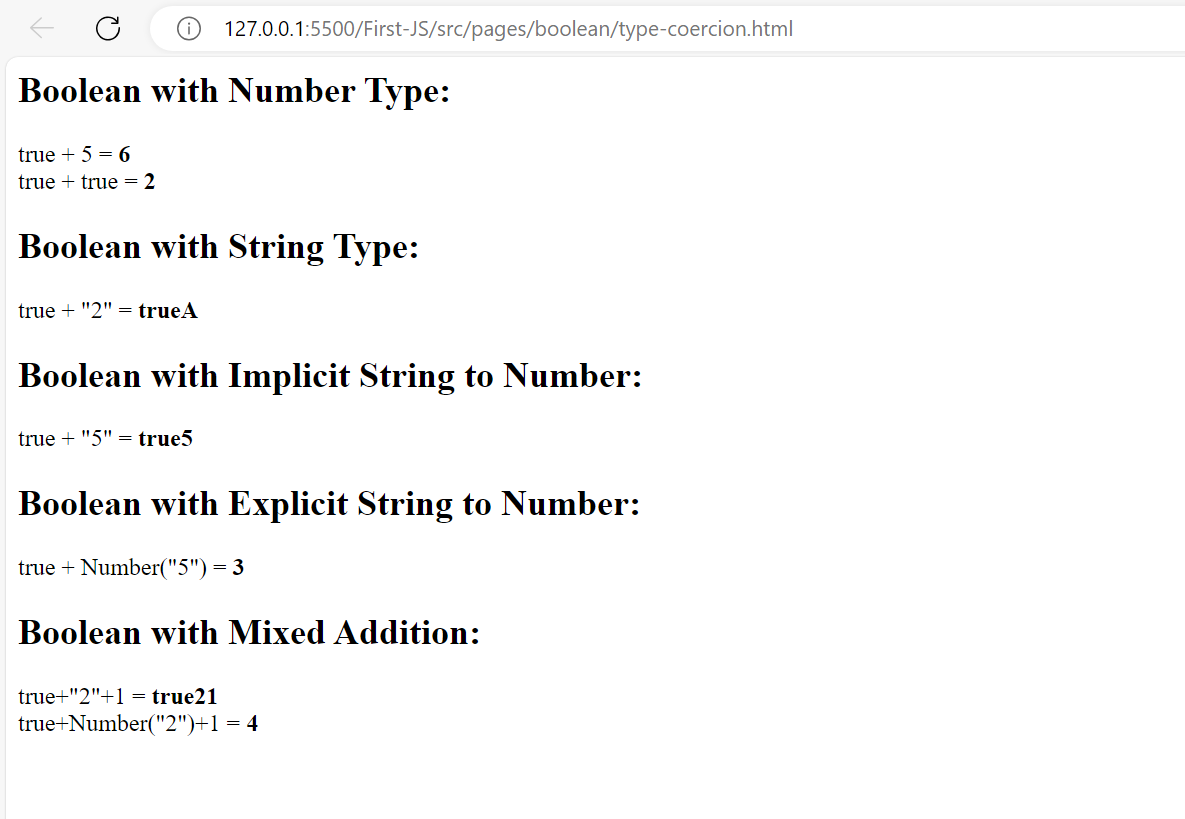
let ex6 = true+"2"+1;

document.write(`true+"2"+1 = <b> ${ex6} </b><br>`);

let ex7 = true+Number("2")+1;

document.write(`true+Number("2")+1 = <b> ${ex7} </b><br>`);

</script>

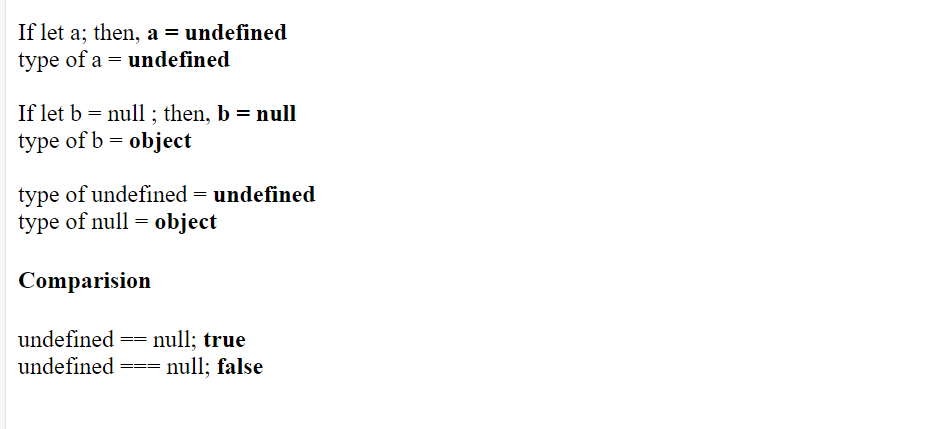


1. **Undefined**

"undefined" is a primitive value that represents the absence of a value or the uninitialized state of a variable. It is also a type in itself, and variables that have been declared but not assigned a value are automatically assigned the value **undefined**.

* **Undefined Value:**
  + When a variable is declared but not assigned a value, it is automatically assigned the value **undefined**.
  + let x;  
    console.log(x); // Outputs: undefined
* **Type and Value:**
  + The **typeof** operator can be used to check the type of a variable.
  + let y;  
    console.log(typeof y); // Outputs: "undefined"
* **Undefined as a Type:**
  + **undefined** is also a type in JavaScript.
  + console.log(typeof undefined); // Outputs: "undefined"
* **Function Parameters:**
  + If a function is called with fewer arguments than declared parameters, the missing parameters are automatically assigned the value **undefined**.
  + function example(a, b, c) {  
     console.log(a, b, c);  
    }  
      
    example(1, 2); // Outputs: 1 2 undefined
* **Checking for Undefined:**
  + You can explicitly check for **undefined** using the equality operator (**==**) or strict equality operator (**===**).
  + let z;  
    console.log(z === undefined); // Outputs: true

**Difference between undefined & null**



1. **Symbol:**

the **Symbol** data type is a primitive data type introduced in ECMAScript 6 (ES6). Symbols are unique and immutable values that can be used as keys for object properties. Unlike strings or numbers, symbols are guaranteed to be unique, which means that no two symbols will ever be equal, even if they have the same name.

Here are some key characteristics of the **Symbol** data type:

* **Uniqueness:** Every time you create a symbol, it is guaranteed to be unique. This means that even if you create multiple symbols with the same description (name), they are distinct entities.
* const symbol1 = Symbol('mySymbol');  
  const symbol2 = Symbol('mySymbol');  
    
  console.log(symbol1 === symbol2); // false
* **Immutability:** Once a symbol is created, its value cannot be changed. Symbols are immutable, meaning their state cannot be modified after creation.
* const symbol = Symbol('mySymbol');  
  symbol.description = 'newDescription'; // This has no effect; symbols are immutable
* **Creating Symbols:**
* Symbols can be created using the **Symbol()** constructor, with an optional description (a string) as its parameter. The description is useful for debugging but does not affect the uniqueness of the symbol.
* const symbol = Symbol('mySymbol');
* **Use as Object Keys:**
* Symbols are commonly used as keys for object properties. Because they are unique, using symbols as keys helps avoid naming conflicts in objects.
* const myObj = {};  
    
  const symbol = Symbol('mySymbol');  
    
  myObj[symbol] = 'Hello, Symbol!';  
    
  console.log(myObj[symbol]); // Hello, Symbol!
* **Built-in Symbols:**
* JavaScript also provides a set of built-in symbols, which are used as property keys for certain predefined behaviors. Examples include **Symbol.iterator** for defining iterable objects and **Symbol.toStringTag** for customizing the **toString** behavior of an object.
* const arr = [1, 2, 3];  
  const iterator = arr[Symbol.iterator]();  
    
  console.log(iterator.next()); // { value: 1, done: false }

## Arrays:

An array is a data structure that allows you to store and organize multiple values within a single variable.

Arrays provide a contiguous block of memory to store data in sequential order.

Allows quick & direct access to an element using index.

Reduces overhead – minimal additional memory required compared to other data structures.

Arrays refers to a formation where items are in order but can be accessed randomly.

Arrays size can dynamically change in javascript.

Can handle different data types in an array.

### **Array Creation:**

**Declaration:**

var products;

let products;

**Initialize:**

var products = [];

var products = new Array();

var products = [“A”,1,true];

Var products = new Array(“A”,1,true);

**Assigning** using properties:

|  |  |  |  |
| --- | --- | --- | --- |
| array | 10 | 20 | 50 |
| Index | 0 | 1 | 2 |
| property | “0” | “1” | “2” |

Here property are implicity strings:

Example;

<script>

var values = [19,"John",true];

for(var property in values){

document.write(`[${property}]: ${values[property]}<br>`)

}

</script>



Check type of property:

<script>

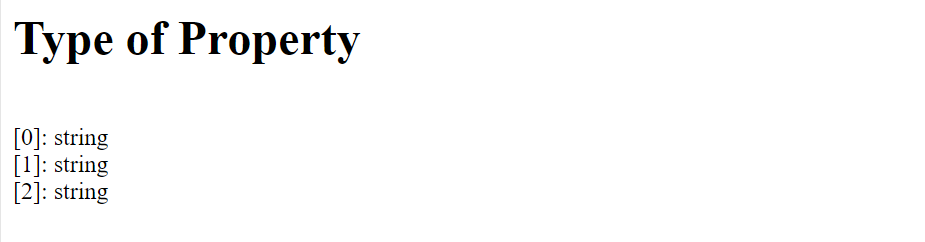
document.write("<h1>Type of Property</h1><br>");

for(var property in values){

document.write(`[${property}]: ${typeof property}<br>`)

}

</script>



<script>

var values = [19,"John",true];

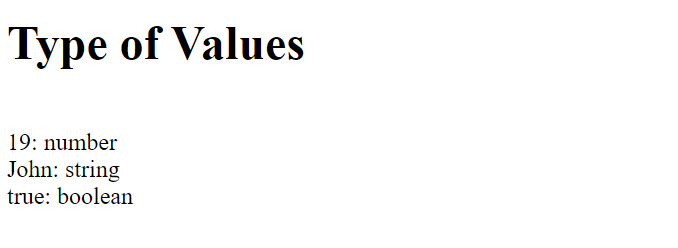
document.write("<h1>Type of Values</h1><br>");

for(var property in values){

document.write(`${values[property]}: ${typeof values[property]}<br>`)

}

</script>



--> Arrays are not refering by index, they are referring to property that have index.

var values = [19,"John",true];

document.write(values[0]); // 19 – internally considers as array

document.write(values["0"]); //19 - still works

**Arrays allows de-structuring:**

<script>

var value = [10,"Shruthi",true, ['I','love','myself'] ,function(){document.write("My Function called")}];

var [id,name,condition,myarray,myfunction] = value;

document.write(myarray[1]) // love

myfunction(); // My Function called

</script>

### **Array Manipulation:**

Javascript array object provides a set of properties and methods to control array.

1. Length: returns total count of elements in an array