JavaScript

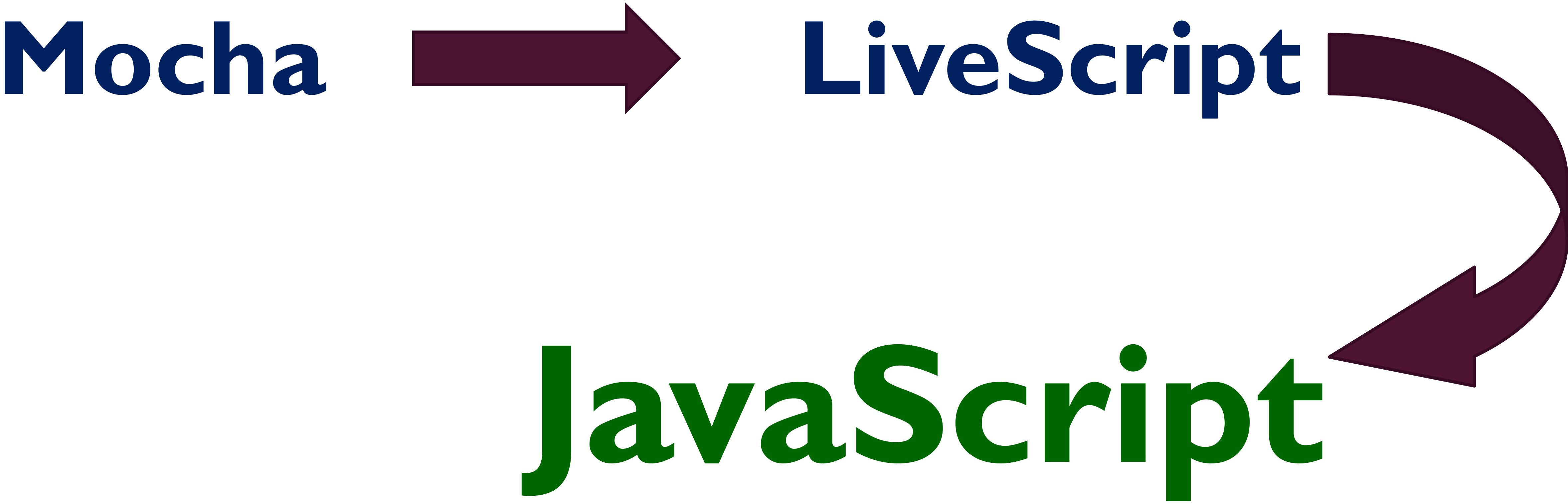
What do you mean by…

- ECMAScript
- JavaScript
- ES5
- ES6
- ES7
- ES2015
- ES2016
- ES2017
- ES2018
- Bable

Popular Web Browser 1995

Netscape Navigator 9

Brendan Eich



Mocha

LiveScript

JavaScript

Internet
Explorer

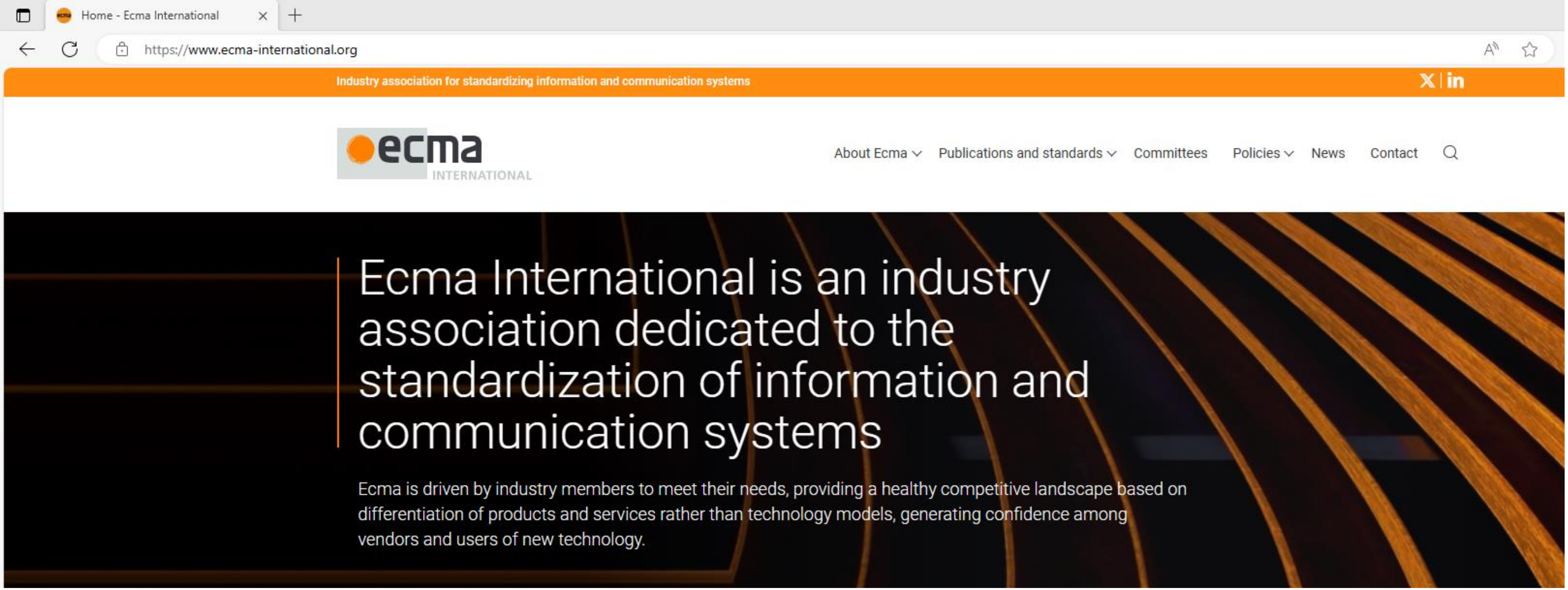
JScript

Year 1996

JavaScript

JScript

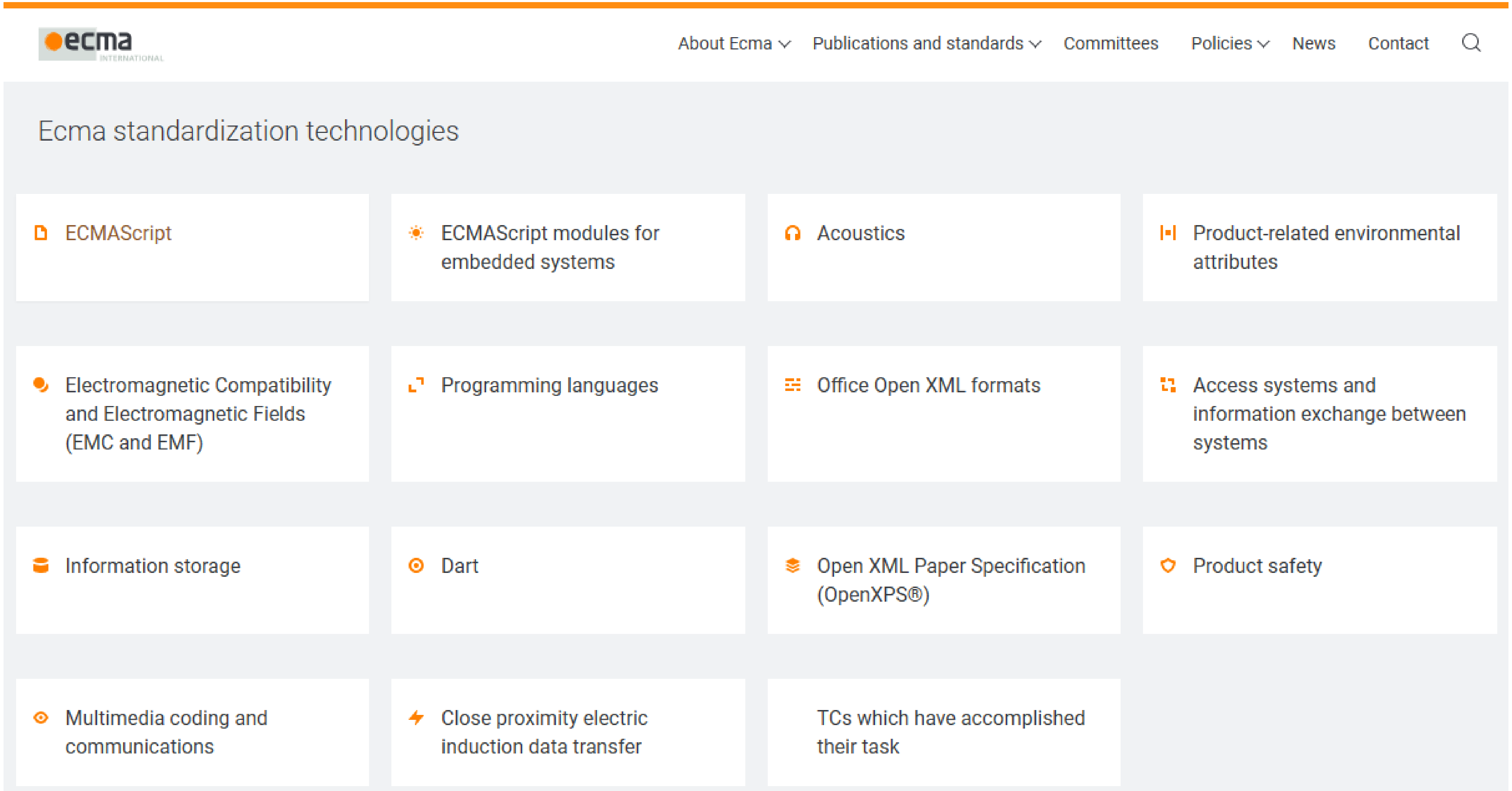
Ecma standardises the different technologies



Home - Ecma International
https://www.ecma-international.org
Industry association for standardizing information and communication systems
ecma INTERNATIONAL
About Ecma
Publications and standards
Committees
Policies
News
Contact

Ecma International is an industry association dedicated to the standardization of information and communication systems

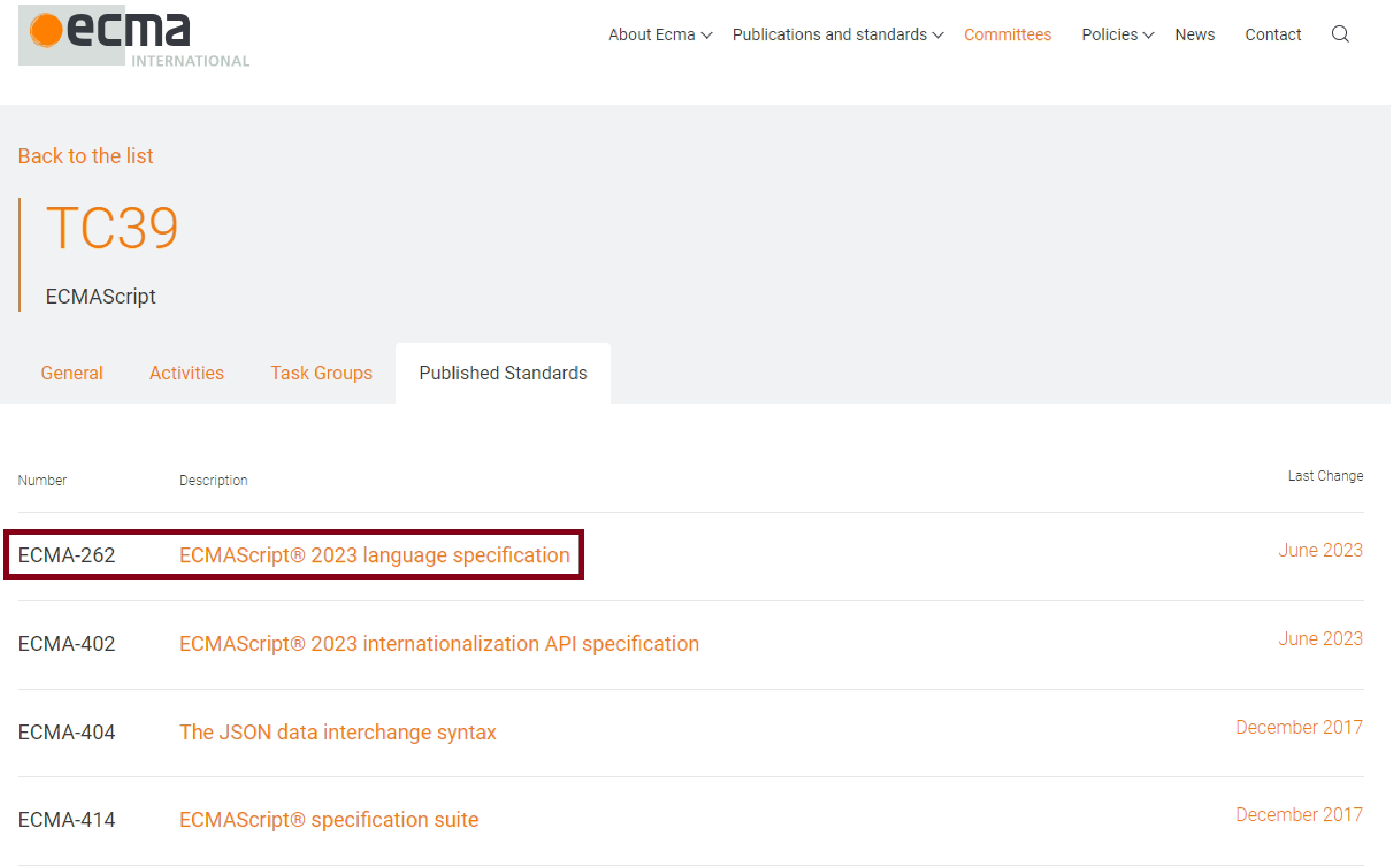
Ecma is driven by industry members to meet their needs, providing a healthy competitive landscape based on differentiation of products and services rather than technology models, generating confidence among vendors and users of new technology.



Ecma standardization technologies

- ECMAScript
- ECMAScript modules for embedded systems
- Acoustics
- Product-related environmental attributes
- Electromagnetic Compatibility and Electromagnetic Fields (EMC and EMF)
- Programming languages
- Office Open XML formats
- Access systems and information exchange between systems
- Information storage
- Dart
- Open XML Paper Specification (OpenXPS®)
- Product safety
- Multimedia coding and communications
- Close proximity electric induction data transfer
- TCs which have accomplished their task

All Scripting Languages works on browser should follow the ECMA-262 Specification



TC39

ECMAScript

Number	Description	Last Change
ECMA-262	ECMAScript® 2023 language specification	June 2023
ECMA-402	ECMAScript® 2023 internationalization API specification	June 2023
ECMA-404	The JSON data interchange syntax	December 2017
ECMA-414	ECMAScript® specification suite	December 2017

Different ECMAScript Editions

- ✓ ES1 : June-1997
- ✓ ES2 : June-1998
- ✓ ES3 : Dec-1999
- ✓ ES4 : June-2003
- ✓ ES5 : Dec-2009, Lot of new features are added

- ✓ ES6/ES2015 : June-2015(Biggest Updates in JS history - Modern)
- ✓ ES7/ES2016: June-2016 (7th Edition)

- ✓ ES2023(ECMAScript 2023): June-2023 (17th Edition)

TC39 Committee

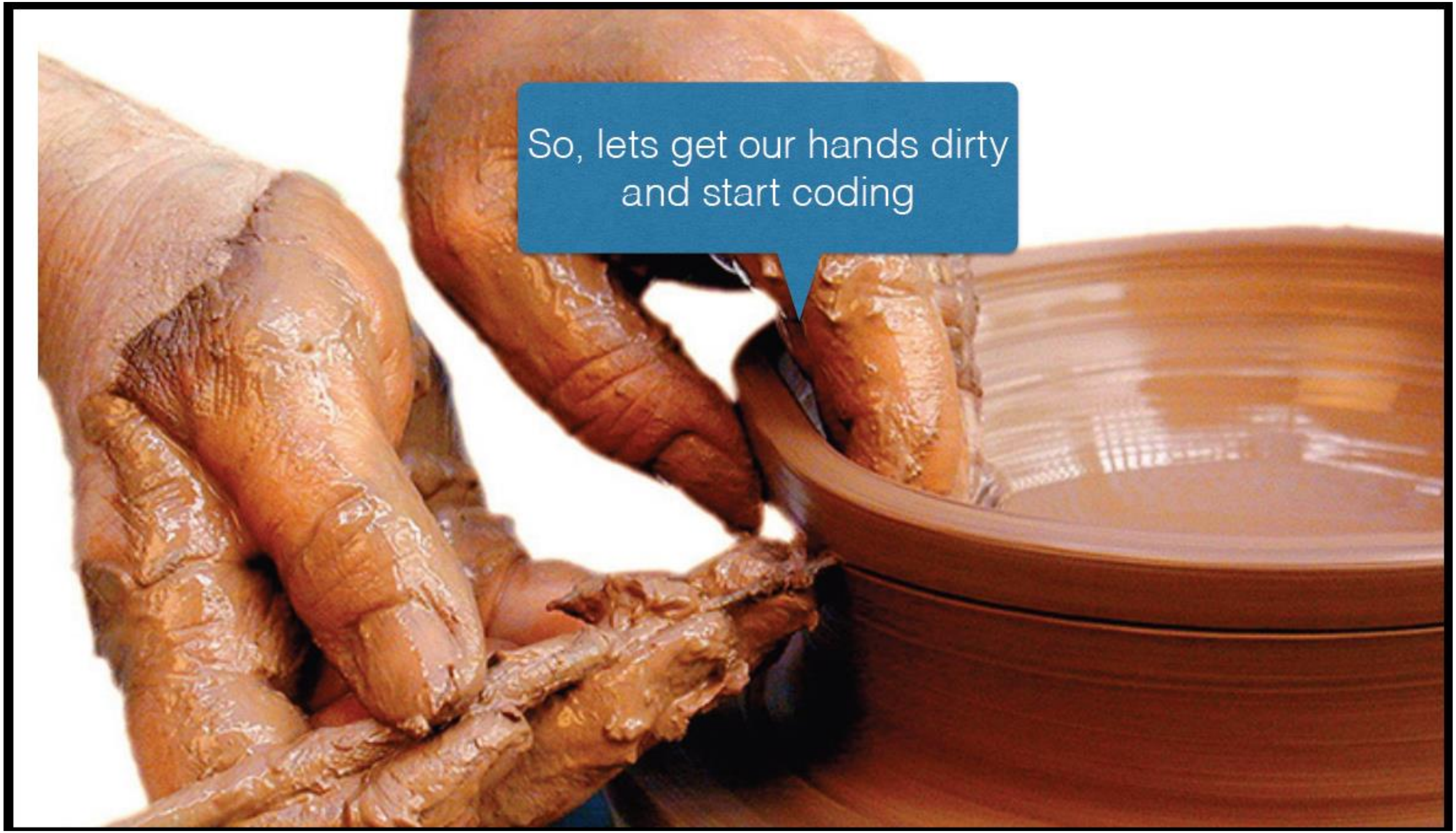
A Year (plus a little) on TC39

Dispatches from the future of JavaScript

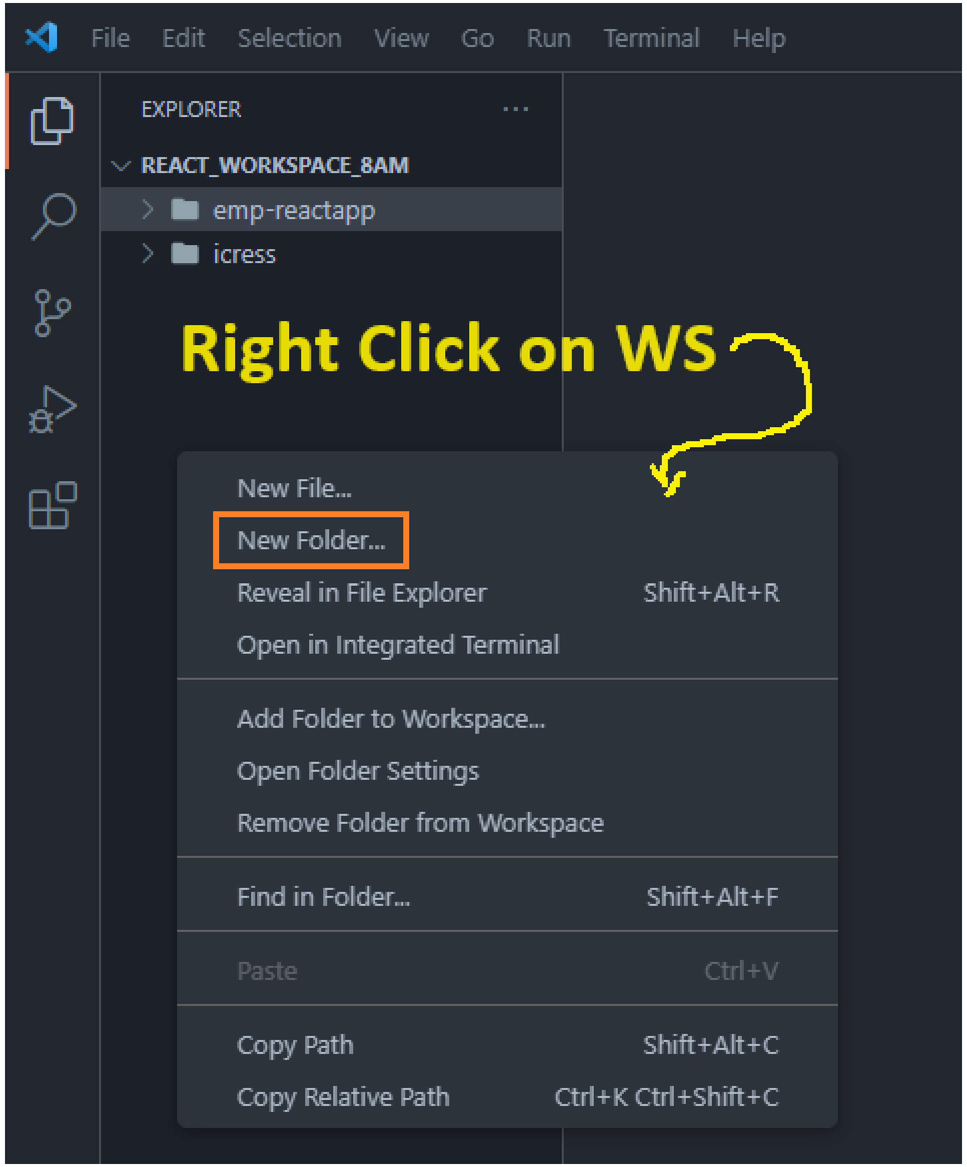
JavaScript supports Backward Compatibility

JavaScript DOES NOT Support forward Compatible

- ✓ i.e., Latest JavaScript Code does not work on VERY OLD BROWSERs
- ✓ For this we have tools like Babel (Transpiles to old)
- ✓ Moto of Babel – Use next generation JavaScript today

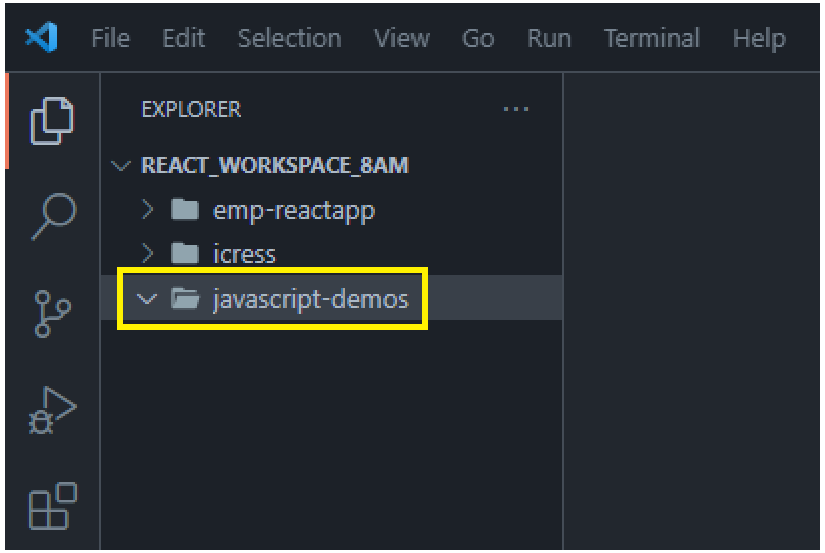


So, lets get our hands dirty and start coding



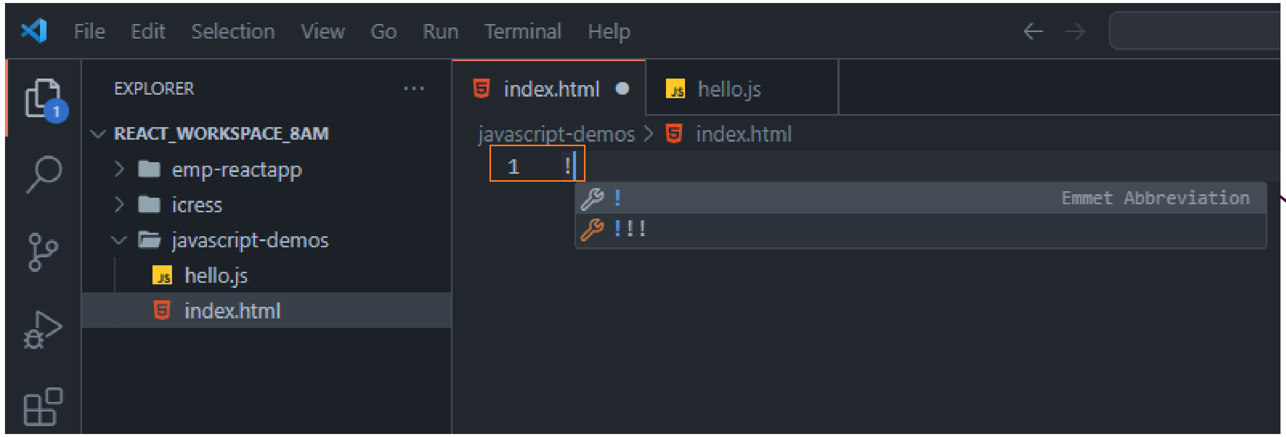
Right Click on WS

File Edit Selection View Go Run Terminal Help
EXPLORER
REACT_WORKSPACE_8AM
emp-reactapp
icress
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C

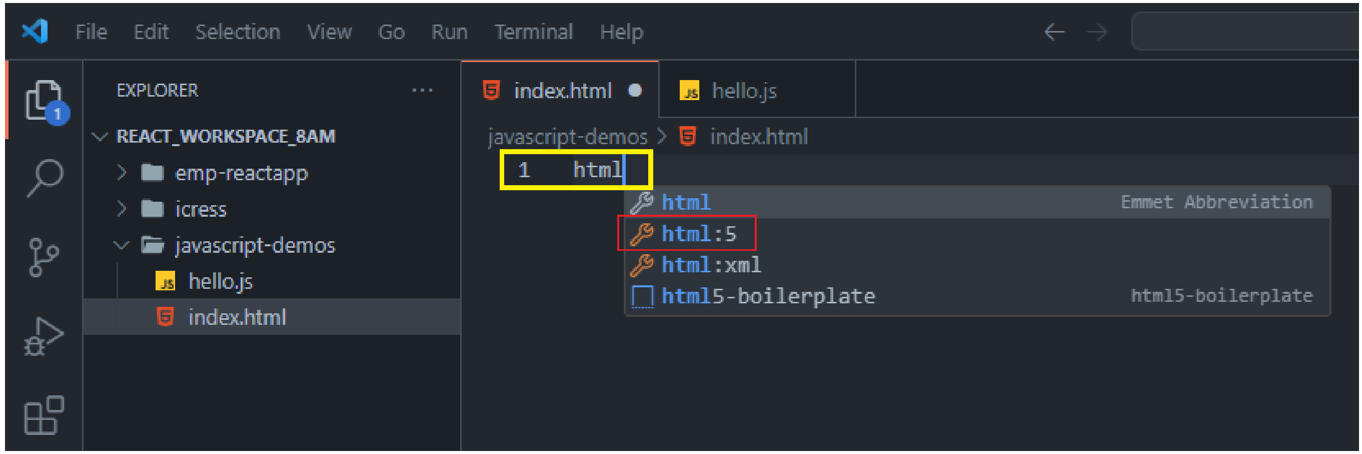


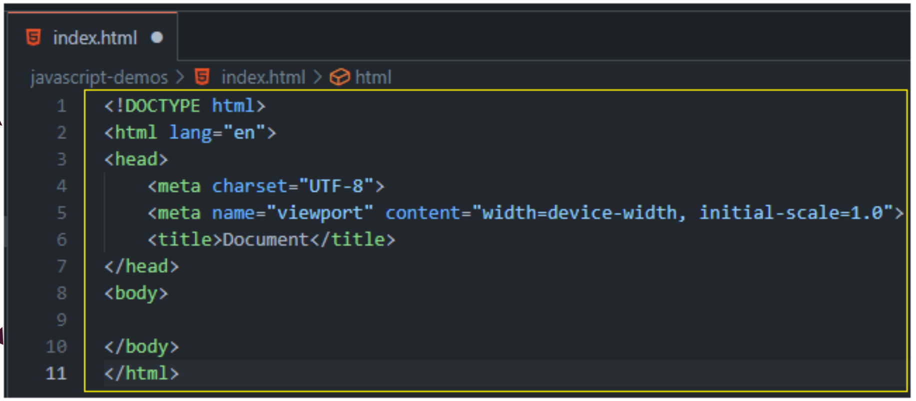
Got to index.html file and add HTML Template code

1. Press ! and Tab (OR)



2. Type html5 and select

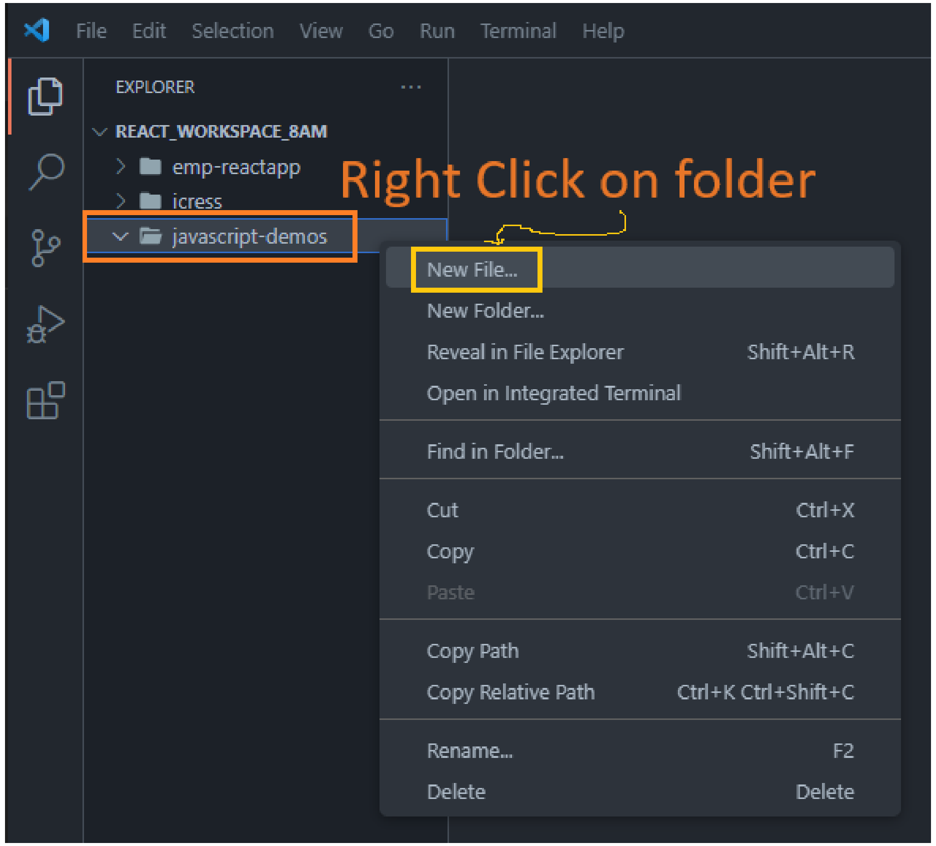




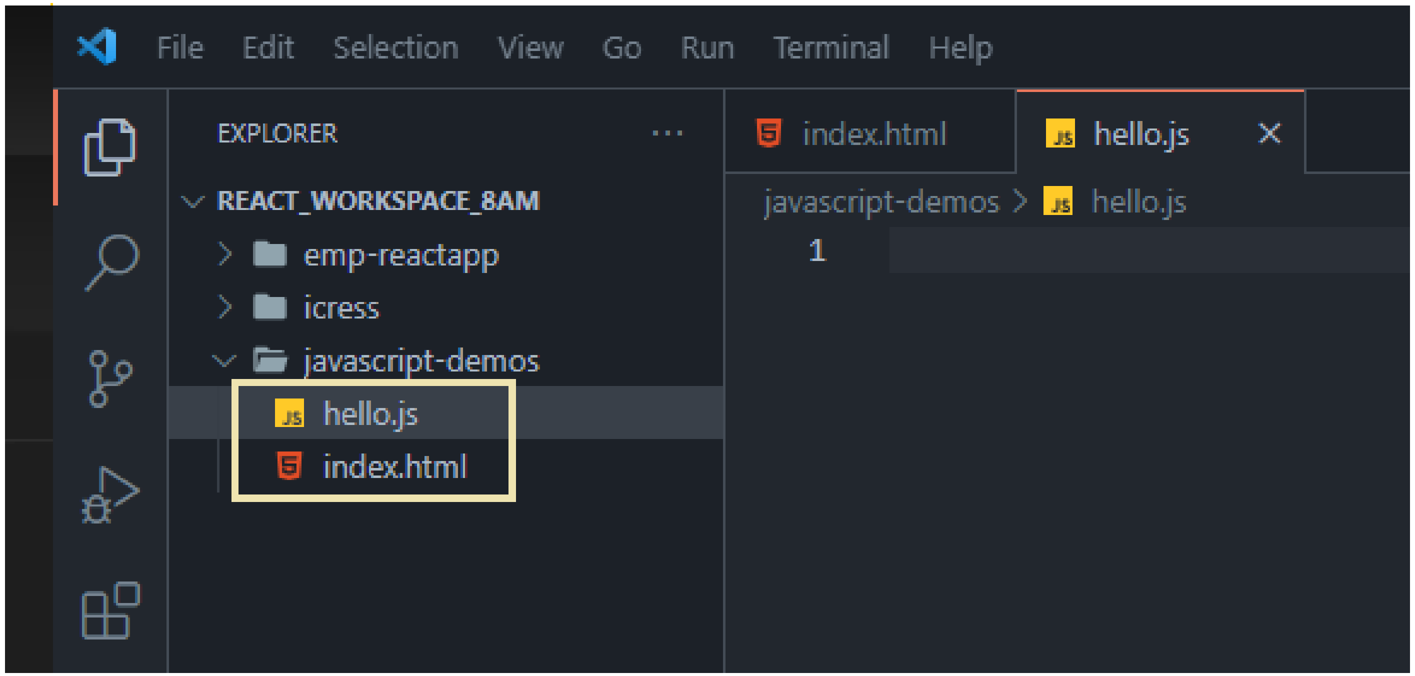
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

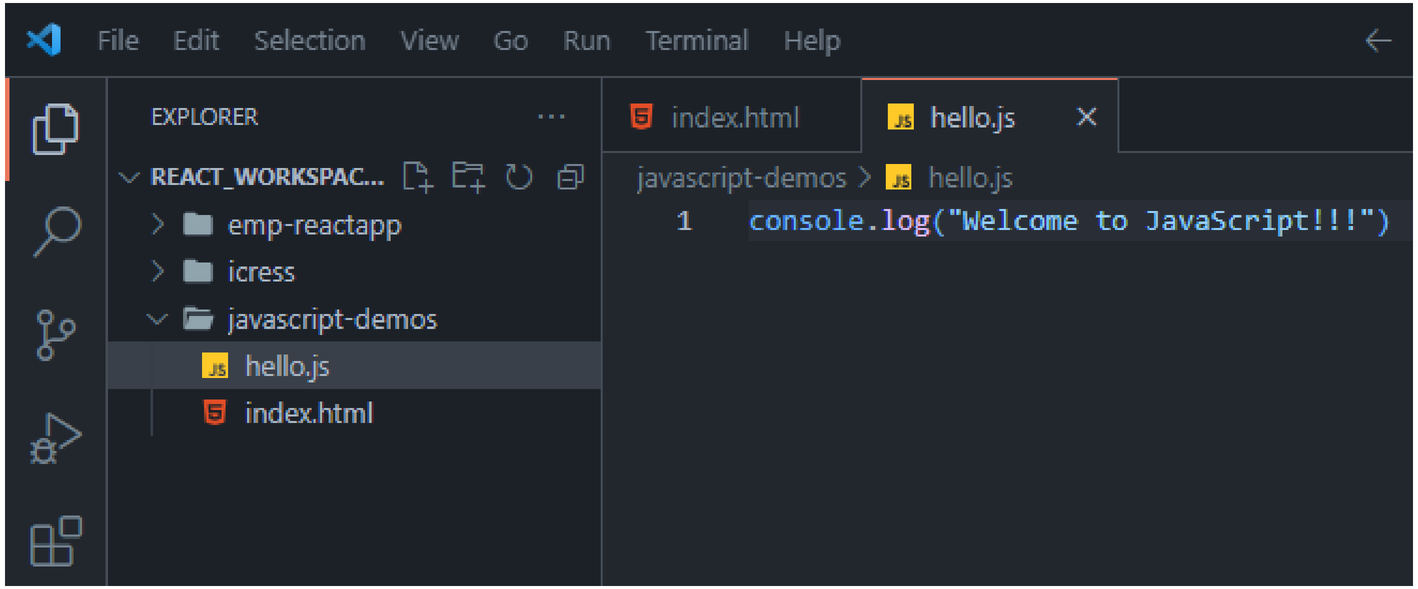
HTML Template

Create JavaScript File & add log statement


File Edit Selection View Go Run Terminal Help
EXPLORER
REACT_WORKSPACE_8AM
emp-reactapp
icress
javascript-demos
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Find in Folder... Shift+Alt+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename... F2
Delete Delete

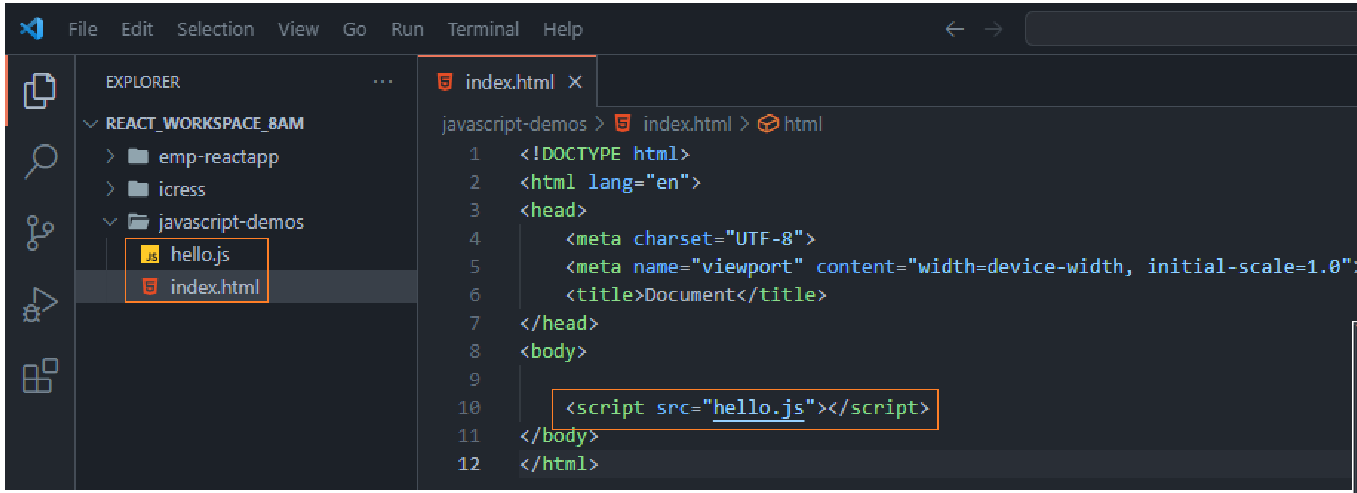
Right Click on folder



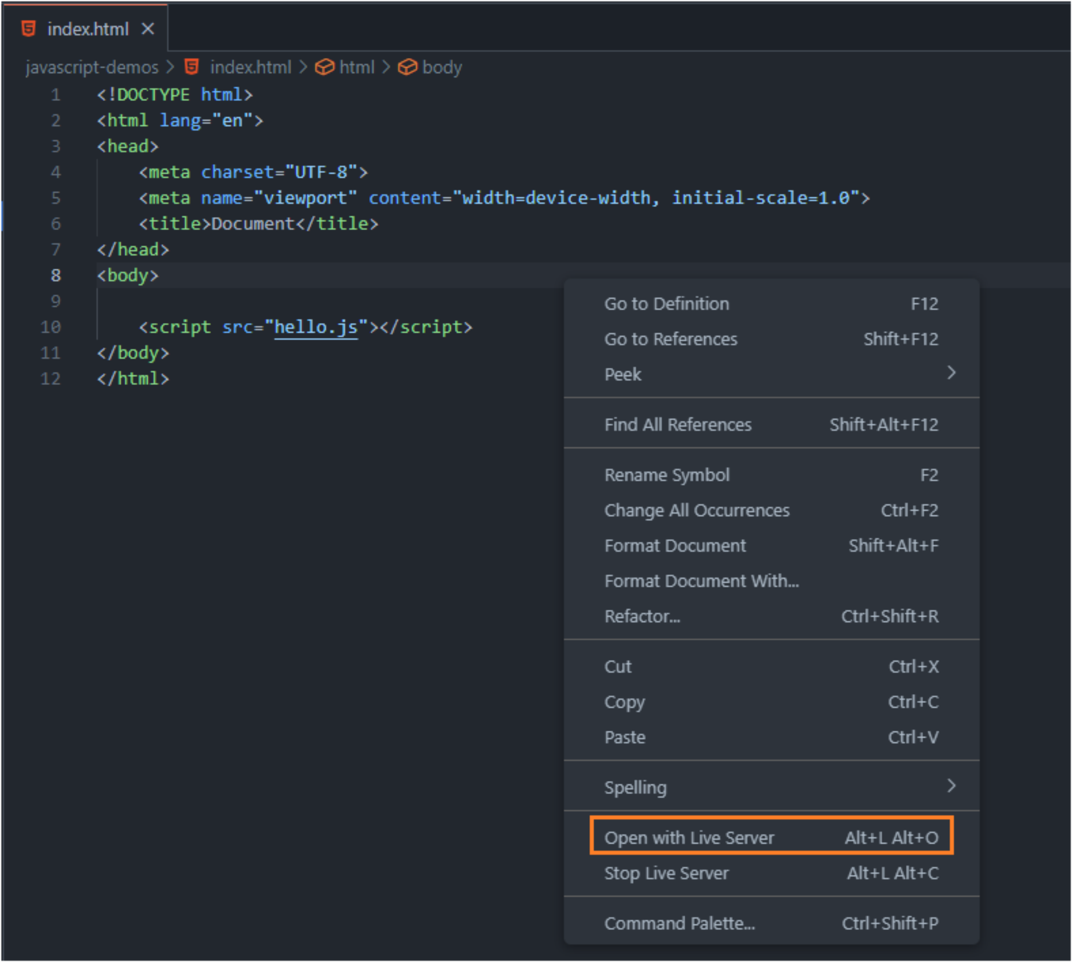


```
console.log("Welcome to JavaScript!!!")
```

Including JavaScript file into HTML File



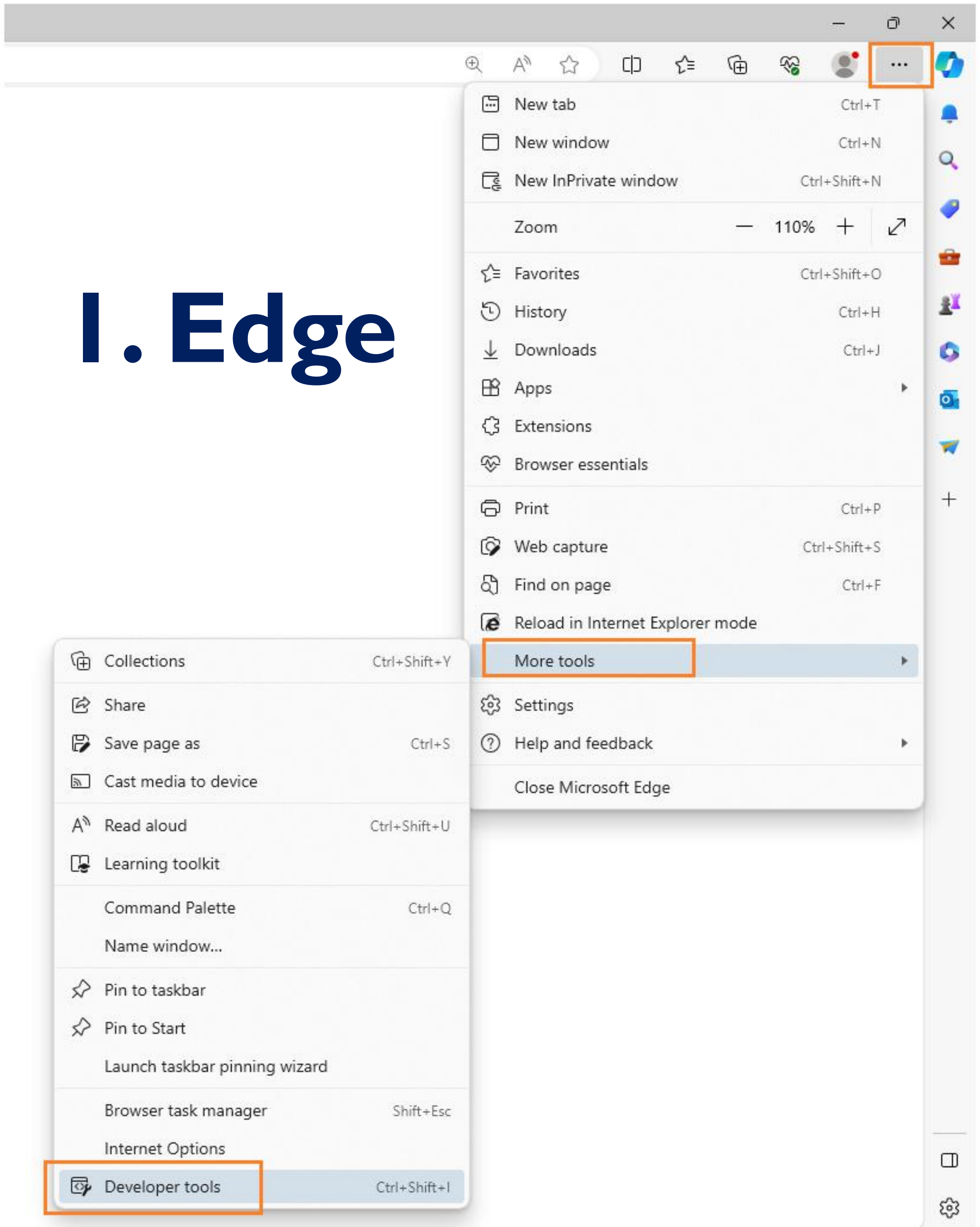
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script src="hello.js"></script>
</body>
</html>
```


index.html
javascript-demos > index.html > html > body
Go to Definition F12
Go to References Shift+F12
Peek
Find All References Shift+Alt+F12
Rename Symbol F2
Change All Occurrences Ctrl+F2
Format Document Shift+Alt+F
Format Document With...
Refactor... Ctrl+Shift+R
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Spelling
Open with Live Server Alt+L Alt+O
Stop Live Server Alt+L Alt+C
Command Palette... Ctrl+Shift+P

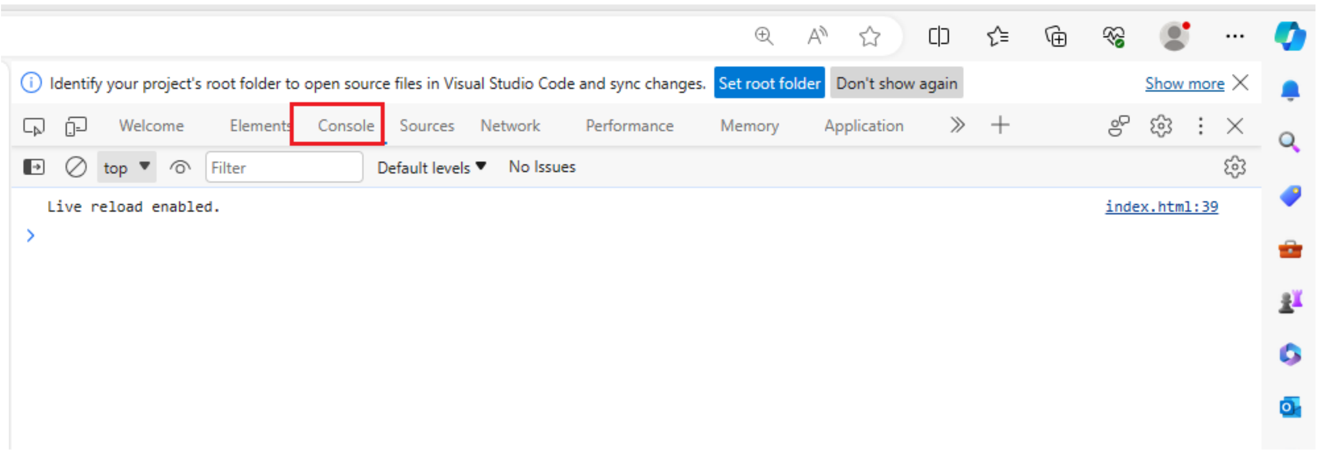
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script src="hello.js"></script>
</body>
</html>
```

Opening Console on Edge OR Chrome browsers

1. Edge



Control + Shift + J

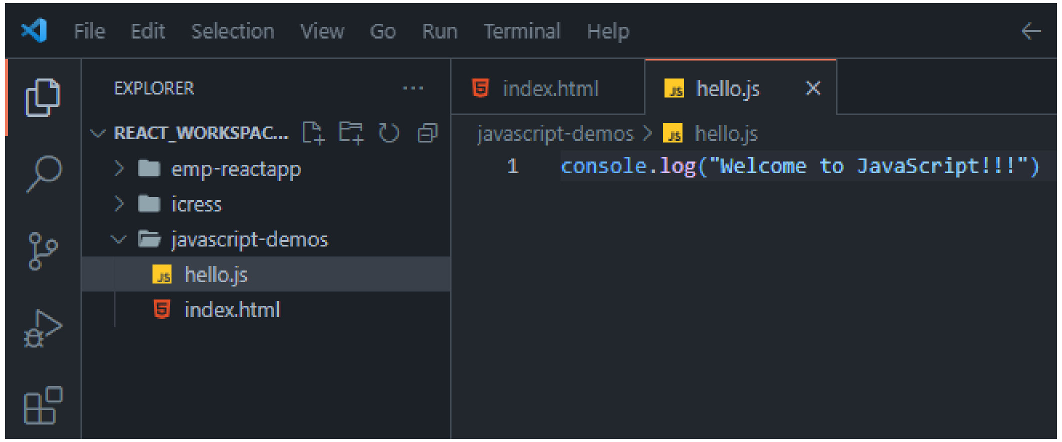


Identify your project's root folder to open source files in Visual Studio Code and sync changes.

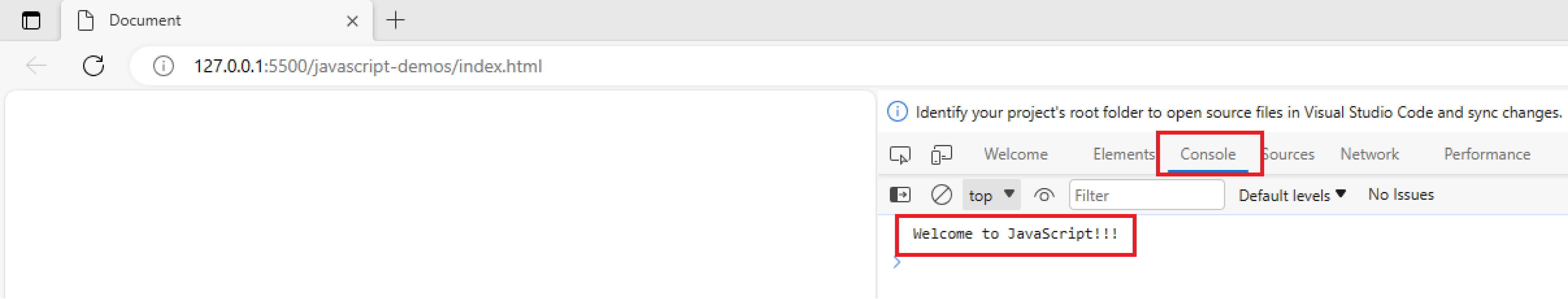
```
Live reload enabled.
```

2. Chrome F12

Hello Word Program in JavaScript



```
console.log("Welcome to JavaScript!!!")
```



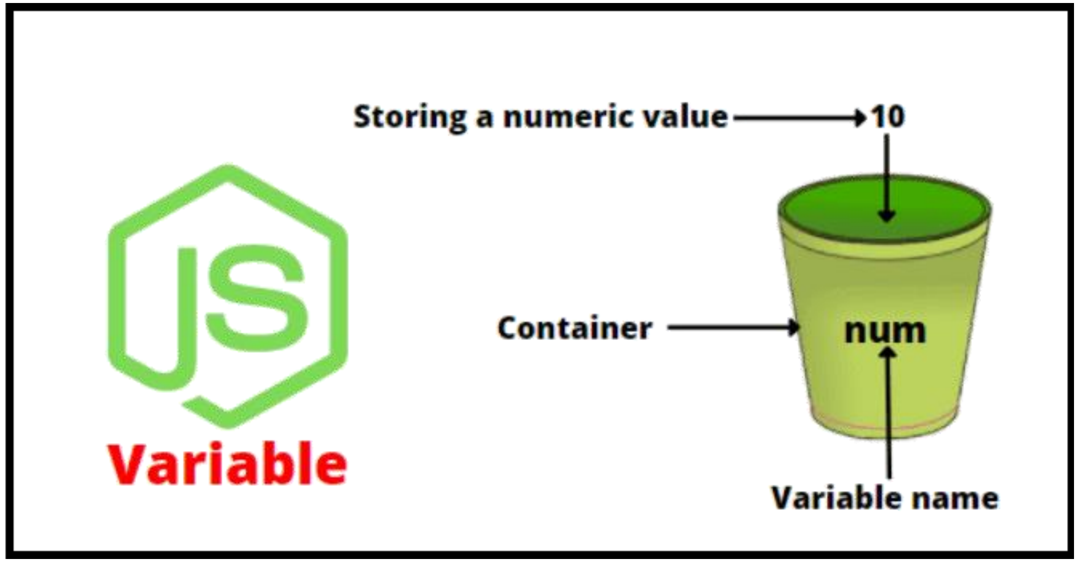
Identify your project's root folder to open source files in Visual Studio Code and sync changes.

```
Welcome to JavaScript!!!
```

Variables in JavaScript

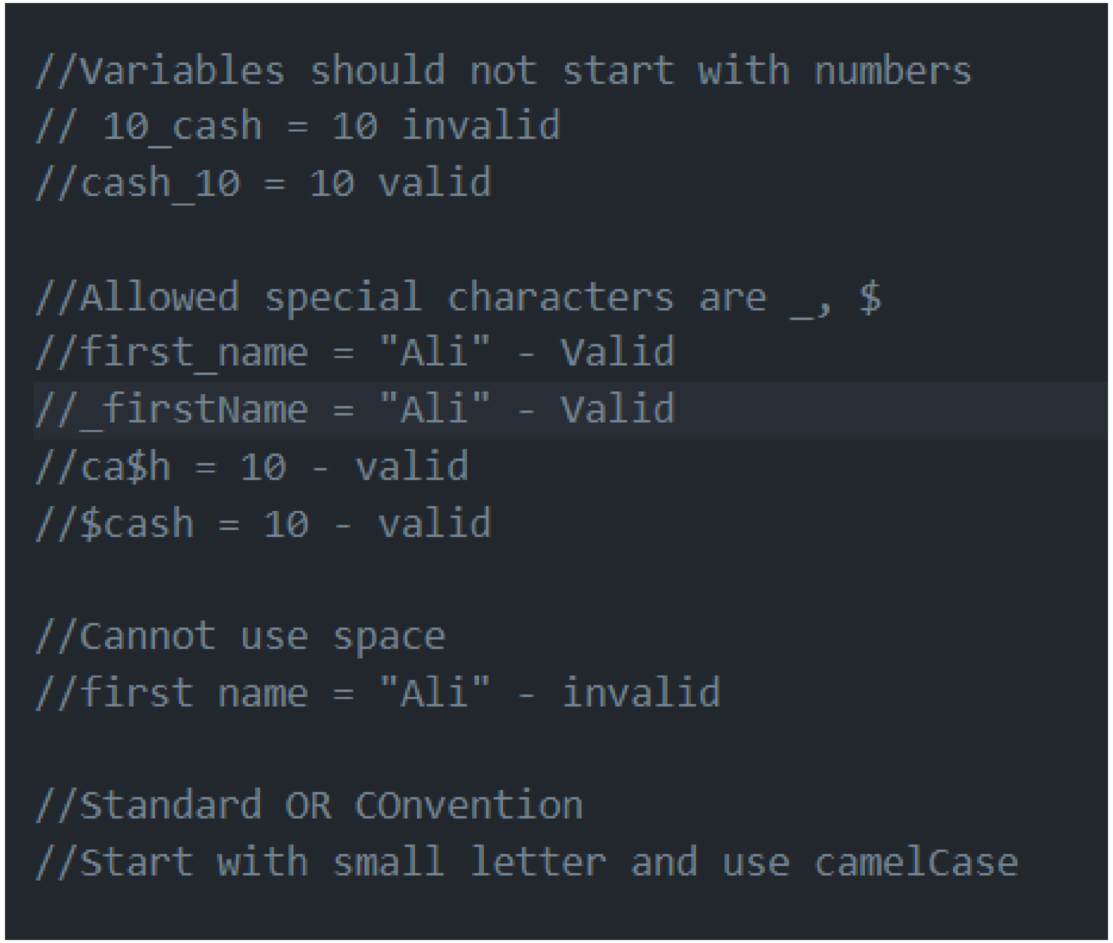
- Variables are memory locations used to store some value which can be changed depending on the conditions or information being passed to the program.

- Variables in JavaScript can be defined using following keywords
  - ✓ var
  - ✓ let
  - ✓ const



Rules of declaring Variables

- Allowed characters a to z, A to Z, 0 to 9, $, _
- Variable names should not start with numbers
- Standard OR Convention is variable should tart with small letter and use camelCase



```
//Variables should not start with numbers
// 10_cash = 10 invalid
//cash_10 = 10 valid

//Allowed special characters are _, $
//first_name = "Ali" - Valid
//_firstName = "Ali" - Valid
//ca$h = 10 - valid
//$cash = 10 - valid

//Cannot use space
//first name = "Ali" - invalid

//Standard OR COnvention
//Start with small letter and use camelCase
```

Variable types in JavaScript

- C and Java are strongly typed programing languages which means when we declare a variable in a program, we must declare its type of variable before it used. Once a variable has defined, we cannot change its data type.

- Variables in JavaScript are weakly types (i.e. not strongly typed),which means a variable will hold an integer, a floating-point number, or a string simultaneously. We can also change the value of a variable through simple reassignment.

- **Example**

```
var x = 20;
x = "Now it is a string.";
x = true;
x = undefined;
```

Declaring Variables in JavaScript

- ✓ var
- ✓ let
- ✓ const

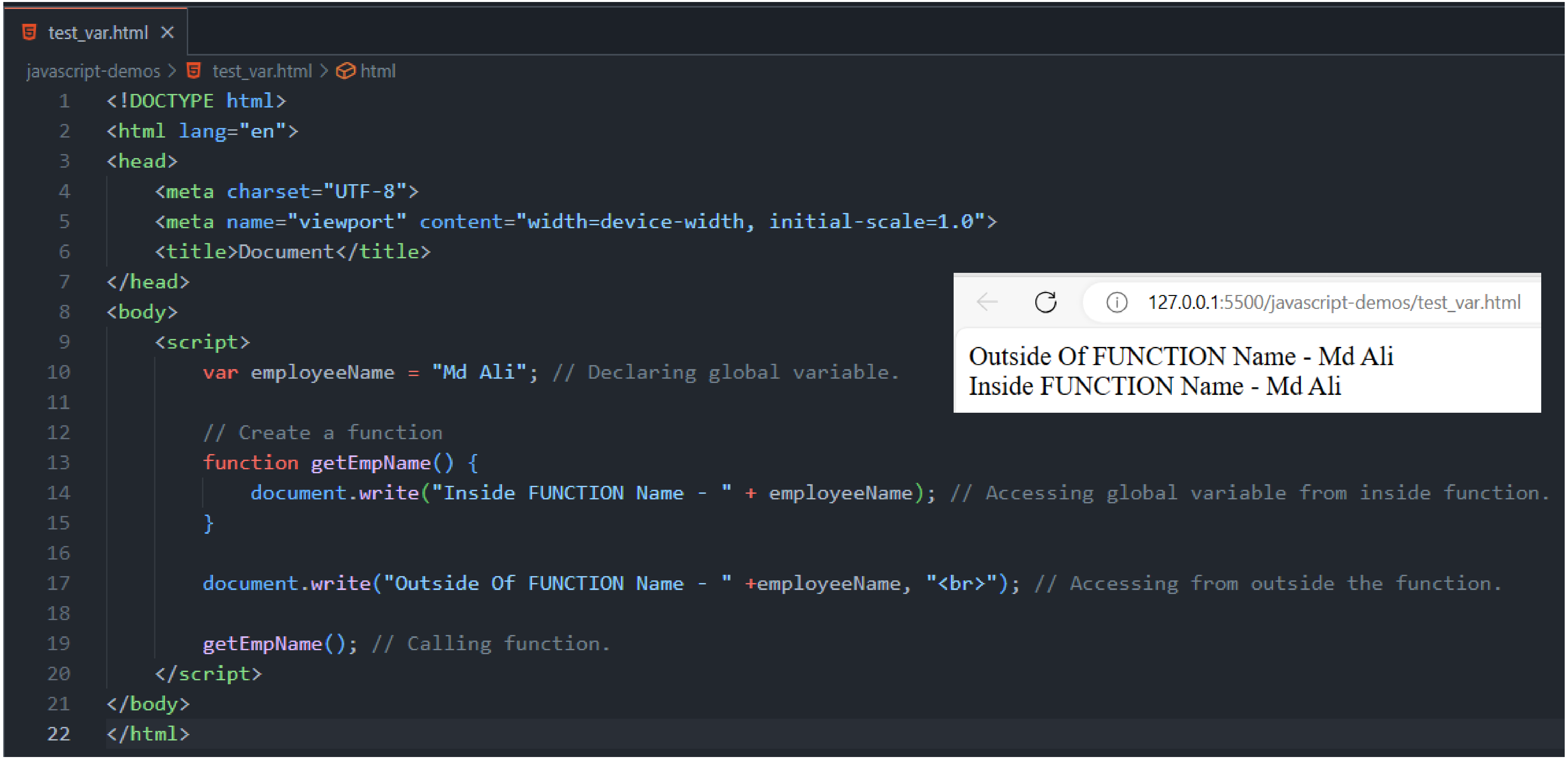
Types of Variables in JavaScript (OR) Variable Scope

- There are two types of variables in JavaScript
  - ✓ Global variable
  - ✓ Local variable

Global Variables

- ✓ A global variable is a variable that is defined in the main part of the script but outside the function.
- ✓ Variable defined outside of the functions is called global variables.
- ✓ We can access a global variable throughout the JavaScript program. We can use them anywhere, even within functions
- ✓ Global variables will be destroyed when we close the web page.
- ✓ Example of creating a global variable with var keyword
  - *var employeeName = "Md Ali";*
  - It declares a global variable called employeeName and assigns it a value of "Md Ali".
  - Here, var keyword can be optional hence same statement we can declare as *employeeName = "Md Ali";*
- ✓ NOTE: it is a good practice to always use the var keyword to avoid errors and make the script easier to read.

Global Variables Demo



```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>
</head>
<body>
    <script>
        var employeeName = "Md Ali"; // Declaring global variable.

        // Create a function
        function getEmpName() {
            document.write("Inside FUNCTION Name - " + employeeName); // Accessing global variable from inside function.
        }

        document.write("Outside Of FUNCTION Name - " +employeeName, "<br>"); // Accessing from outside the function.

        getEmpName(); // Calling function.
    </script>
</body>
</html>
```

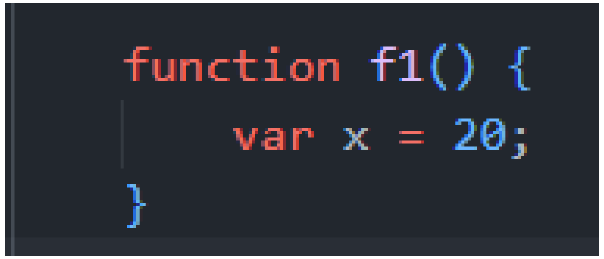
127.0.0.1:5500/javascript-demos/test_var.html

```
Outside Of FUNCTION Name - Md Ali
Inside FUNCTION Name - Md Ali
```

NOTE: Change var to let and try

Local Variables

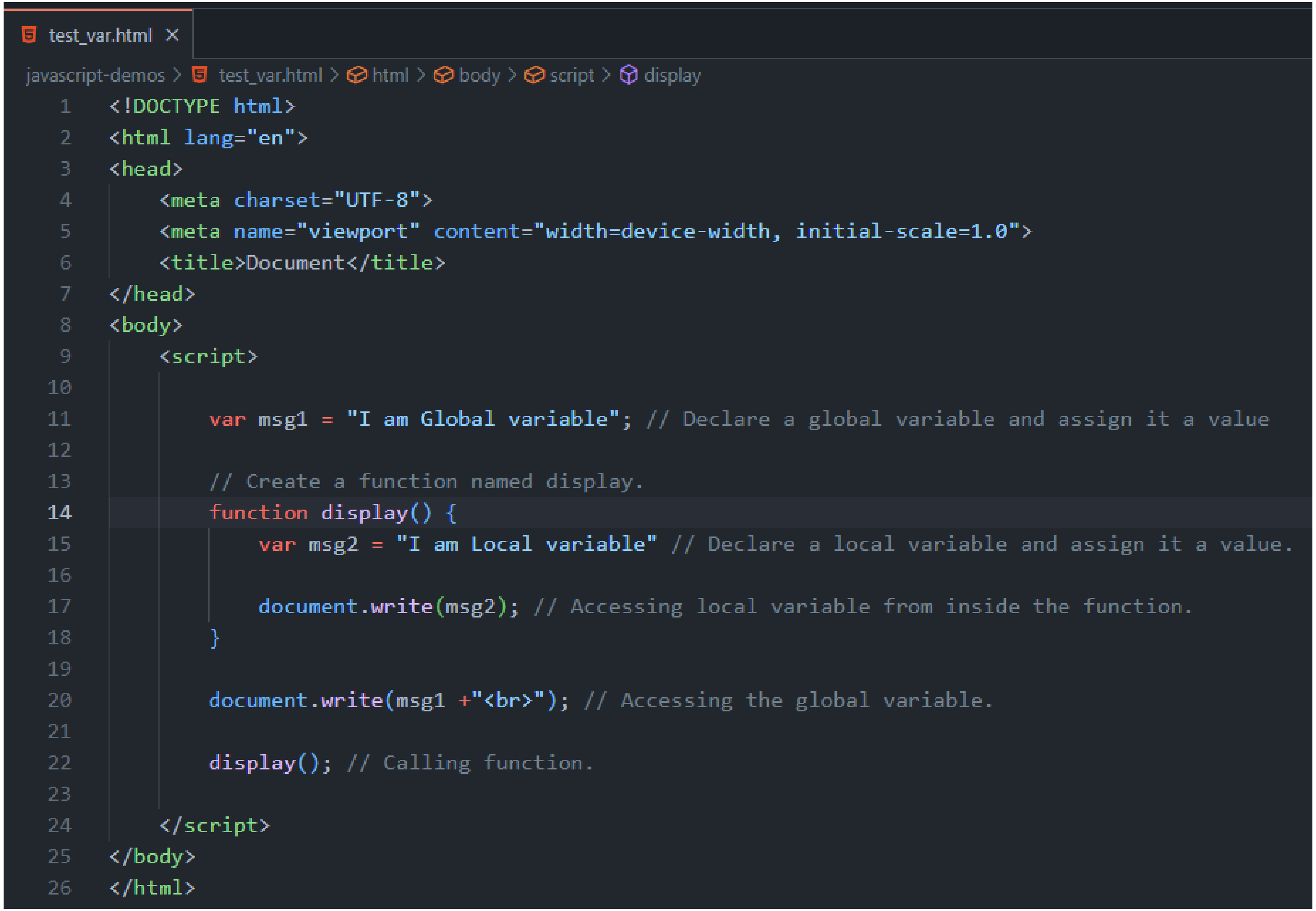
- Local variable is a variable that is defined inside the body of the function with var keyword.
- Local variables can be used or accessed only within the function where it is defined.



```
function f1() {
    var x = 20;
}
```

- In above example, we have defined a variable 'x' having a value '20'
- It is a local variable, as it is declared within a specific function
- When the execution of a function completes, JavaScript interpreter deletes the local variables from the memory.
- **NOTEs**
  - ✓ Variables declared in parameter list of function are always local variables and define only within the body of the function.
  - ✓ If we declare a local variable or functional parameter inside the body of a function with the same name as the global variable THEN local variable takes precedence over a global variable. In this case, we are hiding the global variable.

Global & Local Variables Demo

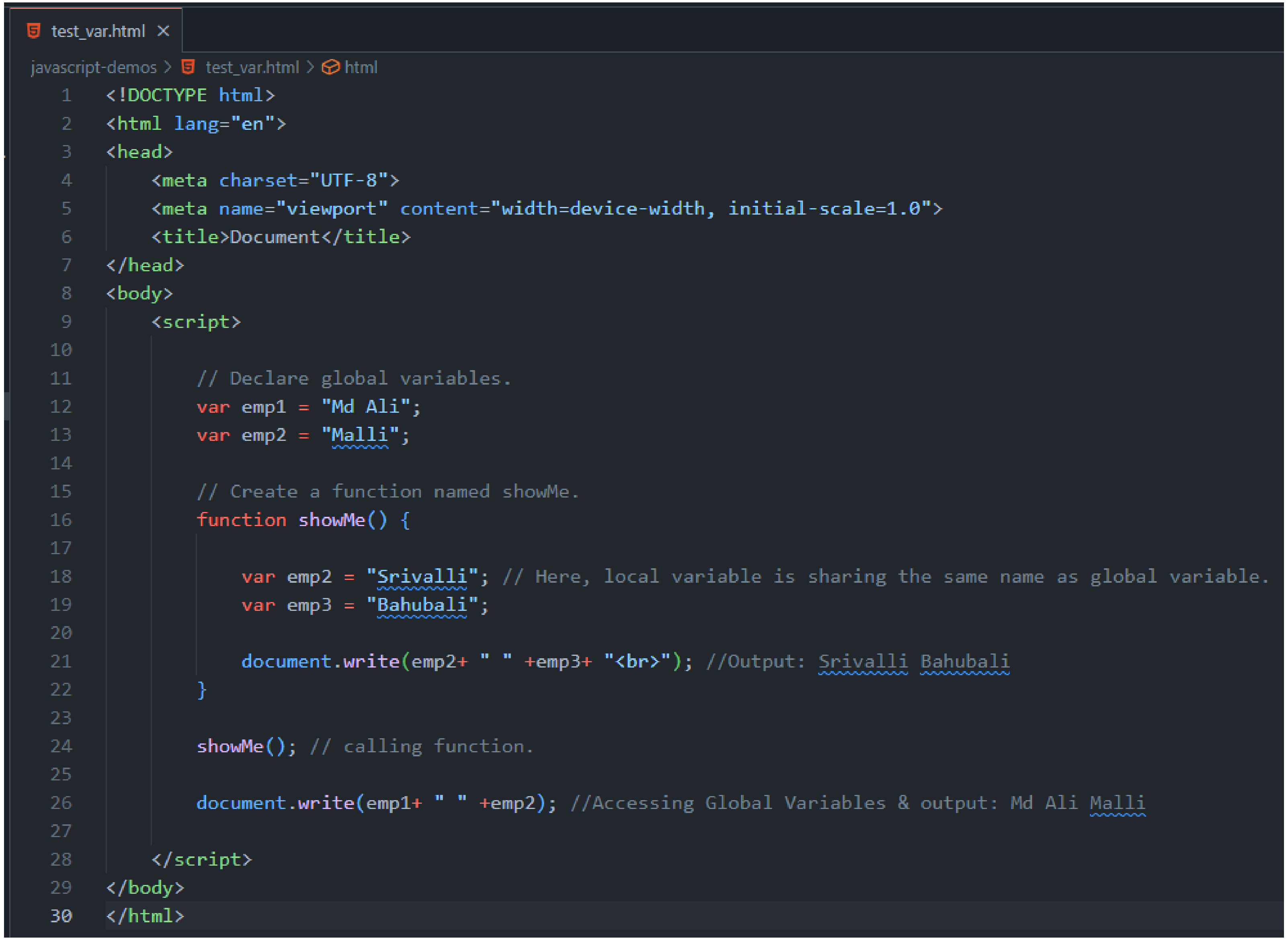

test_var.html ×
javascript-demos > test_var.html > html > body > script > display

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

        var msg1 = "I am Global variable"; // Declare a global variable and assign it a value

        // Create a function named display.
        function display() {
            var msg2 = "I am Local variable" // Declare a local variable and assign it a value.

            document.write(msg2); // Accessing local variable from inside the function.
        }

        document.write(msg1 +"<br>"); // Accessing the global variable.

        display(); // Calling function.

    </script>
</body>
</html>
```

Global & Local Variables Demo-2

Declare two global variables and two local variables within a function. One of the global and local variables will share a common name

NOTE: In this code, when we called variable **emp2** from the function, the value of local variable has displayed. But when we called emp2 from outside function, the value of global variable has displayed. Hence, having two variables with the same name is always confusing. Therefore, it is the best practice to use unique names for all variables to avoid confusion.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

        // Declare global variables.
        var emp1 = "Md Ali";
        var emp2 = "Malli";

        // Create a function named showMe.
        function showMe() {

            var emp2 = "Srivalli"; // Here, local variable is sharing the same name as global variable.
            var emp3 = "Bahubali";

            document.write(emp2+ " " +emp3+ "<br>"); //Output: Srivalli Bahubali
        }

        showMe(); // calling function.

        document.write(emp1+ " " +emp2); //Accessing Global Variables & output: Md Ali Malli

    </script>
</body>
</html>
```

Let in JavaScript

- Let keyword introduced in ECMAScript 6 (ES6)
- Let is used to declare a block-level variable rather than a global variable or variable in a function block.
- let keyword follow the same definition as var, with one exception i.e., let is tied to the block scope, not the function scope.
- Variable defined with let keyword is visible only within the scope in which it is defined.

- **Syntax**
  - ✓ let empName; // Declare a variable.
    empName = 'Md Ali'; // Initialize a variable.
  - ✓ *let empName = 'Md Ali';* // Declare and initialize a variable.

- We can also declare multiple variables using let keyword in a single line

  let empId, empName, salary // Declaration of 3 variables.

  let empId = 1001, empName = "Md Ali", salary = 25000

Why do we need let keyword

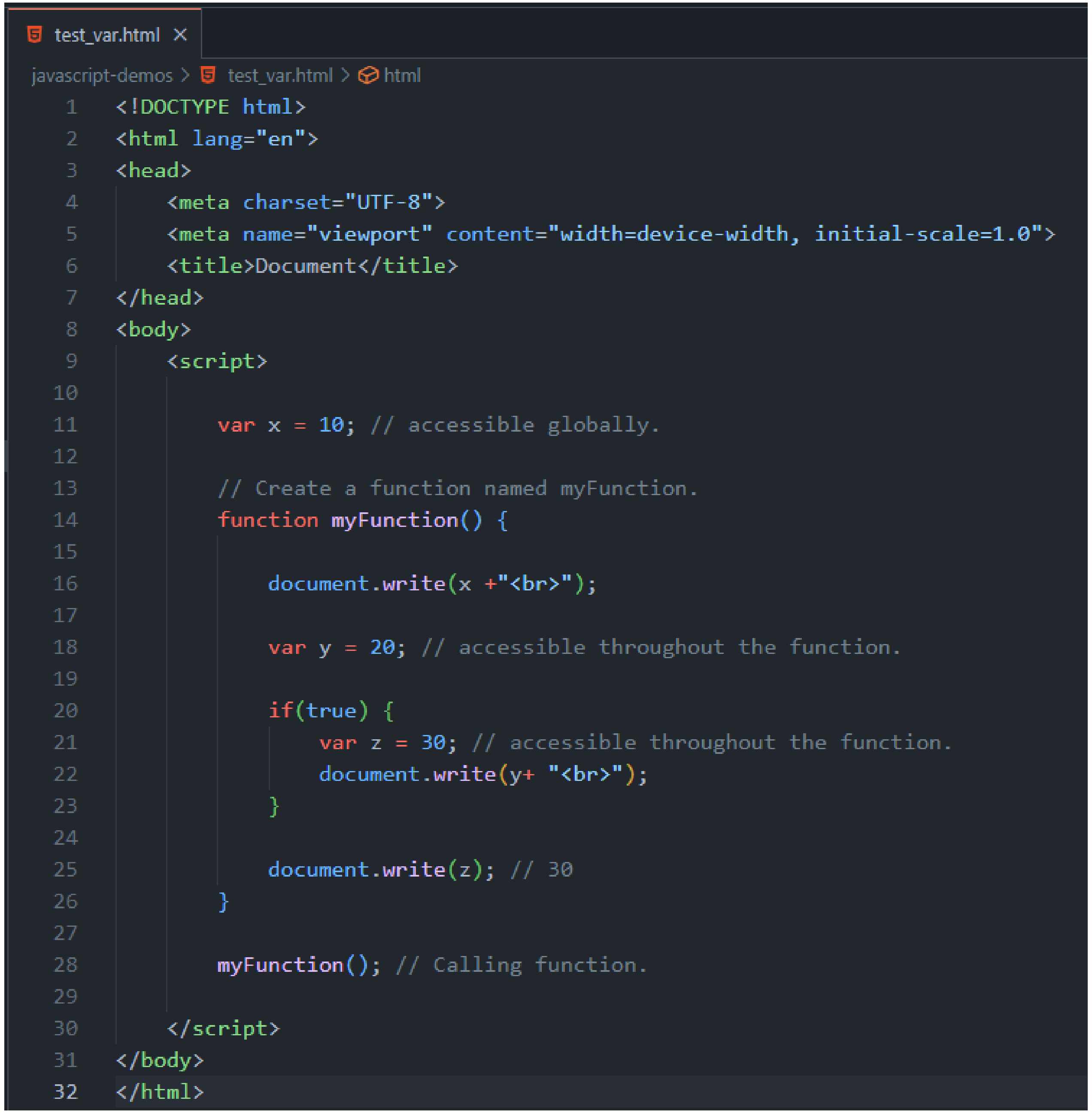
Let us understand ISSUES with **var**

✓ When a variable is declared using var keyword in JavaScript, it is called function scoped variable. It is accessible globally to the script or throughout the script if declared outside the function.

✓ Similarly, if the function scoped variable is declared inside a function, it is accessible throughout the function, but not outside the function.

✓ The variable z is a local variable inside the if statement. The variable z is still accessible after the if statement has finished execution.

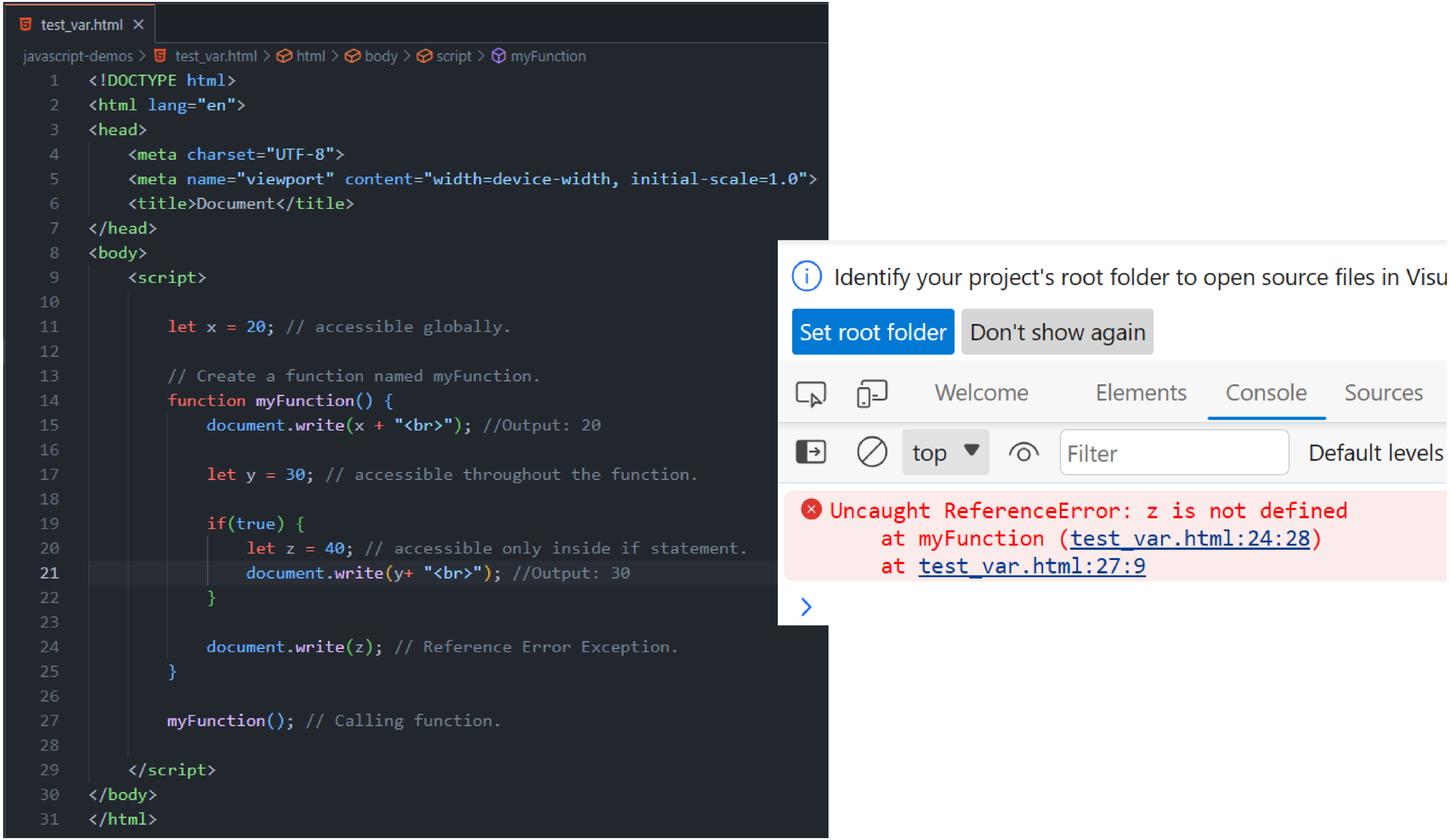
✓ To overcome this problem, JavaScript has introduced let keyword in ES6



```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>
</head>
<body>
    <script>

        var x = 10; // accessible globally.

        // Create a function named myFunction.
        function myFunction() {

            document.write(x +"<br>");

            var y = 20; // accessible throughout the function.

            if(true) {
                var z = 30; // accessible throughout the function.
                document.write(y+ "<br>");
            }

            document.write(z); // 30
        }

        myFunction(); // Calling function.

    </script>
</body>
</html>
```

Block Scoped Variables / let keyword Scope

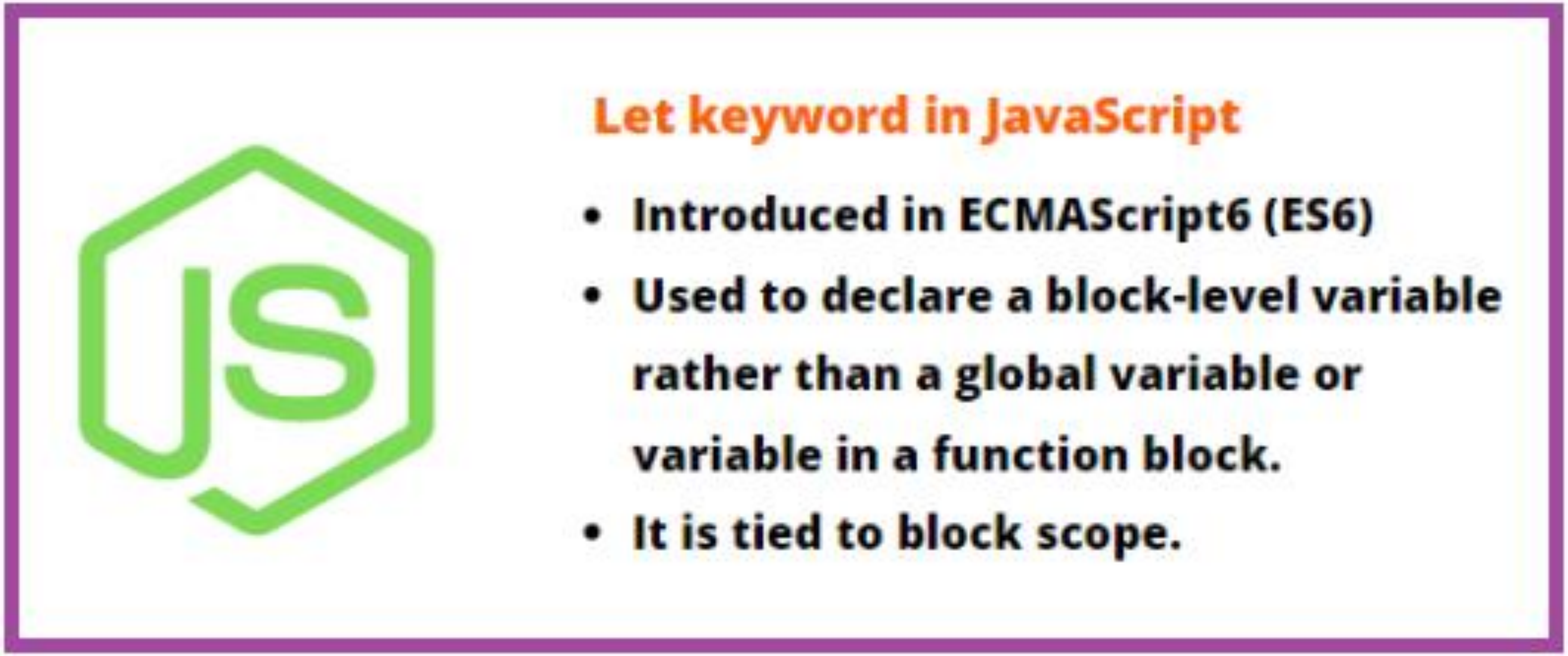
- Variables that are defined using the let keyword are called block scoped variables in JavaScript. Block scoped variables are locked into a block.
- Block Scoped Variables exist only within the current block. They cannot be accessed outside a block and they are removed from the memory as soon as the block finishes the execution.
- A block is a group of statements in Javascript that is created with curly braces { . . . .}
- The block-scoped variables work the same way as function scoped variables when declared outside the function, they are accessed globally.
- But when the block scoped variables are defined inside a block, they are only accessible inside the block in which they are defined, but not outside the function.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

        let x = 20; // accessible globally.

        // Create a function named myFunction.
        function myFunction() {
            document.write(x + "<br>"); //Output: 20

            let y = 30; // accessible throughout the function.

            if(true) {
                let z = 40; // accessible only inside if statement.
                document.write(y+ "<br>"); //Output: 30
            }

            document.write(z); // Reference Error Exception.
        }

        myFunction(); // Calling function.

    </script>
</body>
</html>
```

Identify your project's root folder to open source files in Visu

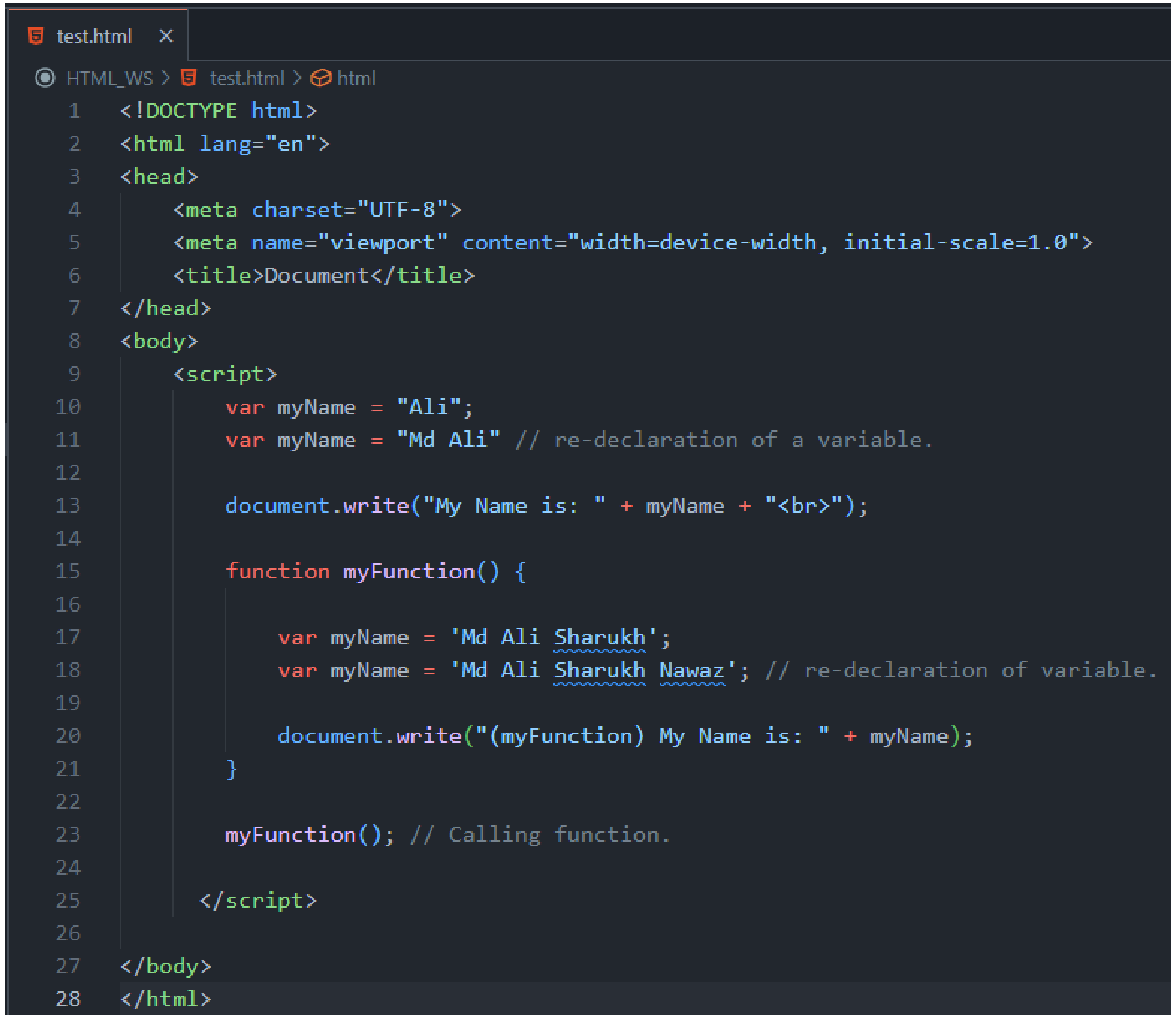
```
Uncaught ReferenceError: z is not defined
    at myFunction (test_var.html:24:28)
    at test_var.html:27:9
```



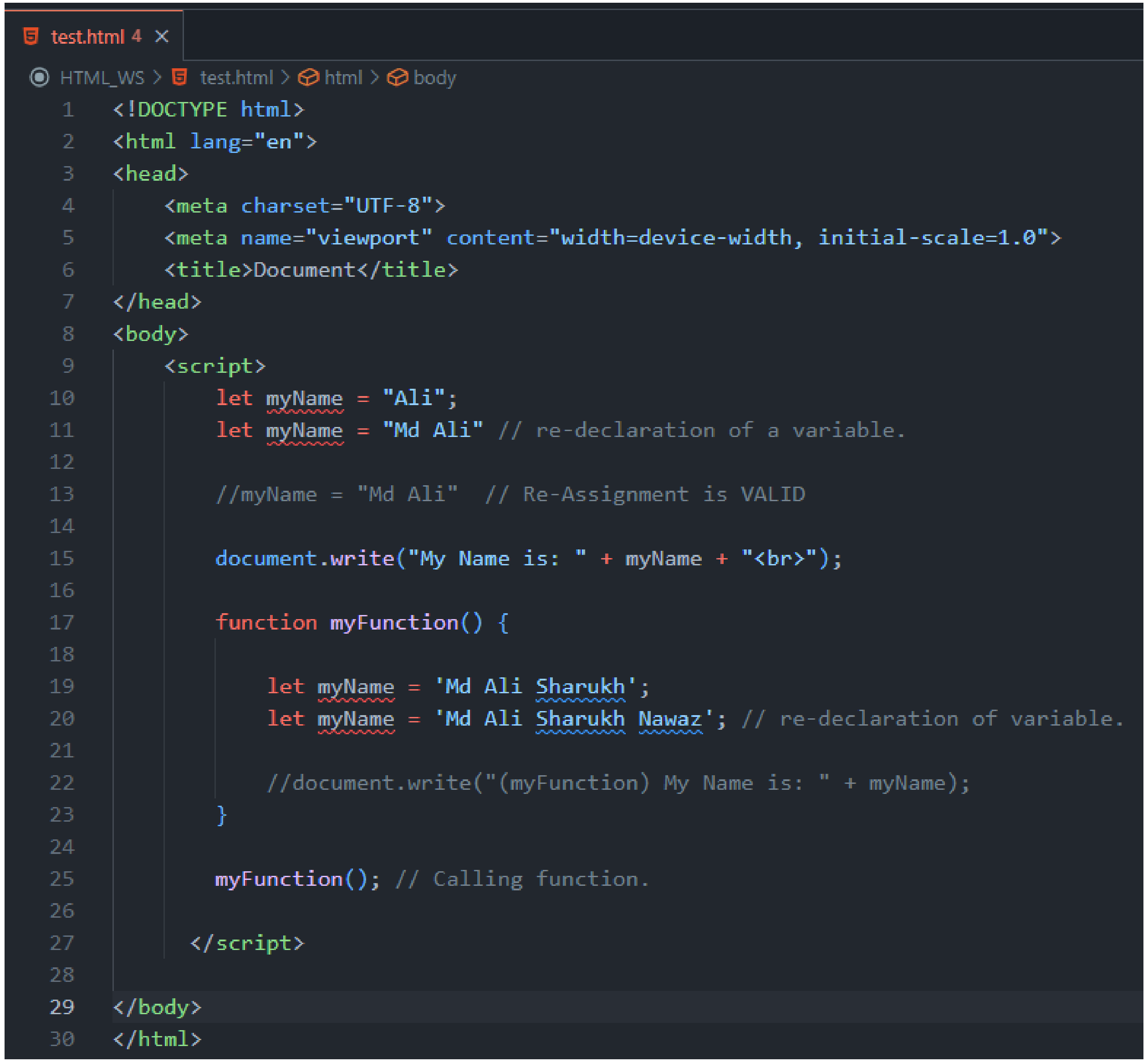
Let keyword in JavaScript

- Introduced in ECMAScript6 (ES6)
- Used to declare a block-level variable rather than a global variable or variable in a function block.
- It is tied to block scope.

Variables Re-declaration in JavaScript using var & let



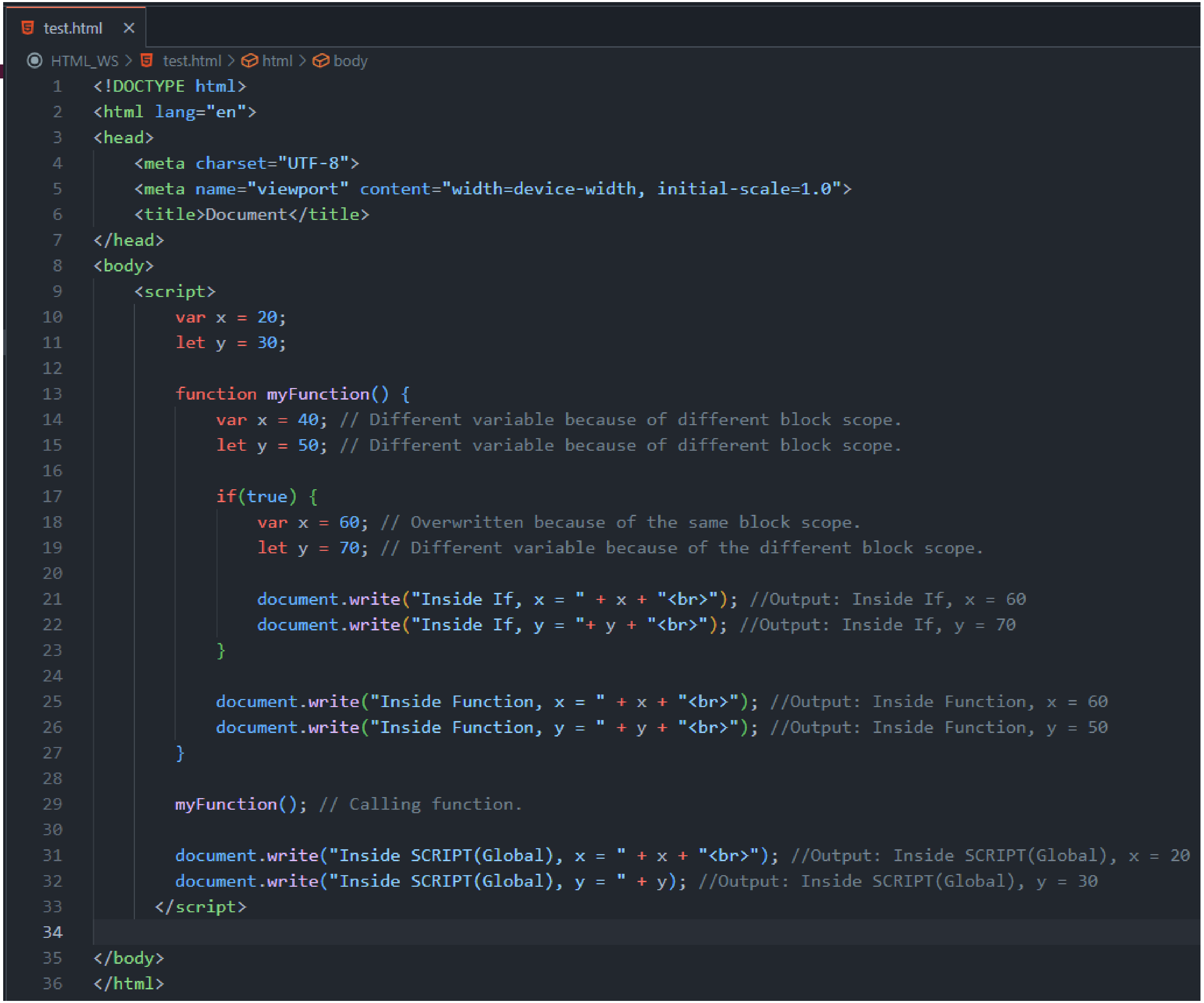
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var myName = "Ali";
        var myName = "Md Ali" // re-declaration of a variable.

        document.write("My Name is: " + myName + "<br>");

        function myFunction() {

            var myName = 'Md Ali Sharukh';
            var myName = 'Md Ali Sharukh Nawaz'; // re-declaration of variable.

            document.write("(myFunction) My Name is: " + myName);
        }

        myFunction(); // Calling function.

      </script>

</body>
</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>
</head>
<body>
    <script>
        let myName = "Ali";
        let myName = "Md Ali" // re-declaration of a variable.

        //myName = "Md Ali"  // Re-Assignment is VALID

        document.write("My Name is: " + myName + "<br>");

        function myFunction() {

            let myName = 'Md Ali Sharukh';
            let myName = 'Md Ali Sharukh Nawaz'; // re-declaration of variable.

            //document.write("(myFunction) My Name is: " + myName);
        }

        myFunction(); // Calling function.

      </script>

</body>
</html>
```

NOTE: If we declare a variable using **let** keyword that is already declared using let keyword in the same scope, then JavaScript interpreter will throw *TypeError* exception.

Understanding Global, Local Variables & Its Scope



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    var x = 20;
    let y = 30;

    function myFunction() {
      var x = 40; // Different variable because of different block scope.
      let y = 50; // Different variable because of different block scope.

      if(true) {
        var x = 60; // Overwritten because of the same block scope.
        let y = 70; // Different variable because of the different block scope.

        document.write("Inside If, x = " + x + "<br>"); //Output: Inside If, x = 60
        document.write("Inside If, y = "+ y + "<br>"); //Output: Inside If, y = 70
      }

      document.write("Inside Function, x = " + x + "<br>"); //Output: Inside Function, x = 60
      document.write("Inside Function, y = " + y + "<br>"); //Output: Inside Function, y = 50
    }

    myFunction(); // Calling function.

    document.write("Inside SCRIPT(Global), x = " + x + "<br>"); //Output: Inside SCRIPT(Global), x = 20
    document.write("Inside SCRIPT(Global), y = " + y); //Output: Inside SCRIPT(Global), y = 30
   </script>

</body>
</html>
```

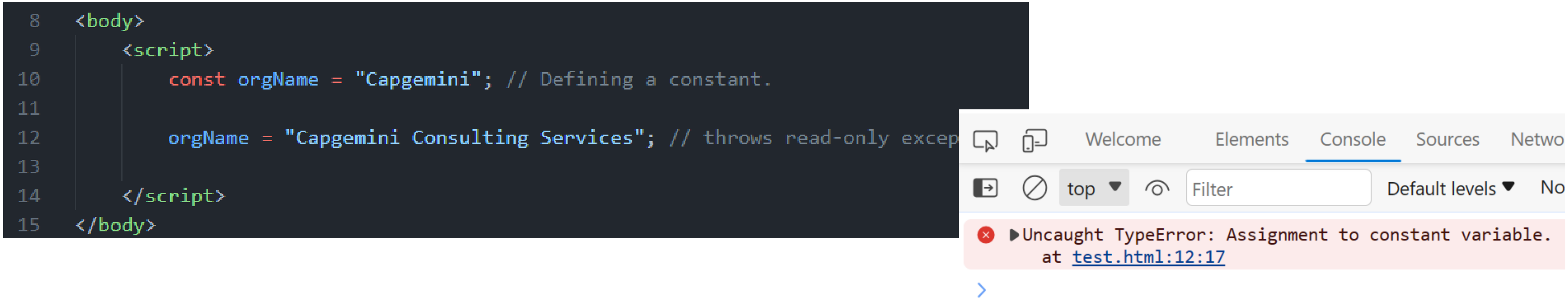
Advantages of let Keyword

If we are using ES6 or greater standard, **let** is recommend to use because of following advantages

- ✓ Let keyword makes script more memory friendly.
- ✓ Let helps to prevent the scoping mistake.
- ✓ Let prevents the accidental bugs in the script code.
- ✓ Let keyword makes the script code easier to read.

const Keyword in JavaScript

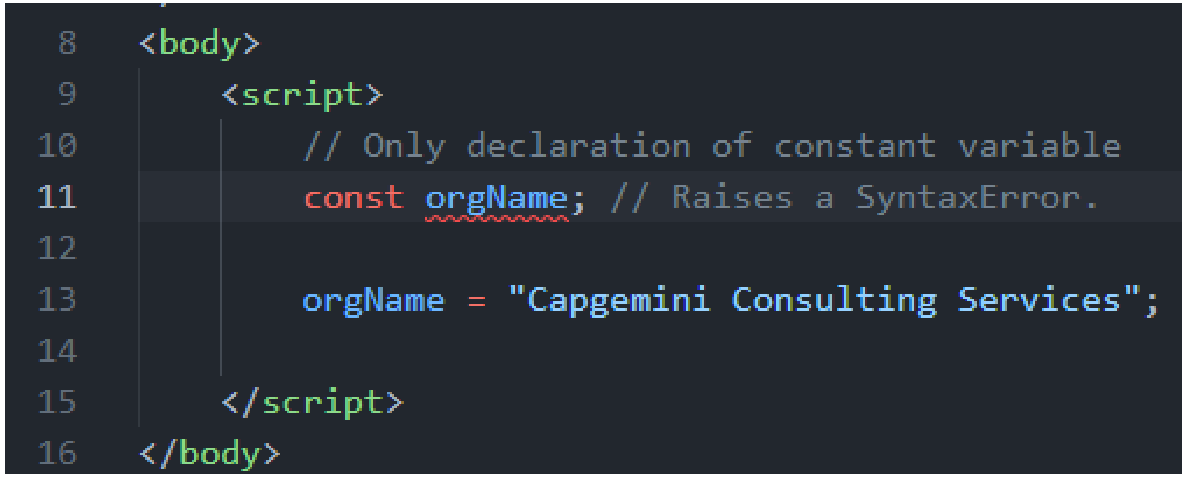
- **const** is another keyword introduced in ES6, It is used to declare constant variables i.e., once the variable is declared value cannot be reassigned.
- If a variable is defined with a **const** keyword, its value cannot be changed (or modified) throughout its lifetime.
- **const** keyword allows us to define a constant value. We must initialize it when the variable is declared.
- If a value of the variable assigned with a **const** keyword is modified, TypeError will be thrown.



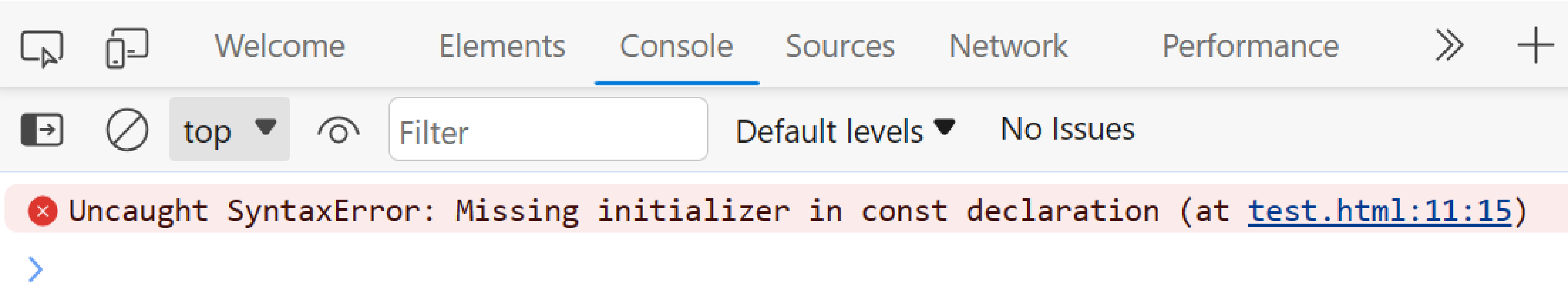
```
<body>
    <script>
        const orgName = "Capgemini"; // Defining a constant.

        orgName = "Capgemini Consulting Services"; // throws read-only excep

    </script>
</body>
```

```
Uncaught TypeError: Assignment to constant variable.
    at test.html:12:17
```

const Keyword in JavaScript…



```
<body>
    <script>
        // Only declaration of constant variable
        const orgName; // Raises a SyntaxError.

        orgName = "Capgemini Consulting Services";

    </script>
</body>
```

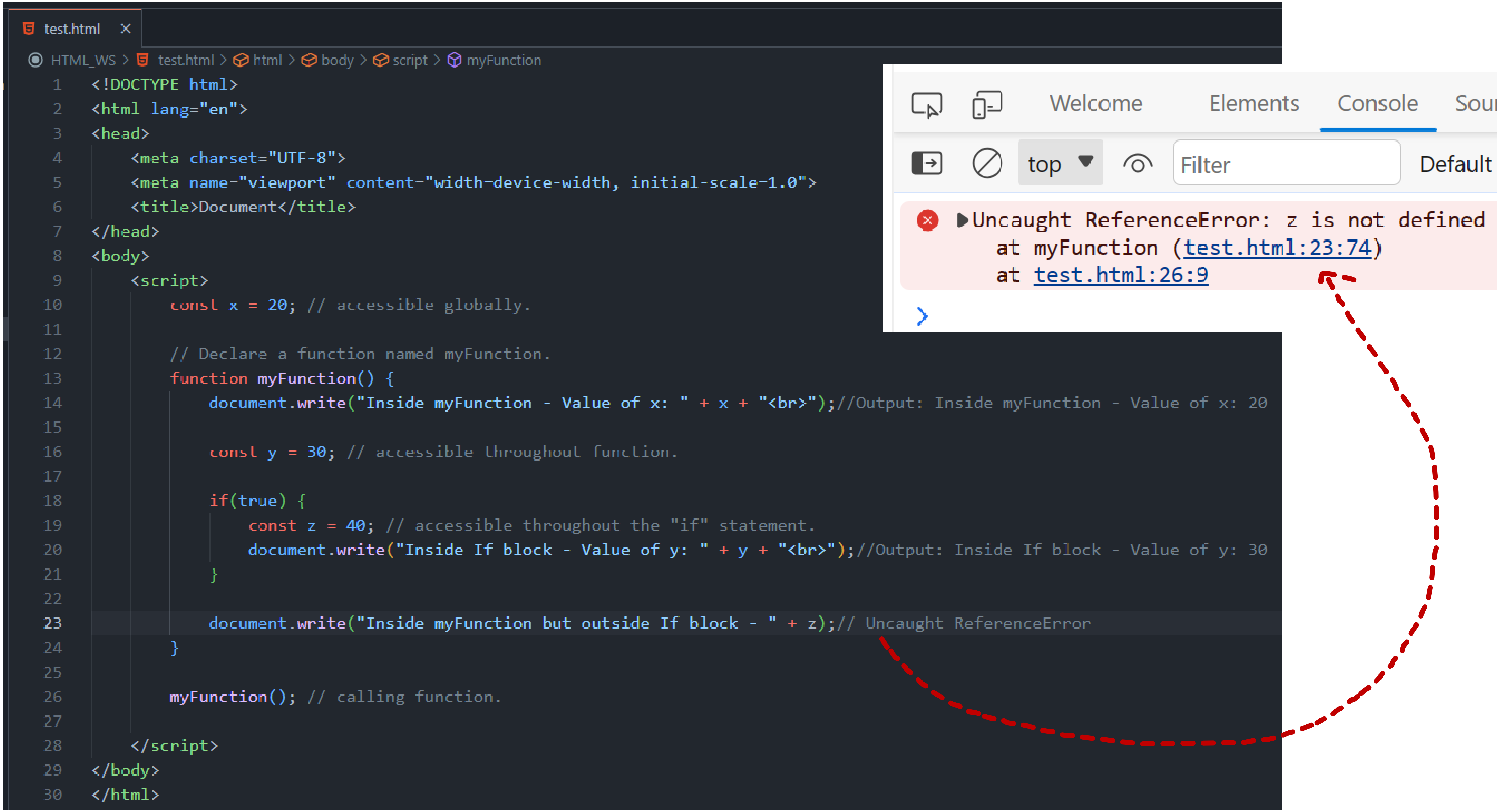


```
Uncaught SyntaxError: Missing initializer in const declaration (at test.html:11:15)
```

Scope of Const keyword

Variables declared with a **const** keyword are **blocked-scope** variables. Constant variables follow the same scoping rules as variables that are declared with the **let** keyword.

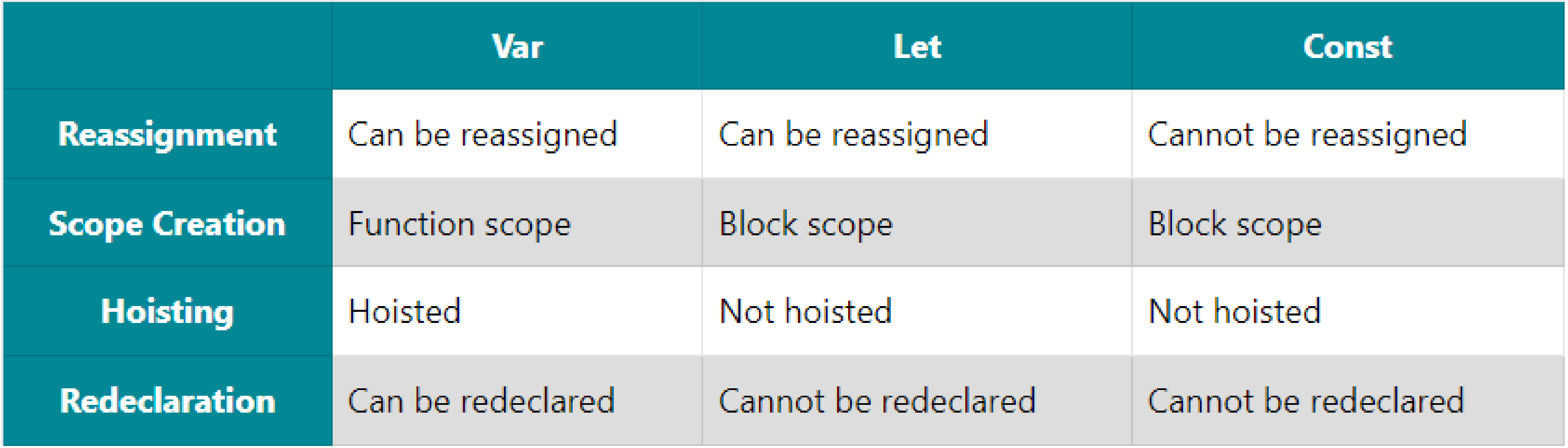
Const also supports blocked scope



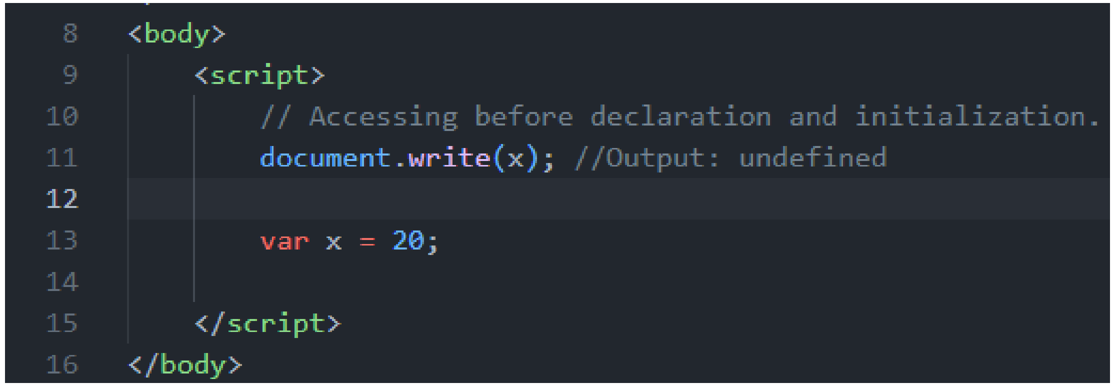
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        const x = 20; // accessible globally.

        // Declare a function named myFunction.
        function myFunction() {
            document.write("Inside myFunction - Value of x: " + x + "<br>");//Output: Inside myFunction - Value of x: 20

            const y = 30; // accessible throughout function.

            if(true) {
                const z = 40; // accessible throughout the "if" statement.
                document.write("Inside If block - Value of y: " + y + "<br>");//Output: Inside If block - Value of y: 30
            }

            document.write("Inside myFunction but outside If block - " + z);// Uncaught ReferenceError
        }

        myFunction(); // calling function.

    </script>
</body>
</html>
```

```
Uncaught ReferenceError: z is not defined
    at myFunction (test.html:23:74)
    at test.html:26:9
```

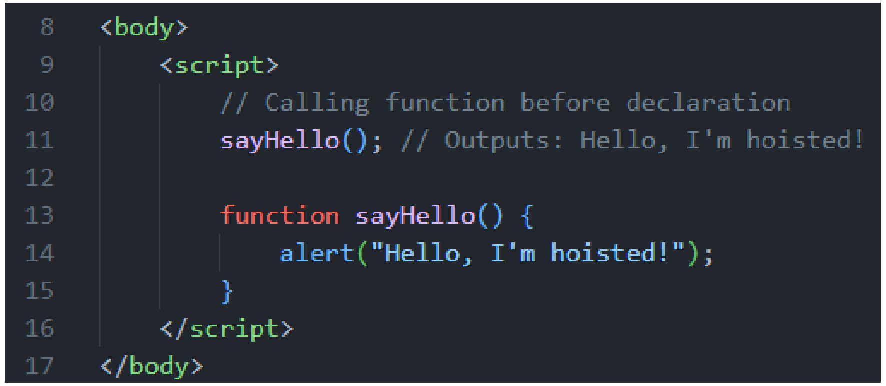
Difference between var, let and const



	Var	Let	Const
Reassignment	Can be reassigned	Can be reassigned	Cannot be reassigned
Scope Creation	Function scope	Block scope	Block scope
Hoisting	Hoisted	Not hoisted	Not hoisted
Redeclaration	Can be redeclared	Cannot be redeclared	Cannot be redeclared

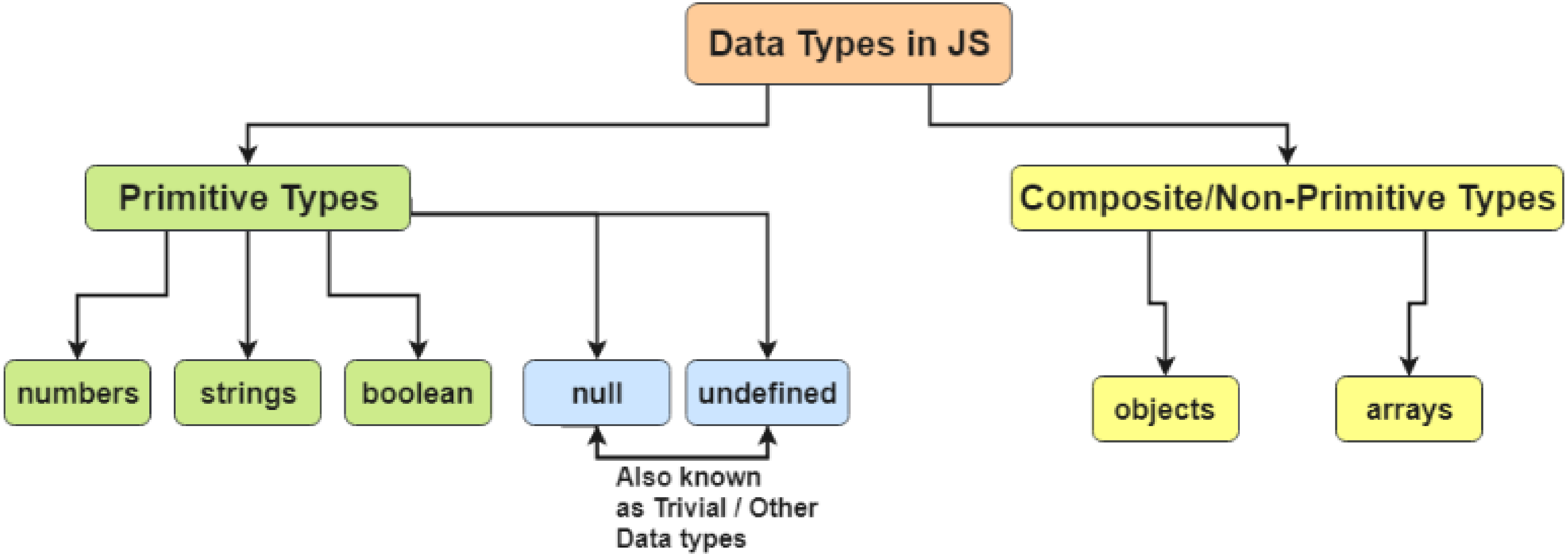


```
<body>
    <script>
        // Accessing before declaration and initialization.
        document.write(x); //Output: undefined

        var x = 20;

    </script>
</body>
```



```
<body>
    <script>
        // Calling function before declaration
        sayHello(); // Outputs: Hello, I'm hoisted!

        function sayHello() {
            alert("Hello, I'm hoisted!");
        }
    </script>
</body>
```

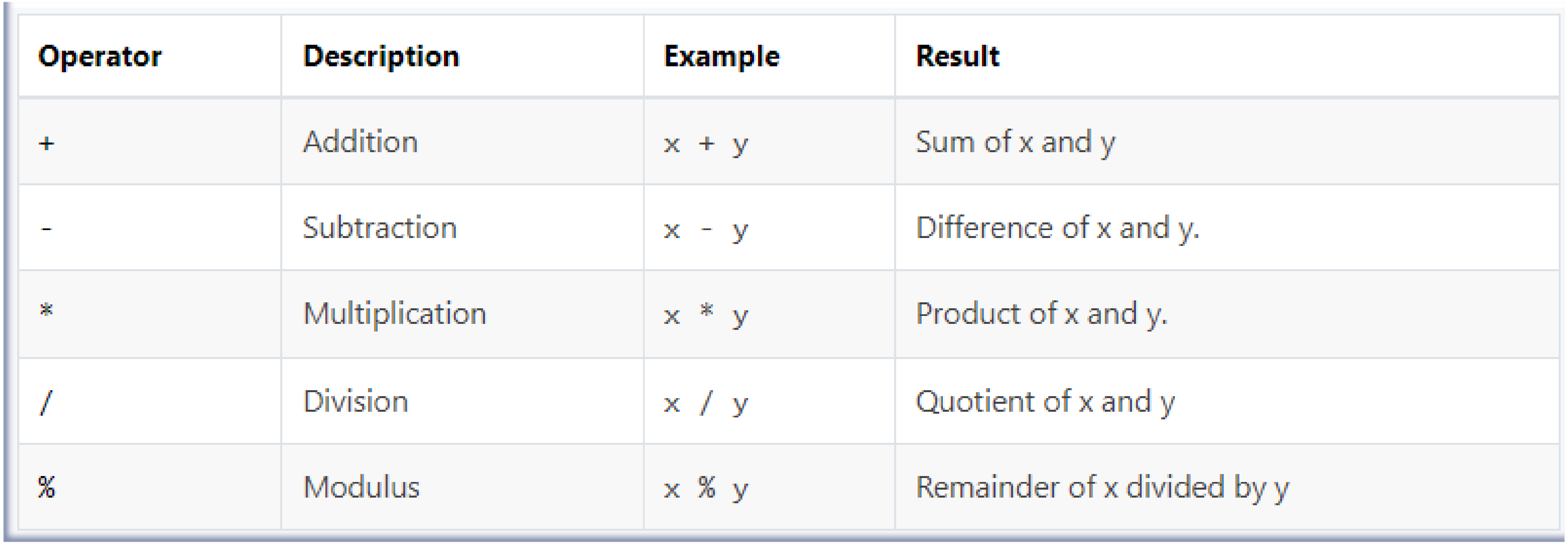
Data types in JavaScript



Types of Operators in JavaScript

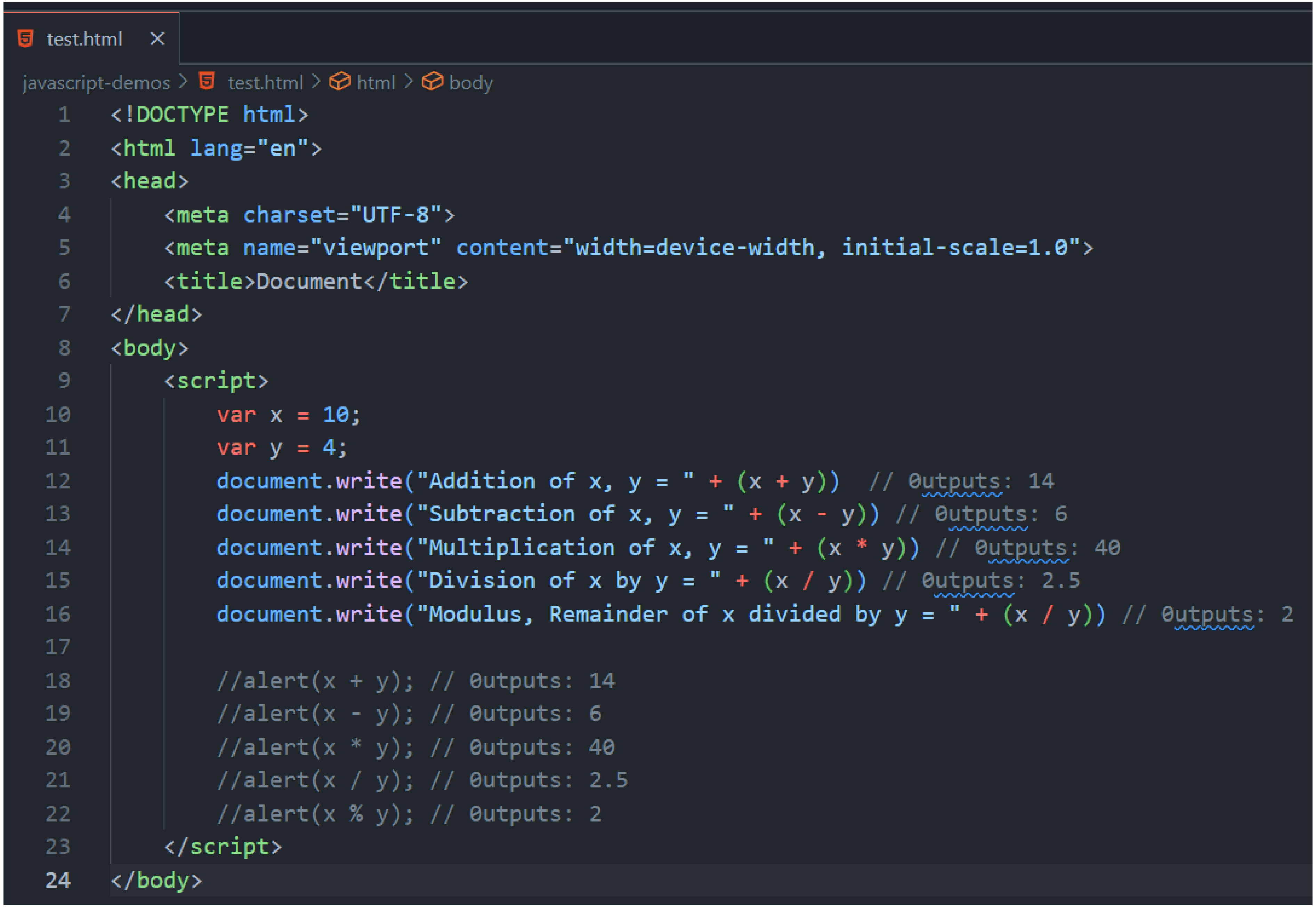
- Arithmetic Operators(Mathematical)
- Assignment Operators
- Comparison (Relational) Operators
- Logical Operators
- Conditional Operators
- String Operators
- Special Operators

Arithmetic Operators(Mathematical)



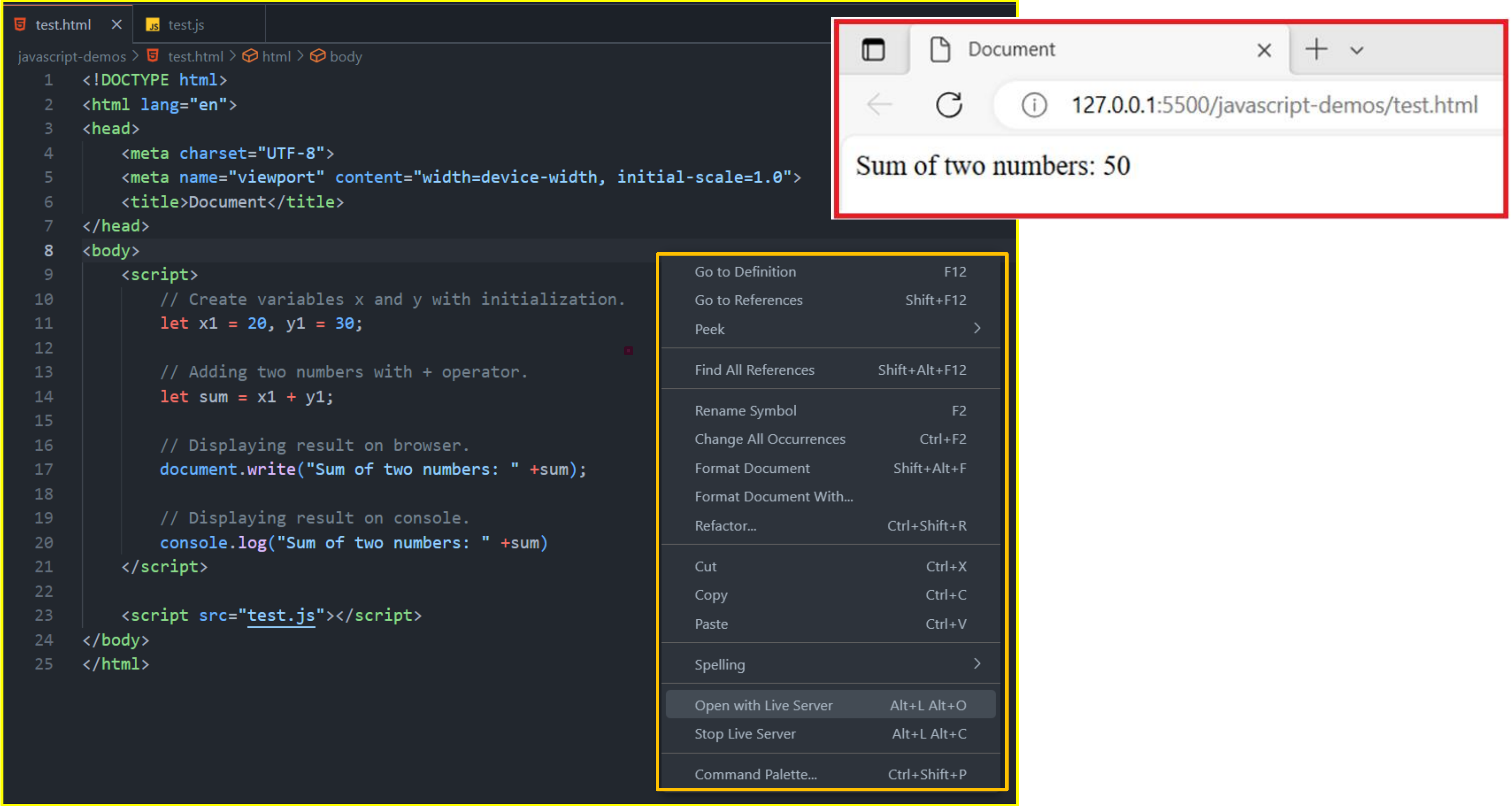
Operator	Description	Example	Result
+	Addition	x + y	Sum of x and y
-	Subtraction	x - y	Difference of x and y.
*	Multiplication	x * y	Product of x and y.
/	Division	x / y	Quotient of x and y
%	Modulus	x % y	Remainder of x divided by y

Arithmetic Operators(Mathematical)…



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 10;
        var y = 4;
        document.write("Addition of x, y = " + (x + y))  // Outputs: 14
        document.write("Subtraction of x, y = " + (x - y)) // Outputs: 6
        document.write("Multiplication of x, y = " + (x * y)) // Outputs: 40
        document.write("Division of x by y = " + (x / y)) // Outputs: 2.5
        document.write("Modulus, Remainder of x divided by y = " + (x / y)) // Outputs: 2

        //alert(x + y); // Outputs: 14
        //alert(x - y); // Outputs: 6
        //alert(x * y); // Outputs: 40
        //alert(x / y); // Outputs: 2.5
        //alert(x % y); // Outputs: 2
    </script>
</body>
```

Arithmetic Operators(Mathematical)…



test.html
test.js
javascript-demos > test.html > html > body

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // Create variables x and y with initialization.
        let x1 = 20, y1 = 30;

        // Adding two numbers with + operator.
        let sum = x1 + y1;

        // Displaying result on browser.
        document.write("Sum of two numbers: " +sum);

        // Displaying result on console.
        console.log("Sum of two numbers: " +sum)
    </script>

    <script src="test.js"></script>
</body>
</html>
```

Go to Definition F12
Go to References Shift+F12
Peek
Find All References Shift+Alt+F12
Rename Symbol F2
Change All Occurrences Ctrl+F2
Format Document Shift+Alt+F
Format Document With...
Refactor... Ctrl+Shift+R
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Spelling
Open with Live Server Alt+L Alt+O
Stop Live Server Alt+L Alt+C
Command Palette... Ctrl+Shift+P
Document
127.0.0.1:5500/javascript-demos/test.html

```
Sum of two numbers: 50
```

Automatic Type Conversion JavaScript

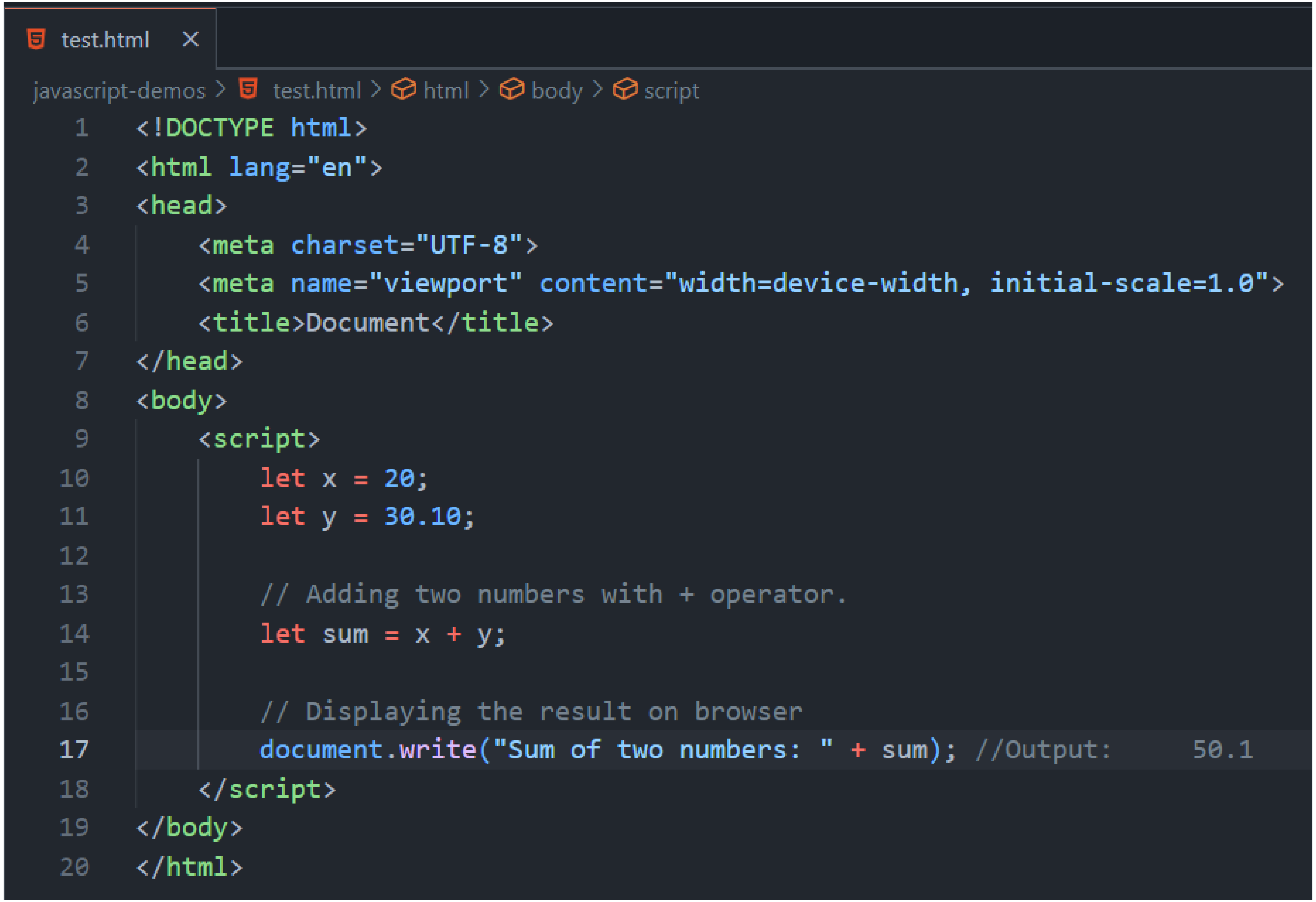
- JavaScript automatically perform type conversion when working with mathematical operators.
- When we use addition or other mathematical operators, JavaScript automatically converts different values, such as integer and float to appropriate type.
- The addition operator (+) also performs string concatenation. It joins two or more strings into a single string
  - ✓ example, "Java" + "Script" gives the result "JavaScript" string.
- Combining Strings and Numbers. JavaScript will convert numbers into strings, depending on the following rule
  - ✓ If the first operand (or value) in expression is a string, JavaScript converts any number into string in the expression and then the plus operator (+) concatenates the strings into a single string. example, "4" + 4 + 4 returns a string "444"
  - ✓ If the first two or more values in expression are numbers, and the rest of the expression contains a string, JavaScript first performs the numeric part of the expression and then converts the result into a string. Example, (4 + 4 + "4") returns the string "84"
  - ✓ NOTE, Plus (+) operator does not convert string values to numeric values

Automatic Type Conversion JavaScript…

- In case of Subtraction...If both of the values in the expression are strings, JavaScript try to convert strings into numbers. If this conversion is not possible, the value of NaN (not a number) is returned

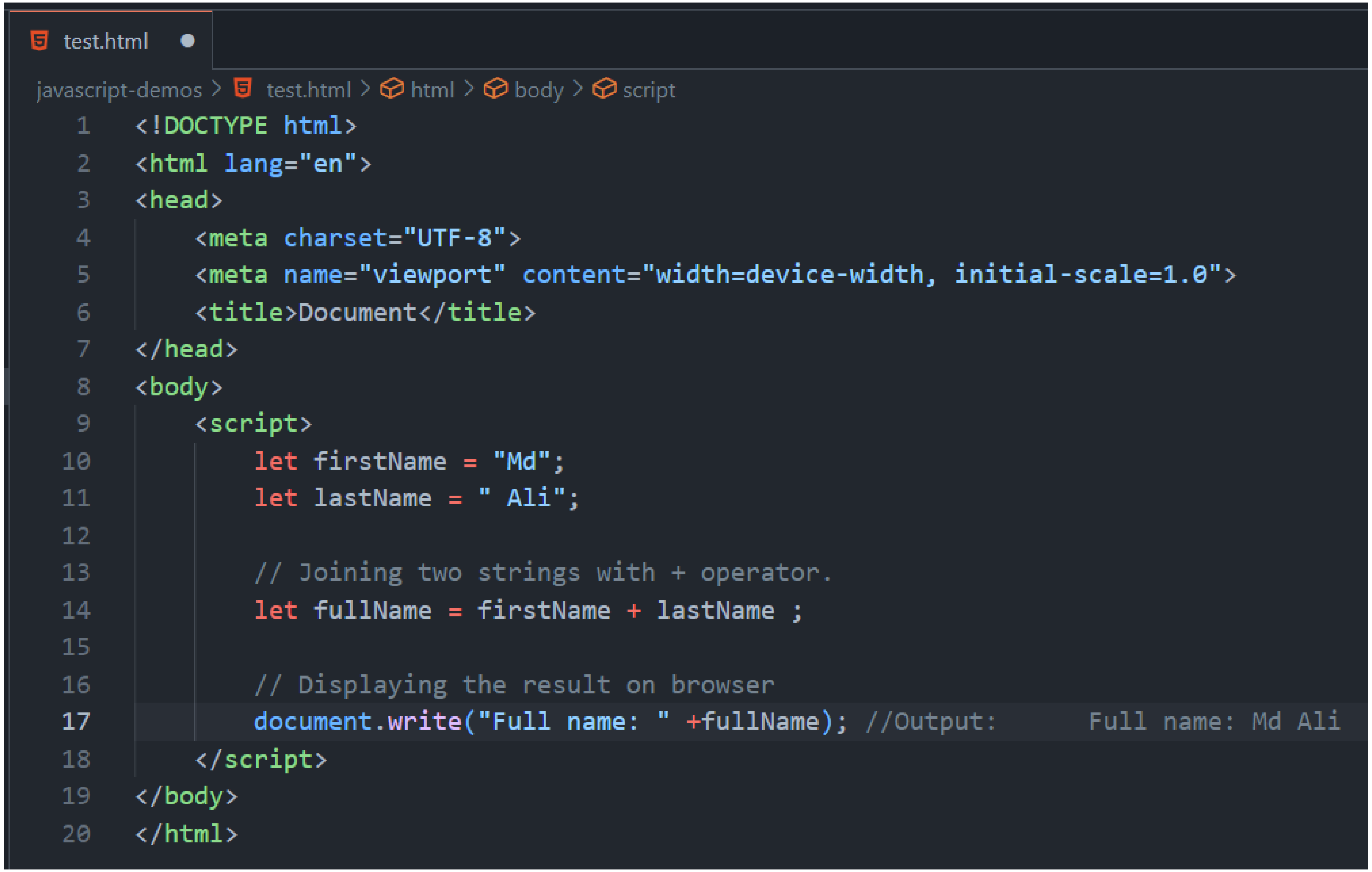
- When a number and a string in the expression is being multiplied, the string is converted to a number first and then multiplied by the other number. If this conversion is not possible, the value of NaN is returned.

- When a number and a string in the expression is being divided, the string is converted to a number first and then divide by the other number. If this conversion is not possible, the value of NaN is returned.



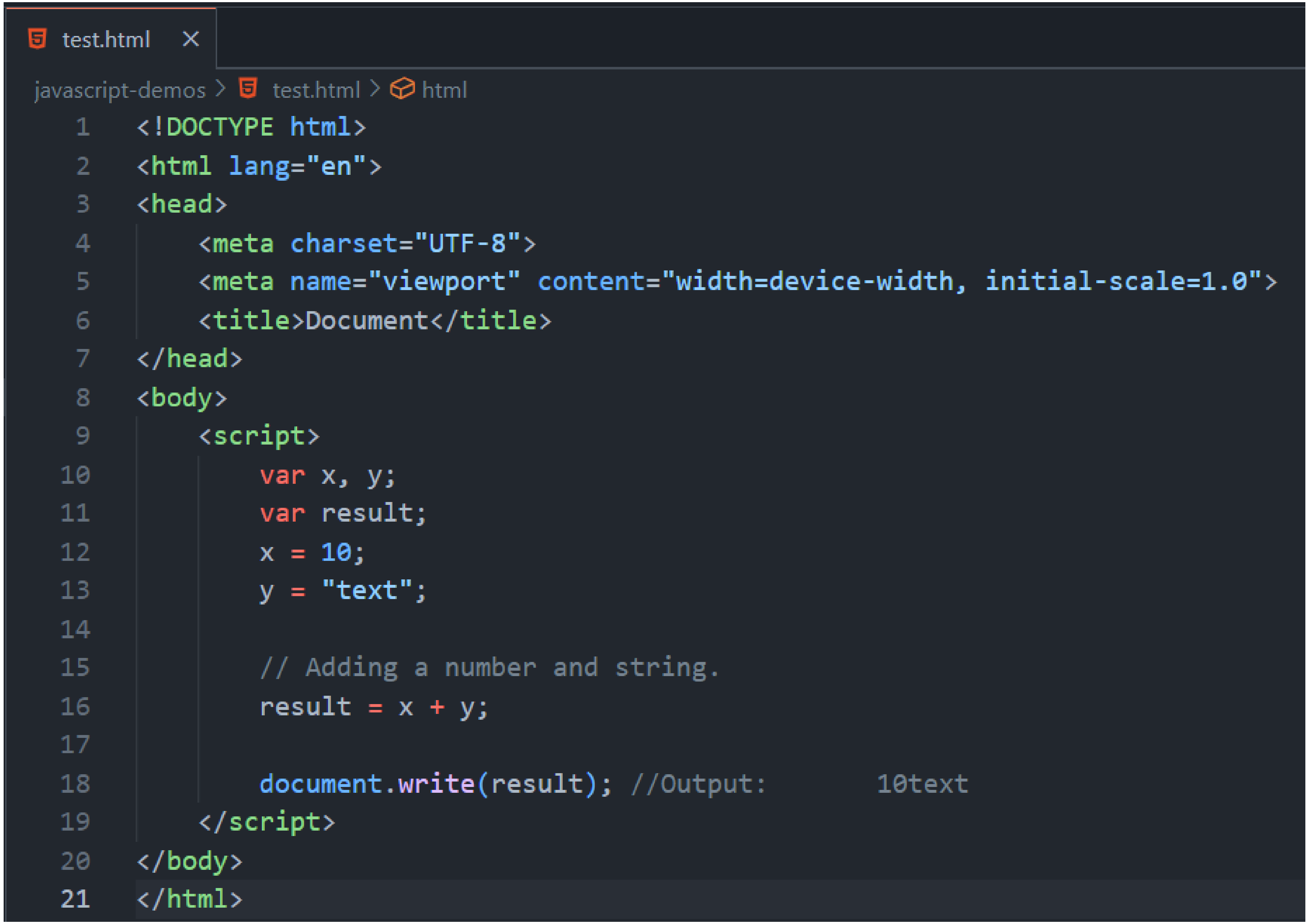
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        let x = 20;
        let y = 30.10;

        // Adding two numbers with + operator.
        let sum = x + y;

        // Displaying the result on browser
        document.write("Sum of two numbers: " + sum); //Output:     50.1
    </script>
</body>
</html>
```

String Concatenation using (+) Operator

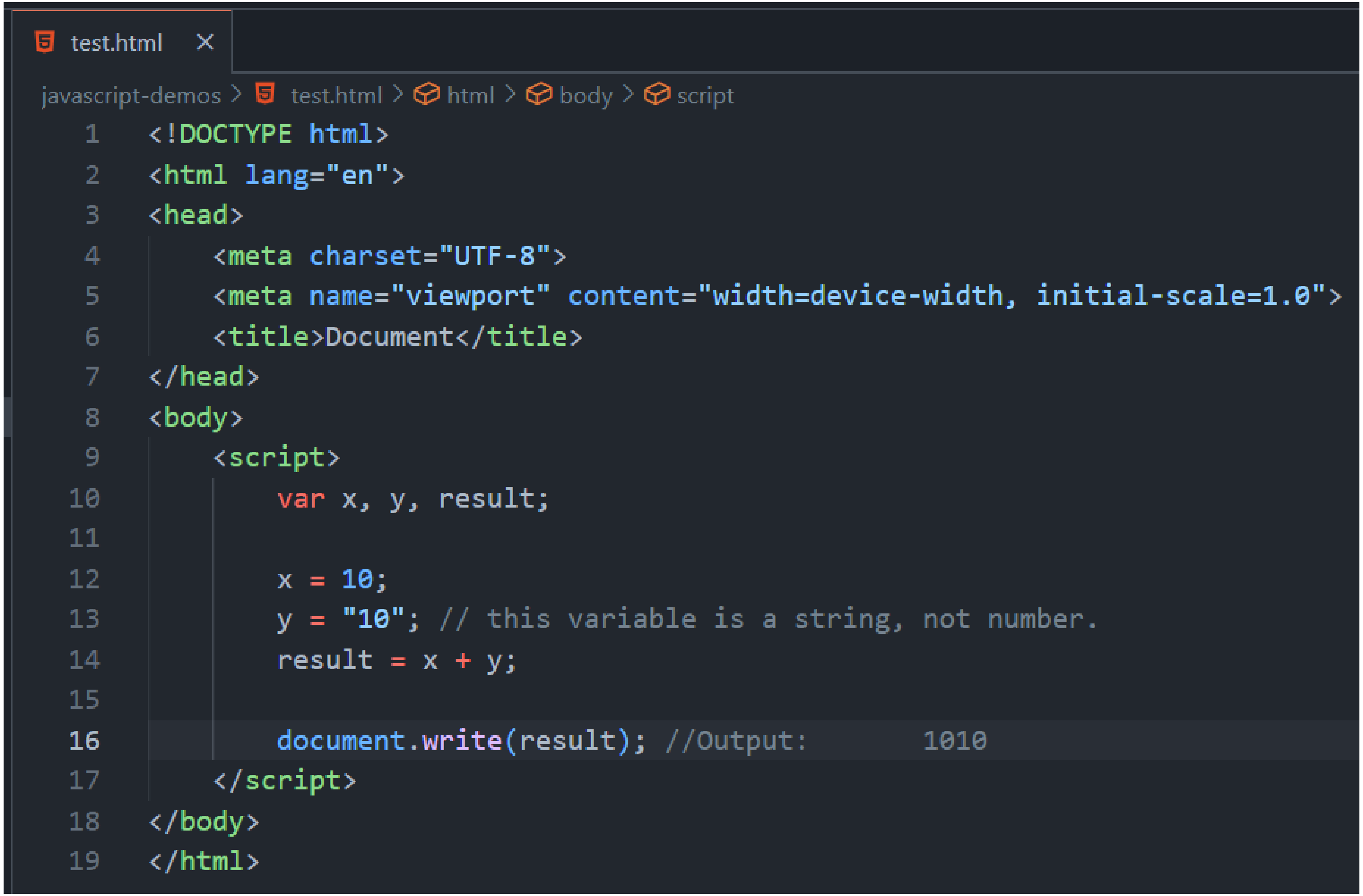


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        let firstName = "Md";
        let lastName = " Ali";

        // Joining two strings with + operator.
        let fullName = firstName + lastName ;

        // Displaying the result on browser
        document.write("Full name: " +fullName); //Output:      Full name: Md Ali
    </script>
</body>
</html>
```

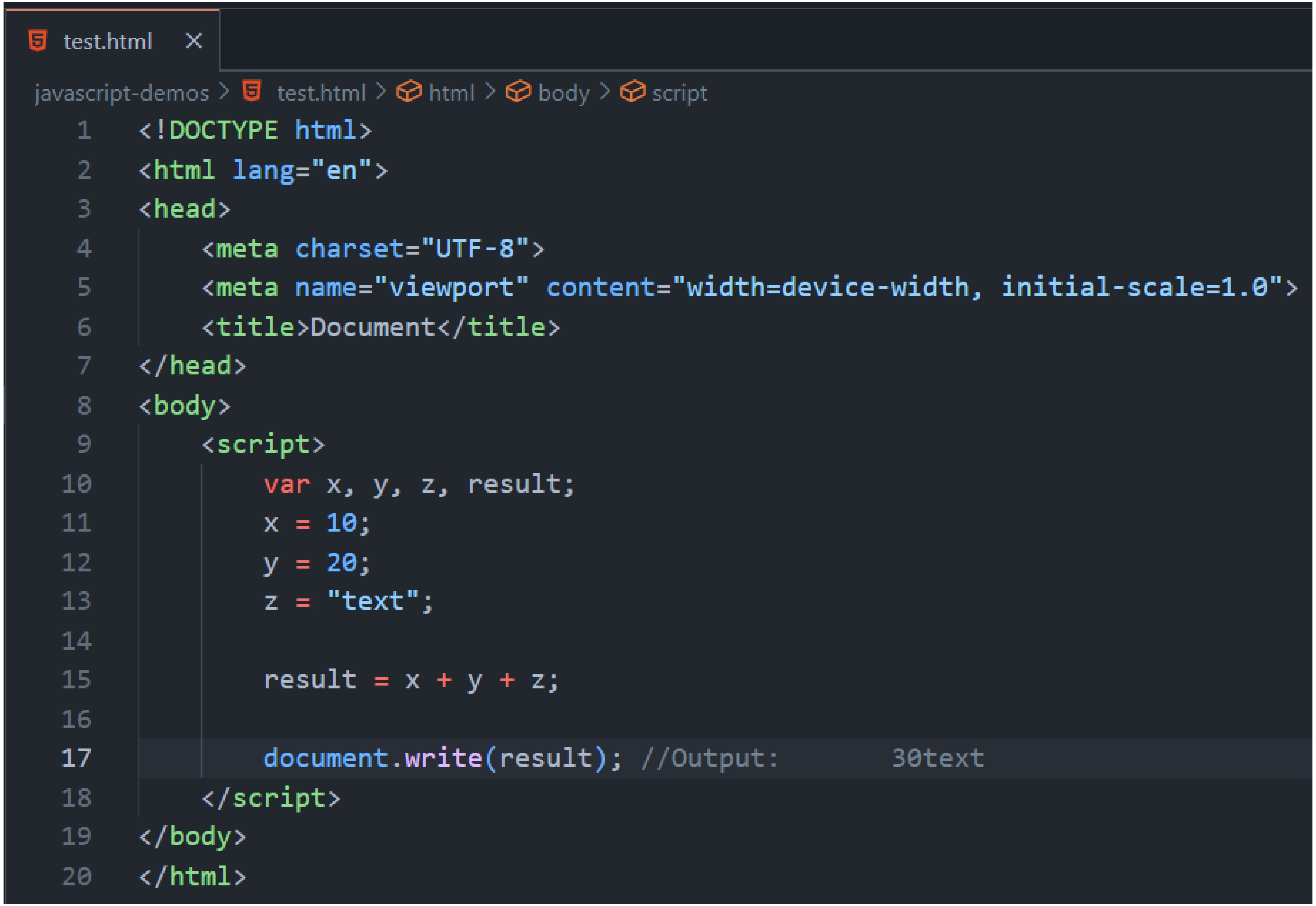
Combining Strings and Numbers



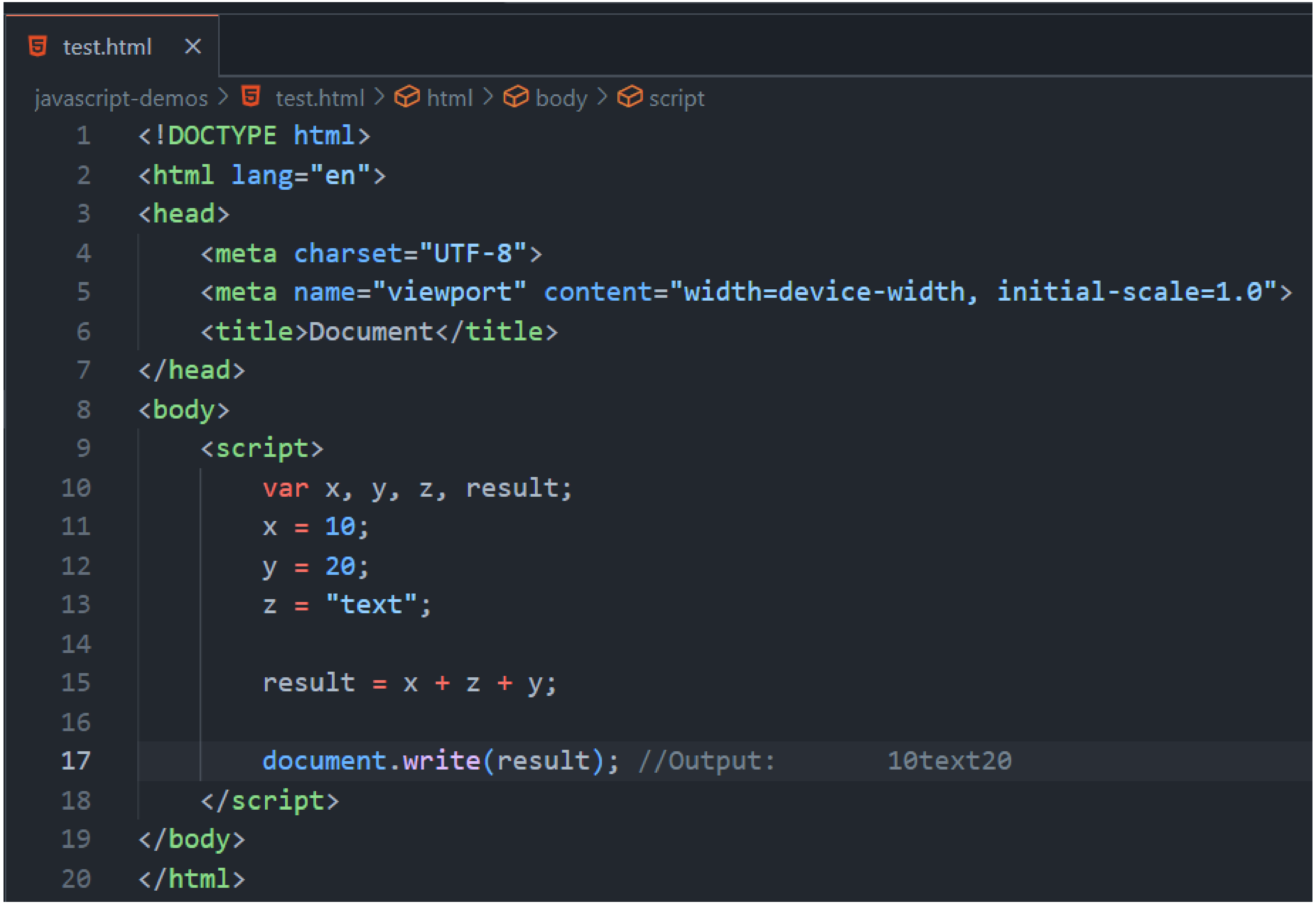
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y;
        var result;
        x = 10;
        y = "text";

        // Adding a number and string.
        result = x + y;

        document.write(result); //Output:       10text
    </script>
</body>
</html>
```



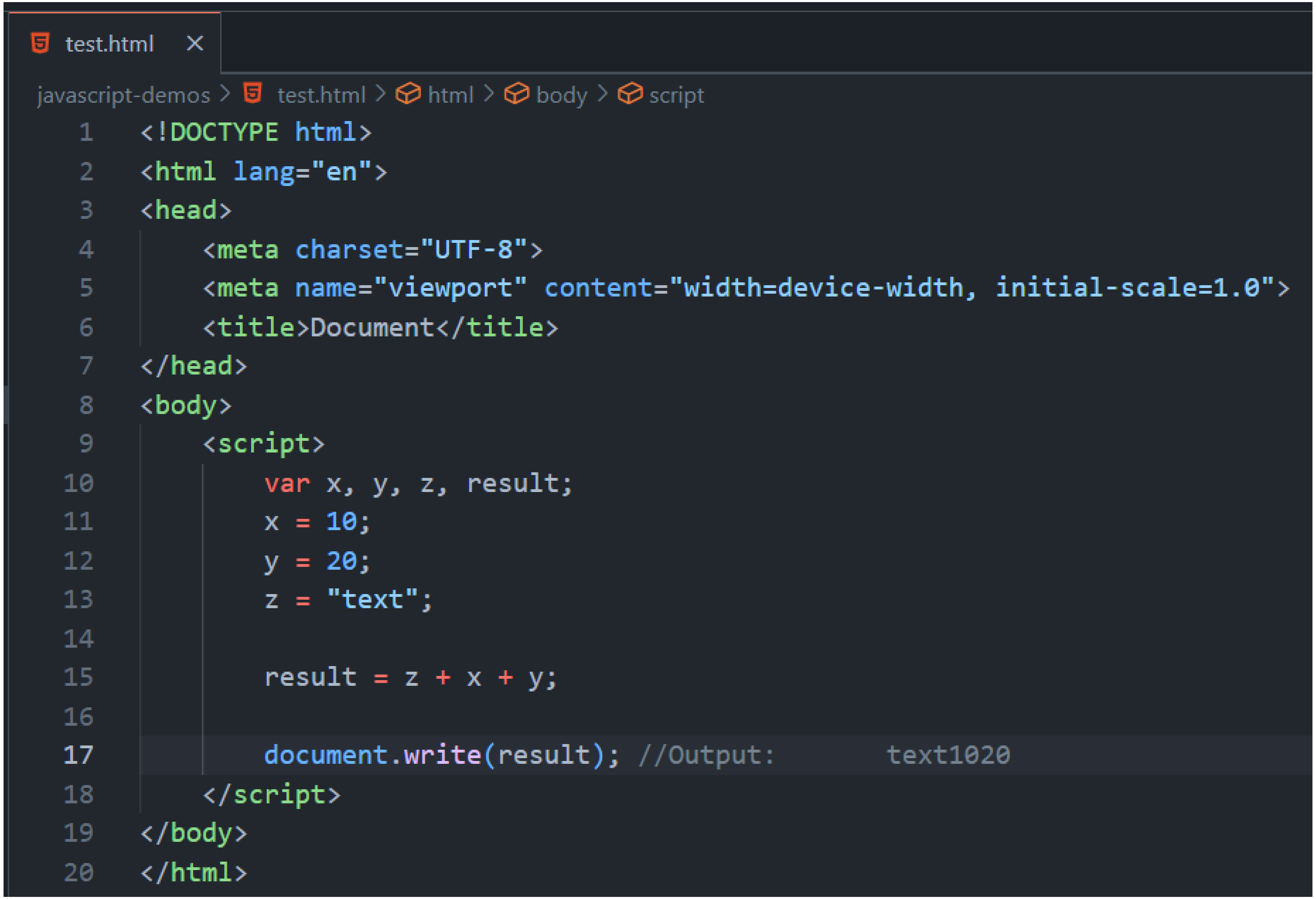
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, result;

        x = 10;
        y = "10"; // this variable is a string, not number.
        result = x + y;

        document.write(result); //Output:       1010
    </script>
</body>
</html>
```



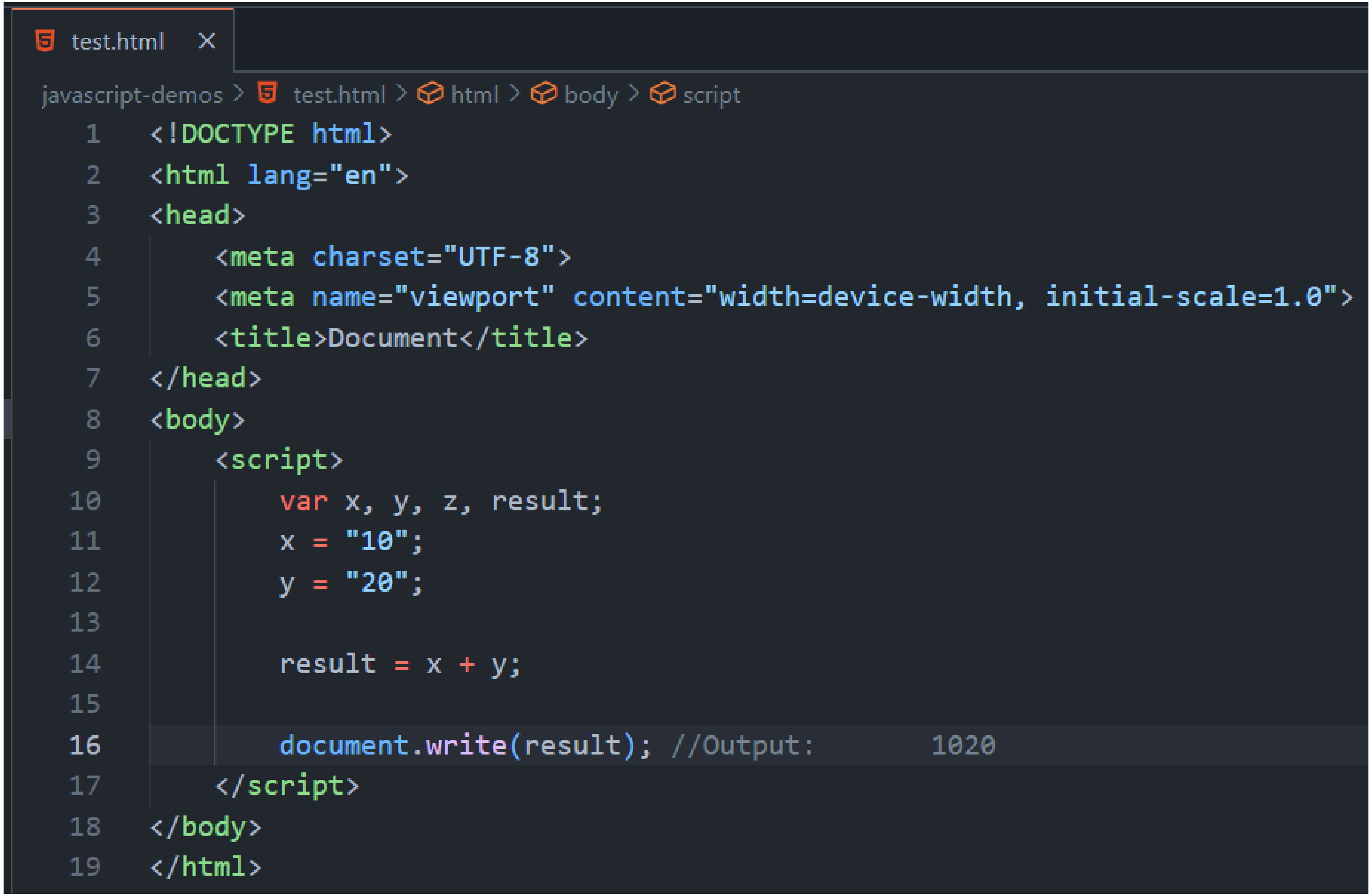
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = 10;
        y = 20;
        z = "text";

        result = x + y + z;

        document.write(result); //Output:       30text
    </script>
</body>
</html>
```



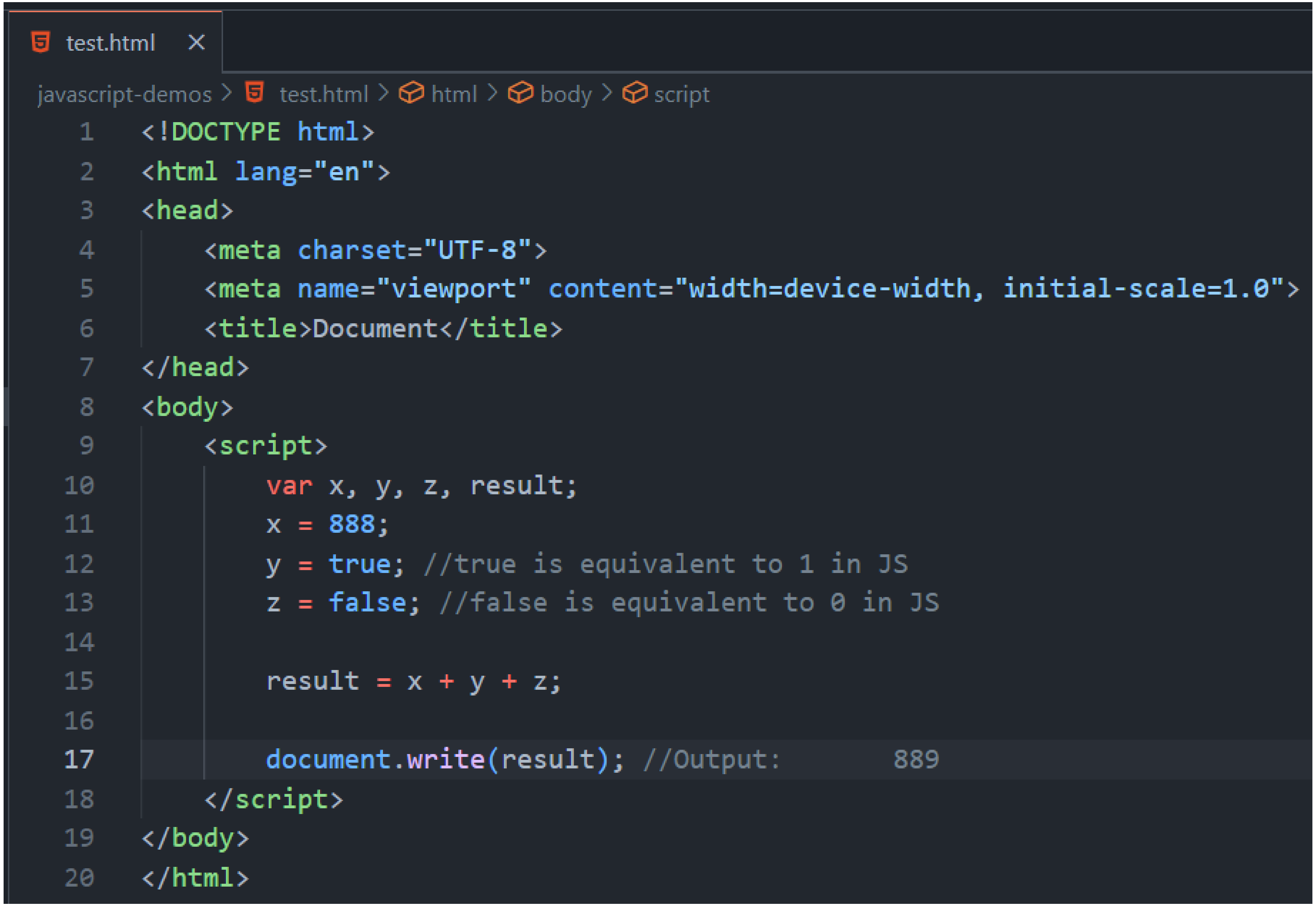
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = 10;
        y = 20;
        z = "text";

        result = x + z + y;

        document.write(result); //Output:       10text20
    </script>
</body>
</html>
```



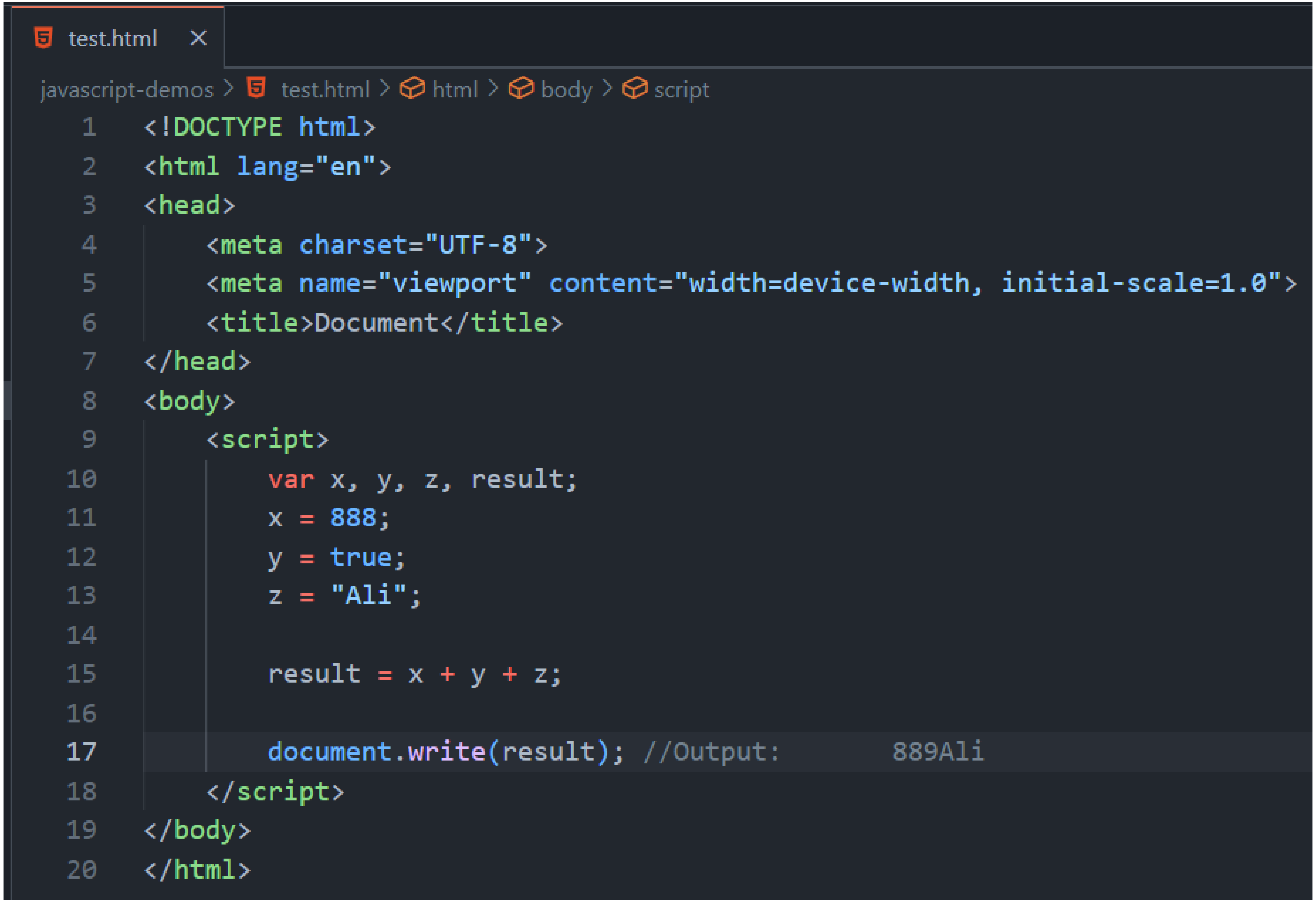
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = 10;
        y = 20;
        z = "text";

        result = z + x + y;

        document.write(result); //Output:       text1020
    </script>
</body>
</html>
```



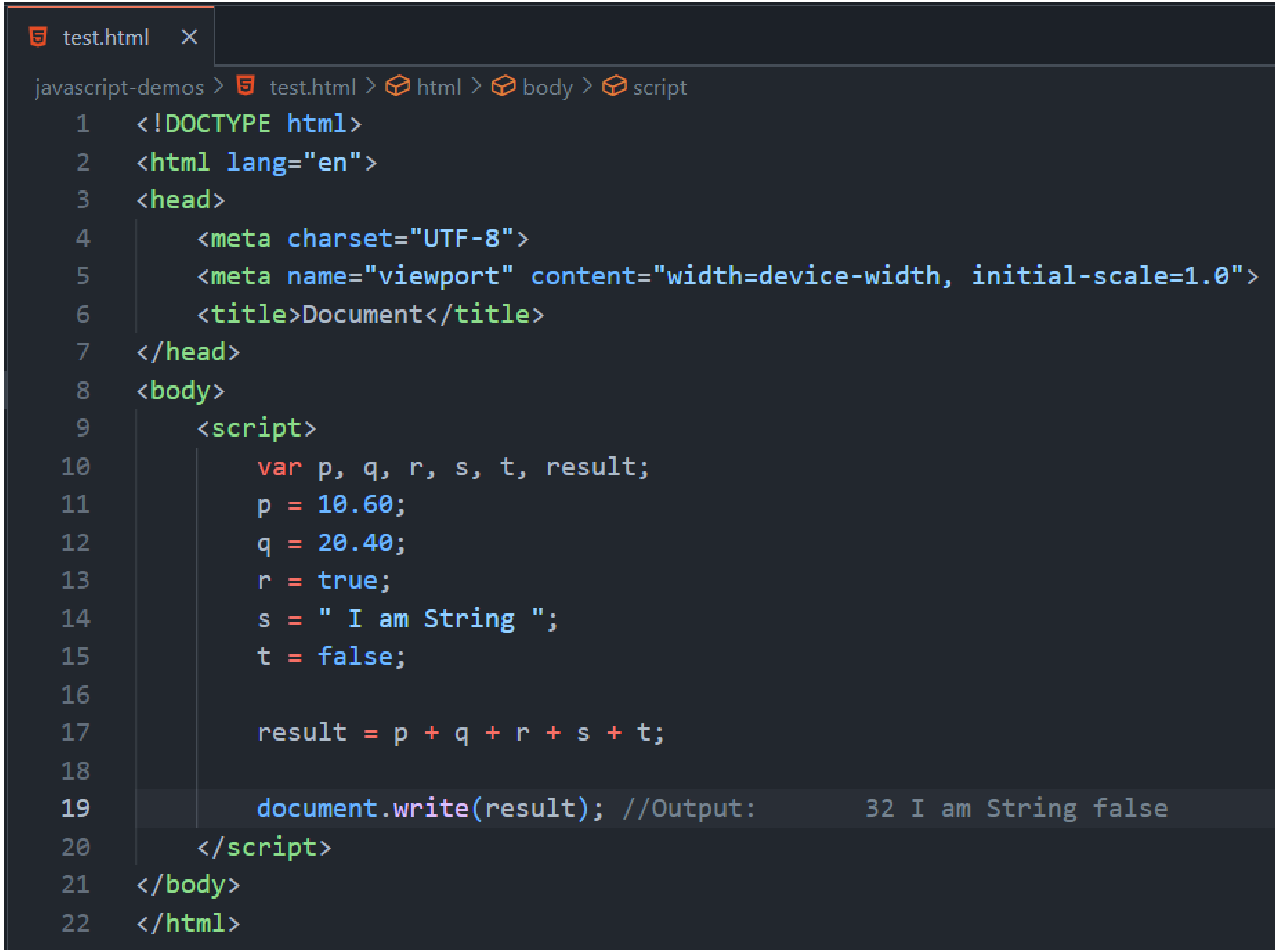
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = "10";
        y = "20";

        result = x + y;

        document.write(result); //Output:       1020
    </script>
</body>
</html>
```



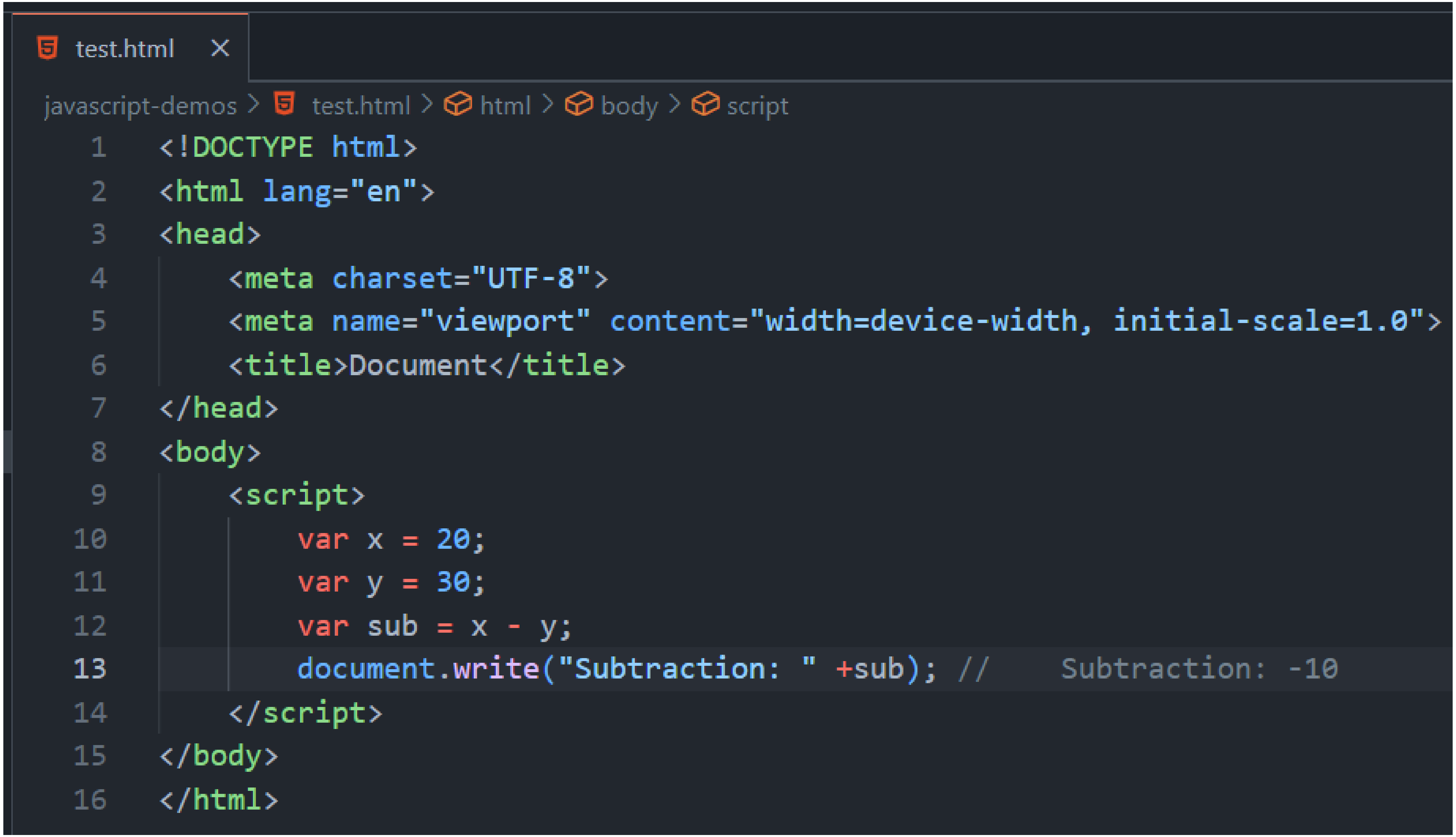
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = 888;
        y = true; //true is equivalent to 1 in JS
        z = false; //false is equivalent to 0 in JS

        result = x + y + z;

        document.write(result); //Output:       889
    </script>
</body>
</html>
```



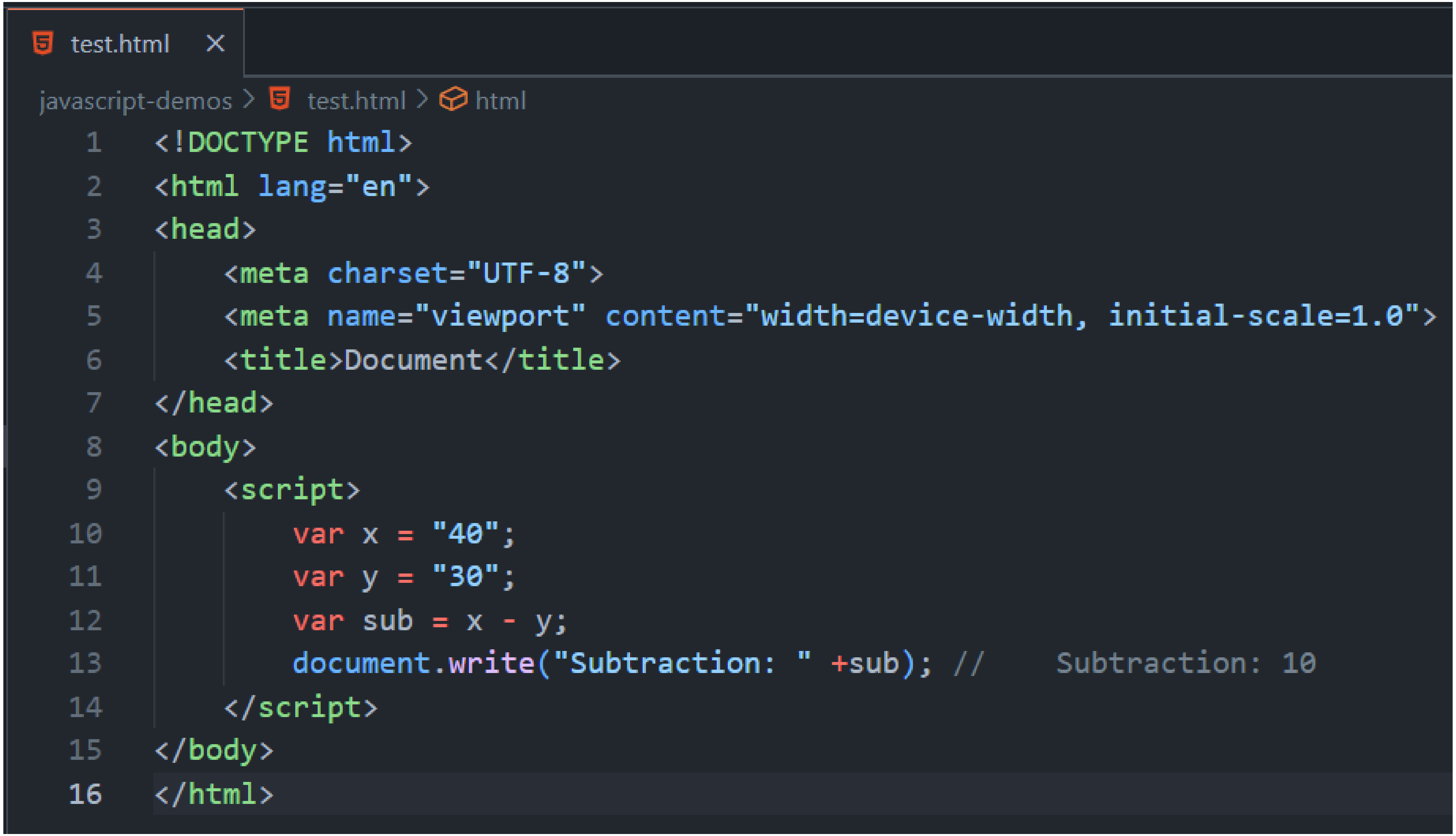
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x, y, z, result;
        x = 888;
        y = true;
        z = "Ali";

        result = x + y + z;

        document.write(result); //Output:        889Ali
    </script>
</body>
</html>
```



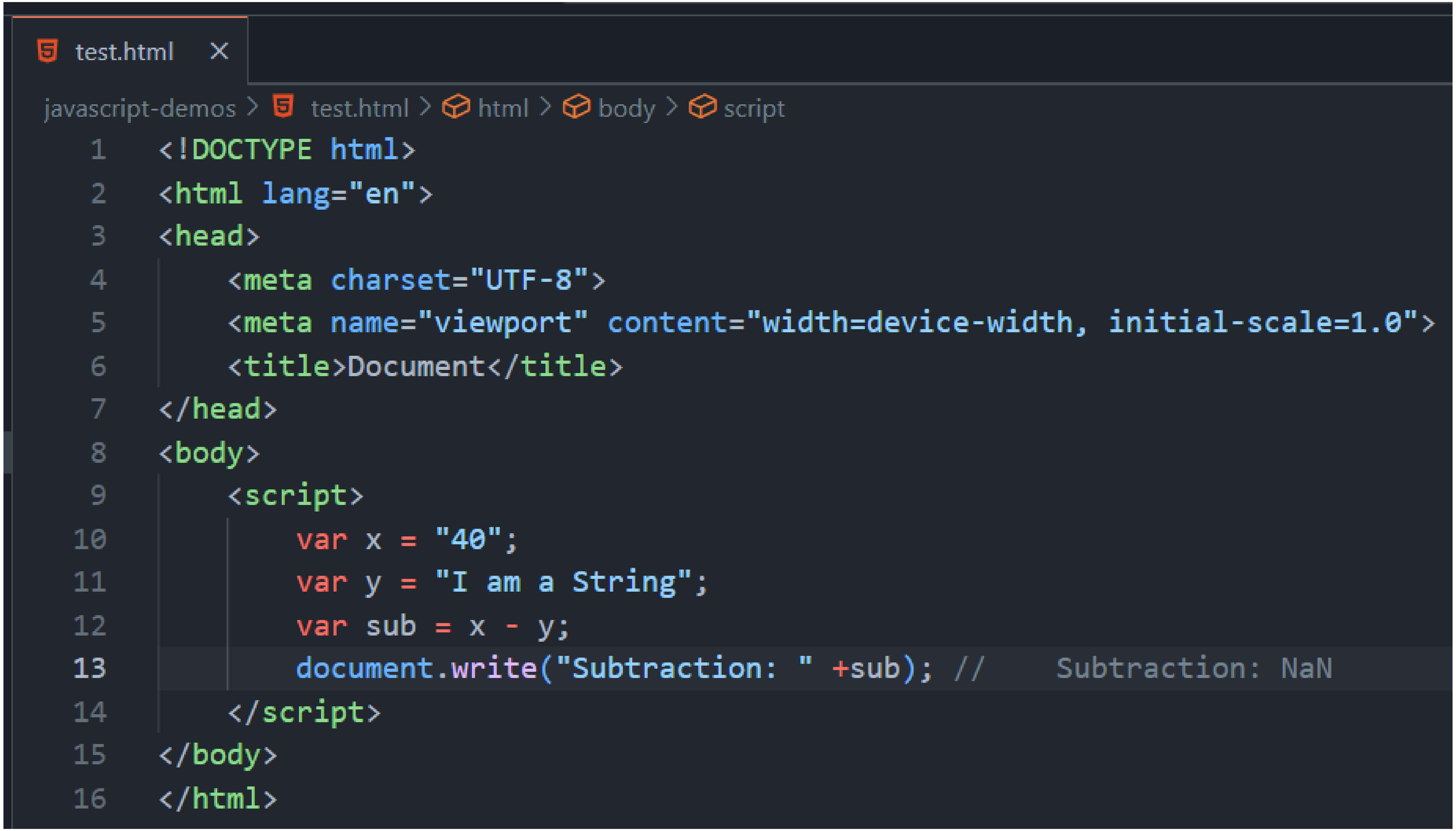
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var p, q, r, s, t, result;
        p = 10.60;
        q = 20.40;
        r = true;
        s = " I am String ";
        t = false;

        result = p + q + r + s + t;

        document.write(result); //Output:       32 I am String false
    </script>
</body>
</html>
```



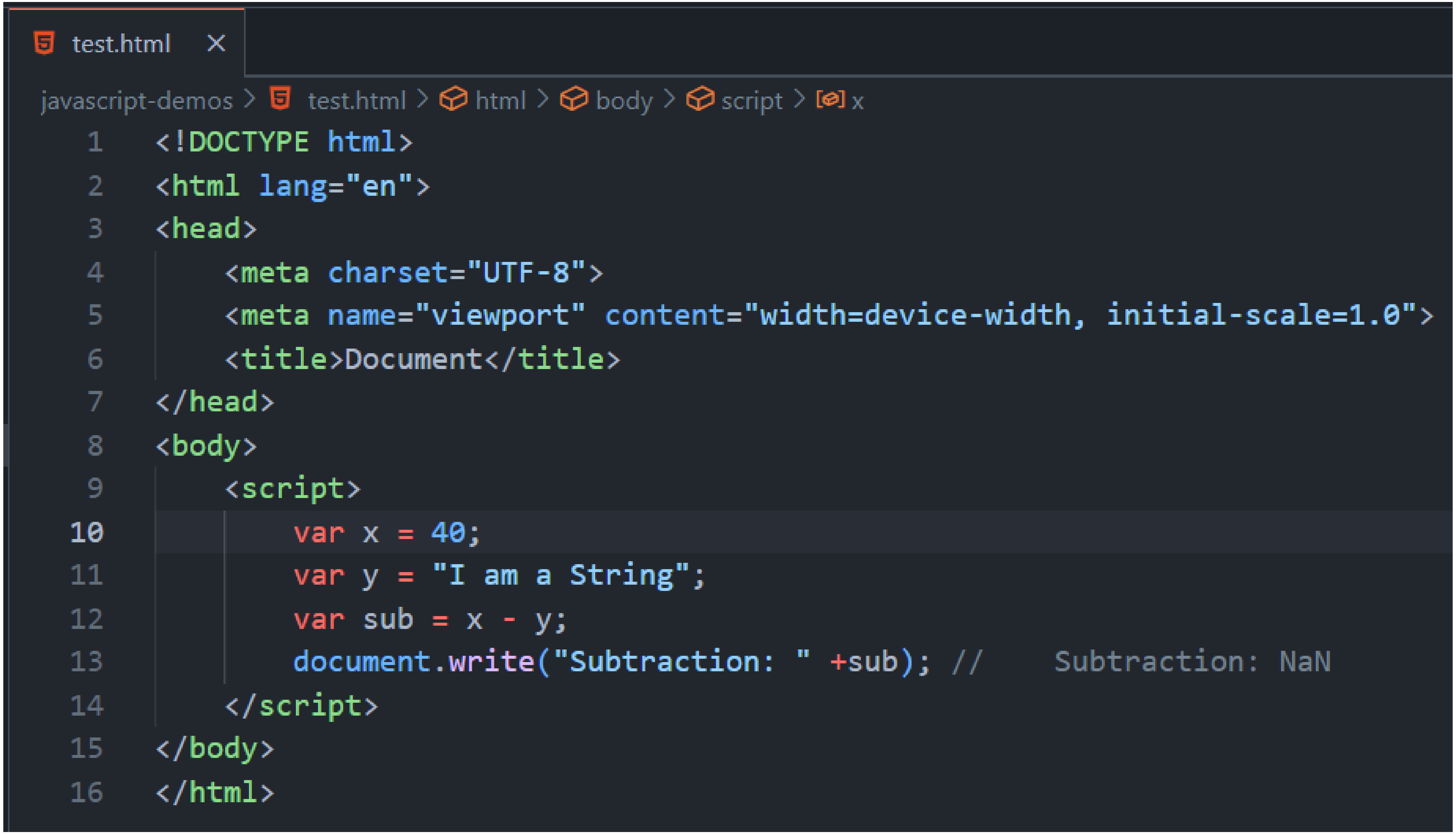
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 20;
        var y = 30;
        var sub = x - y;
        document.write("Subtraction: " +sub); //    Subtraction: -10
    </script>
</body>
</html>
```



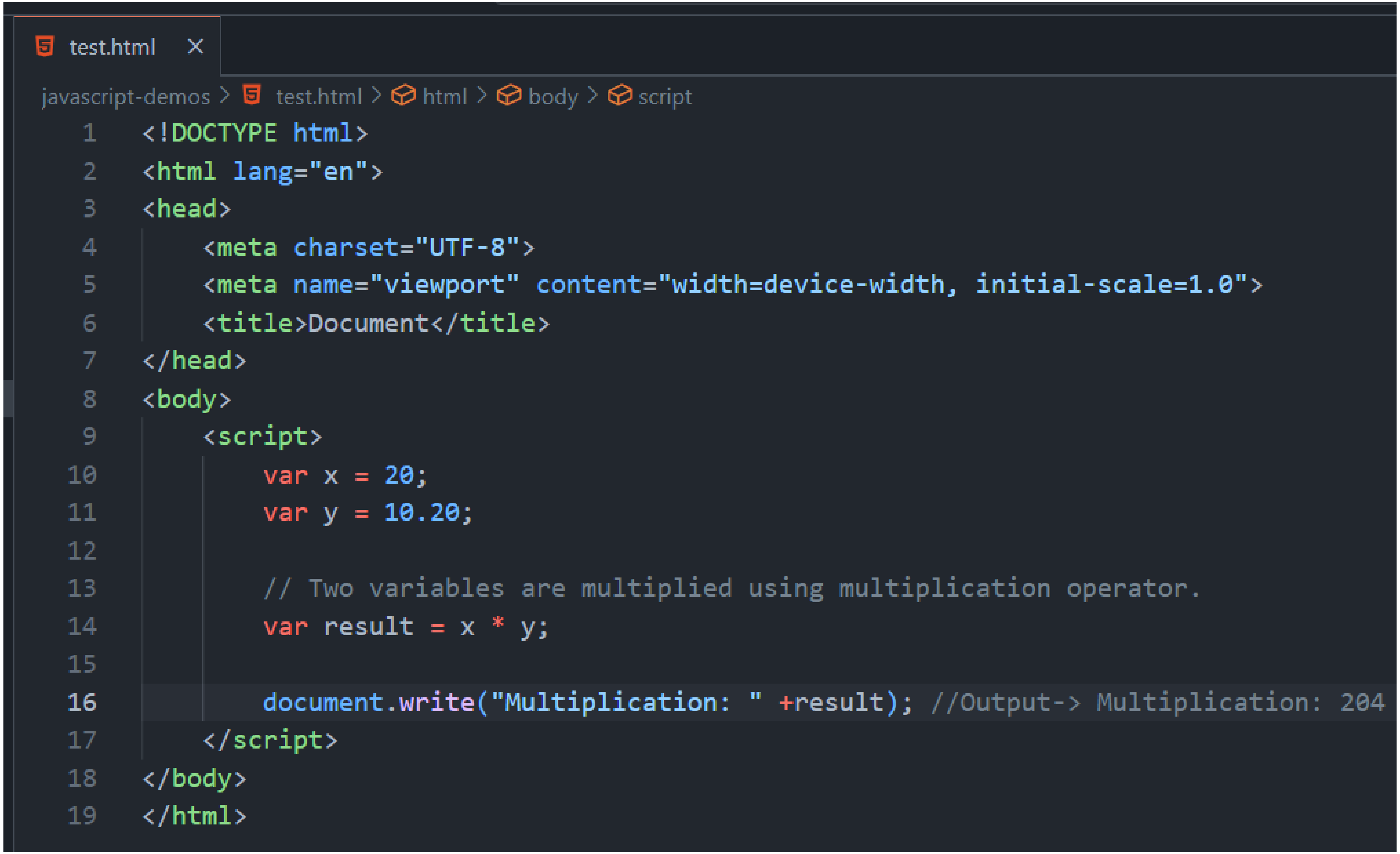
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = "40";
        var y = "30";
        var sub = x - y;
        document.write("Subtraction: " +sub); //    Subtraction: 10
    </script>
</body>
</html>
```



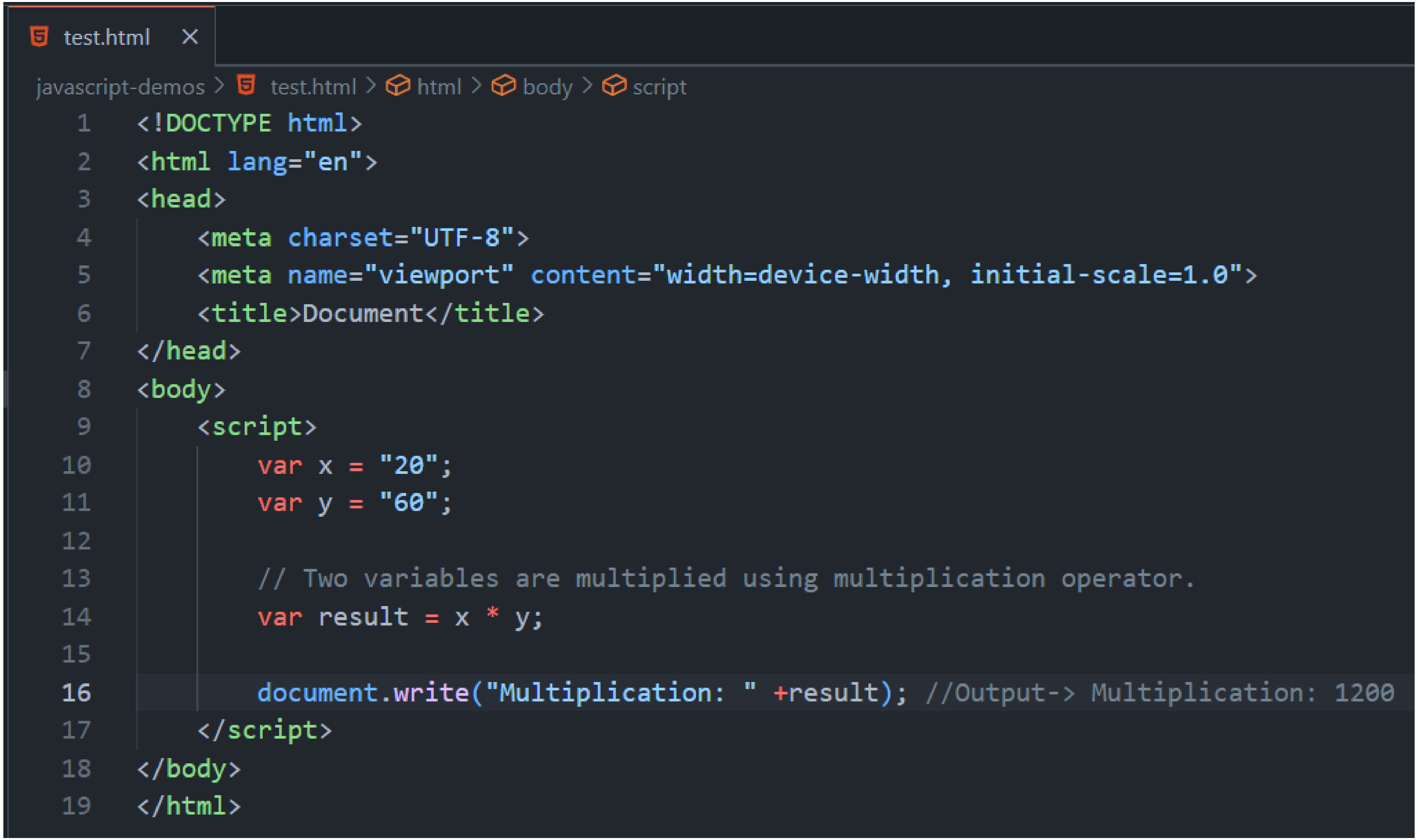
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = "40";
        var y = "I am a String";
        var sub = x - y;
        document.write("Subtraction: " +sub); //    Subtraction: NaN
    </script>
</body>
</html>
```



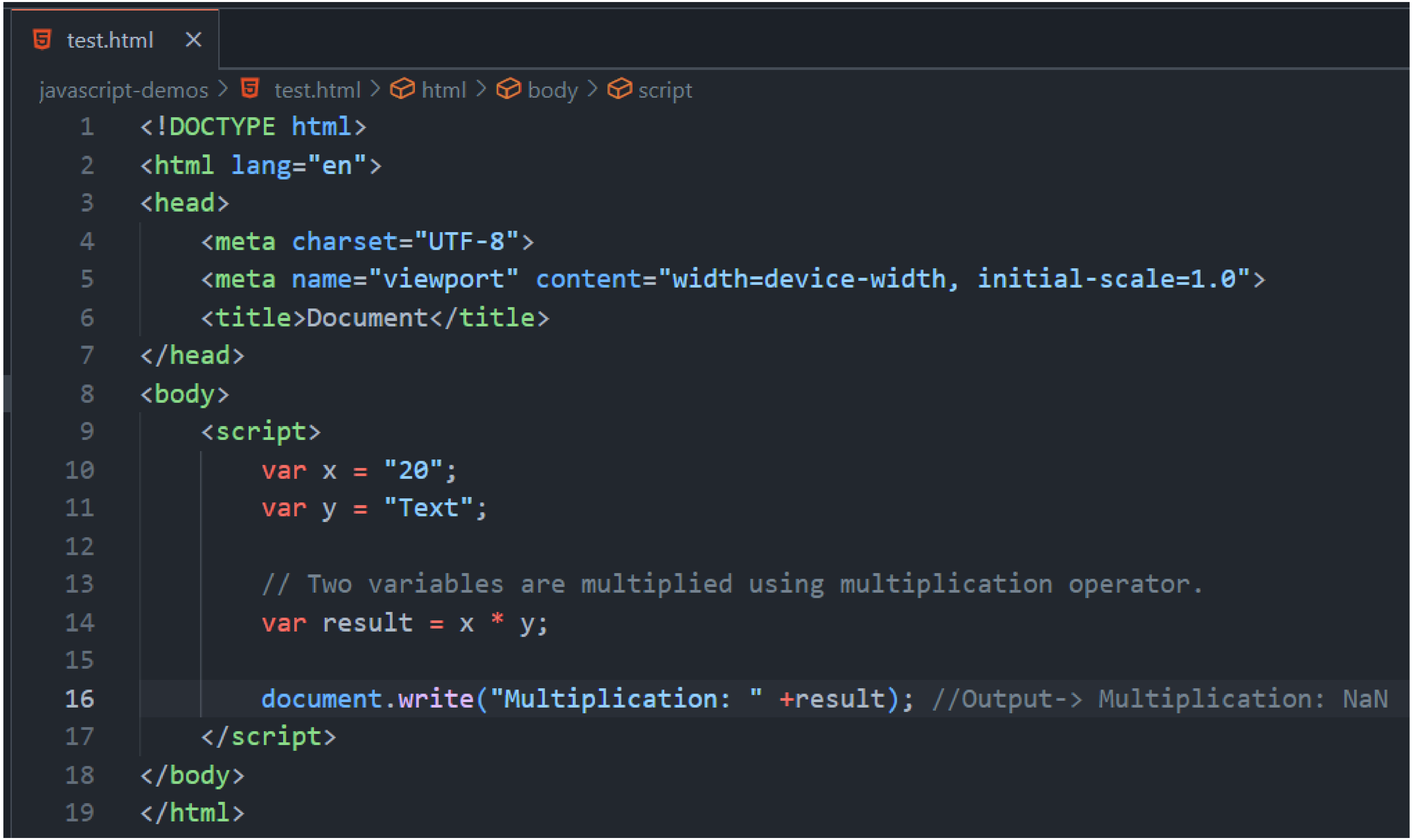
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 40;
        var y = "I am a String";
        var sub = x - y;
        document.write("Subtraction: " +sub); //    Subtraction: NaN
    </script>
</body>
</html>
```



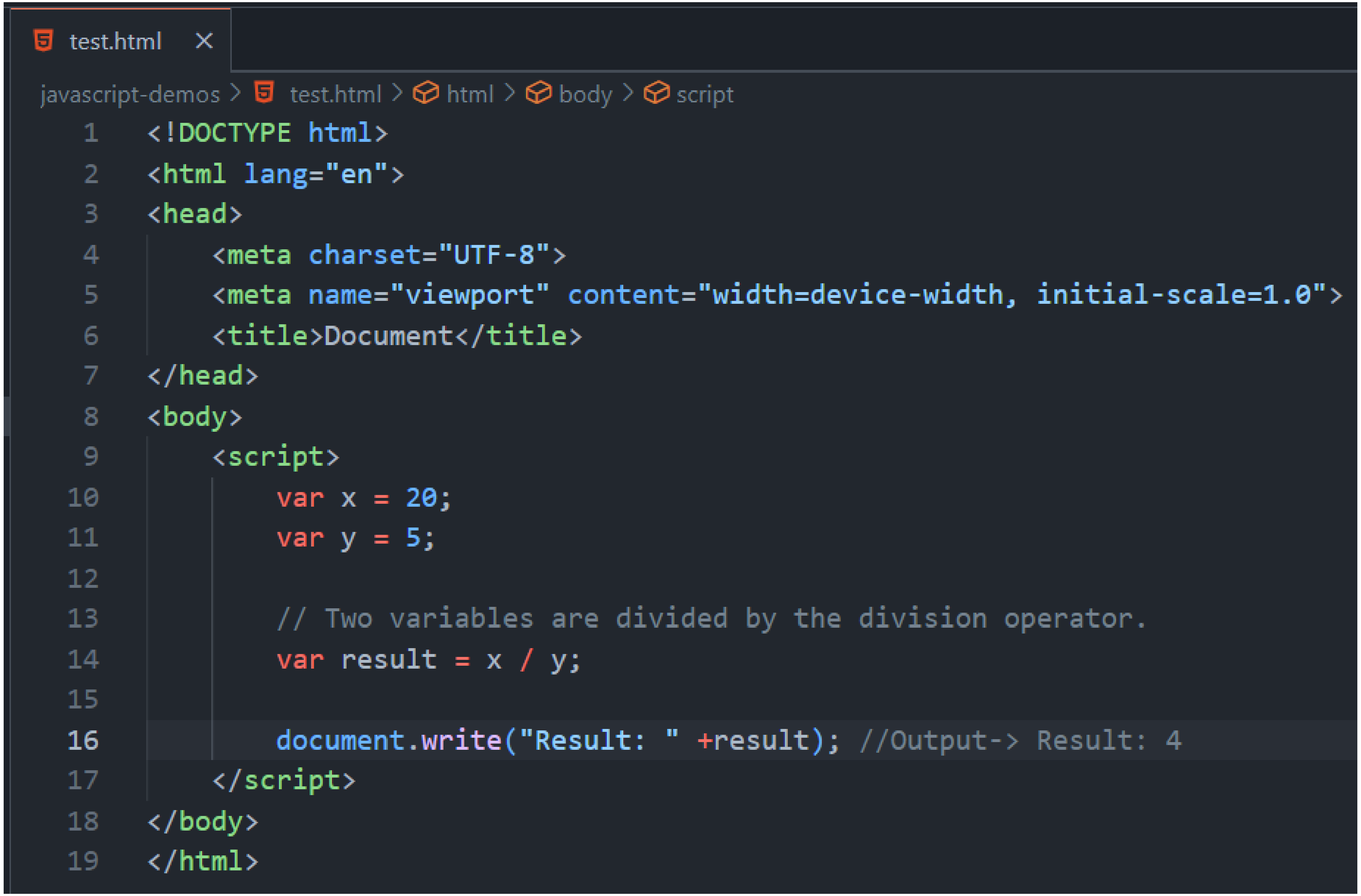
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 20;
        var y = 10.20;

        // Two variables are multiplied using multiplication operator.
        var result = x * y;

        document.write("Multiplication: " +result); //Output-> Multiplication: 204
    </script>
</body>
</html>
```



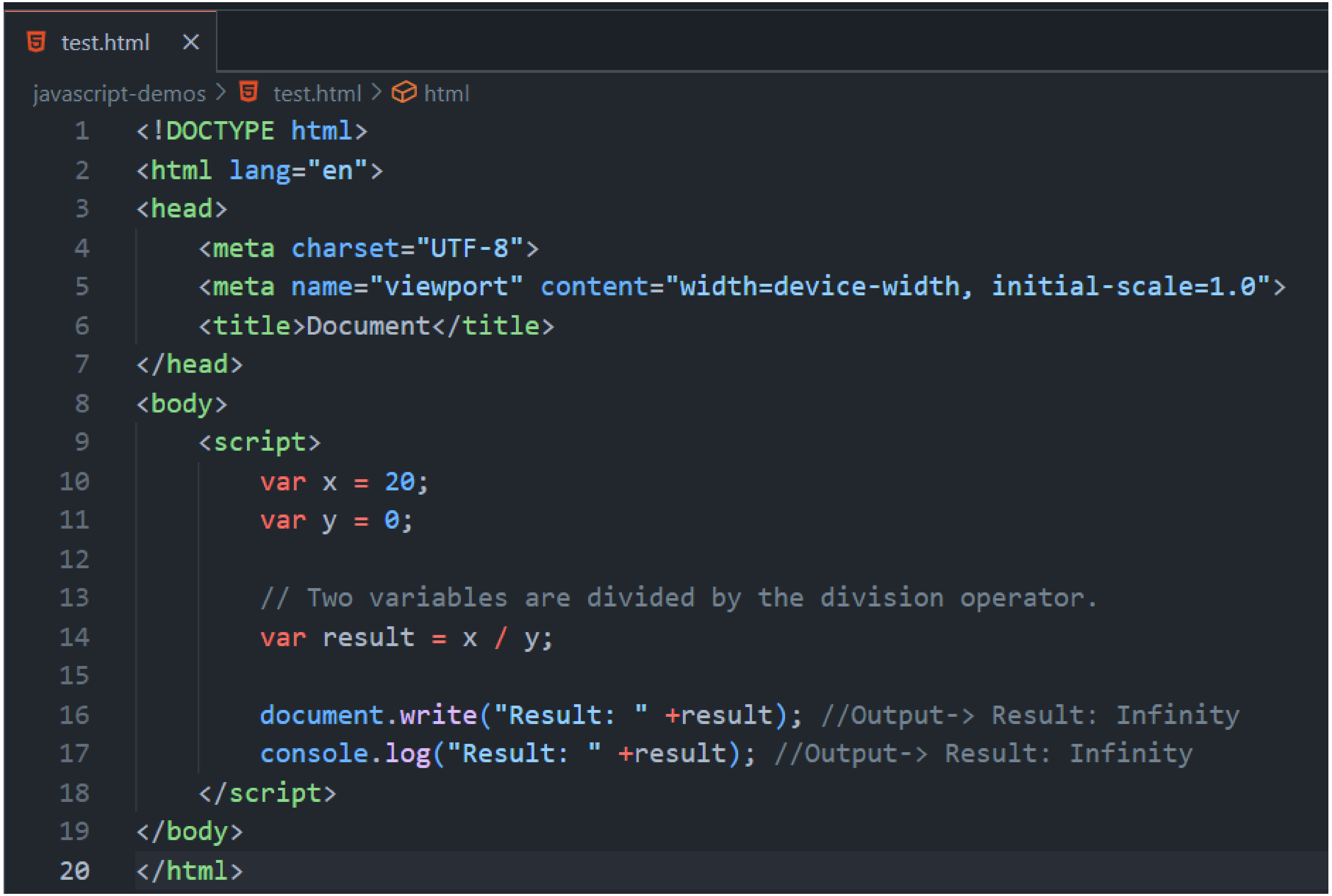
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = "20";
        var y = "60";

        // Two variables are multiplied using multiplication operator.
        var result = x * y;

        document.write("Multiplication: " +result); //Output-> Multiplication: 1200
    </script>
</body>
</html>
```



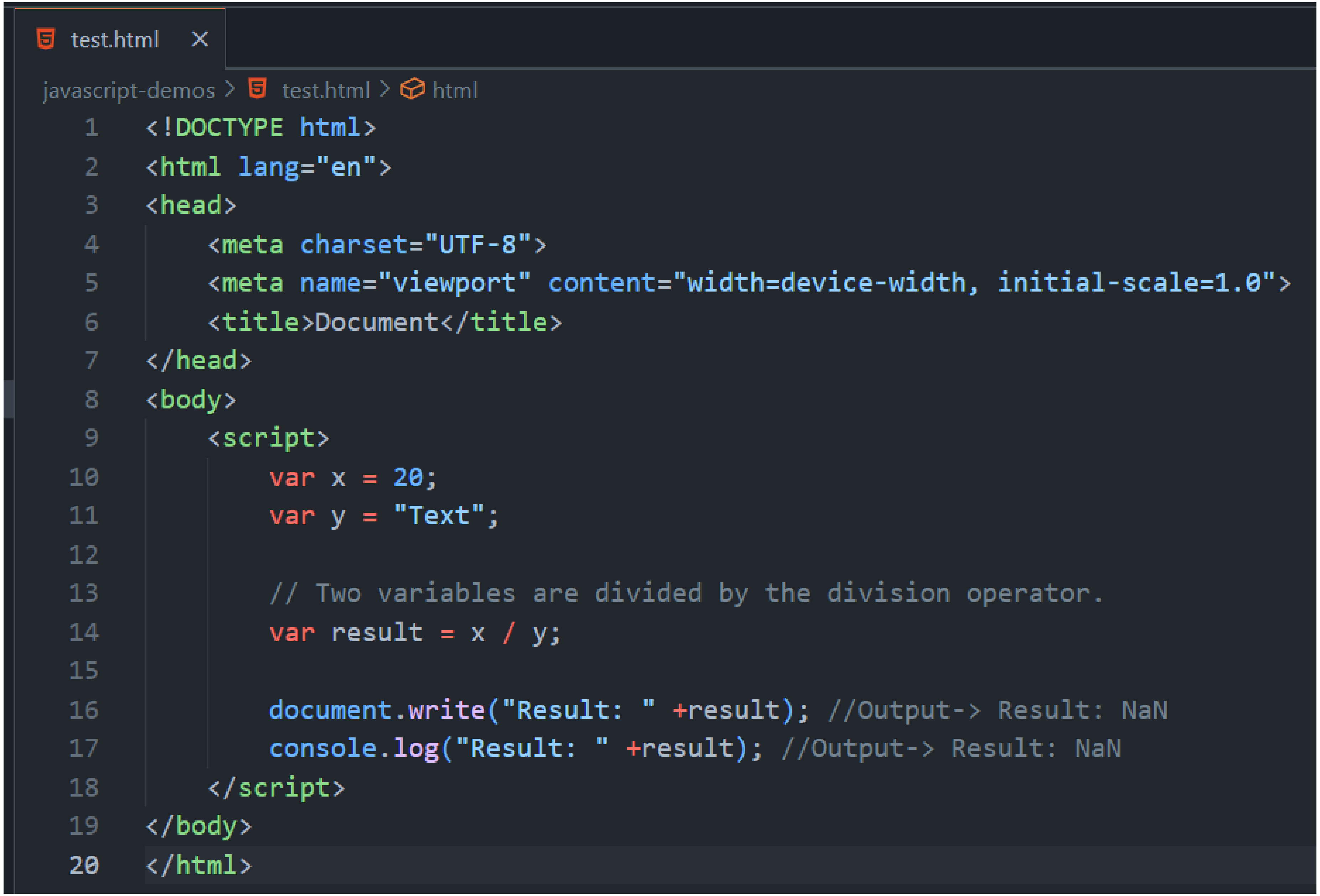
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = "20";
        var y = "Text";

        // Two variables are multiplied using multiplication operator.
        var result = x * y;

        document.write("Multiplication: " +result); //Output-> Multiplication: NaN
    </script>
</body>
</html>
```



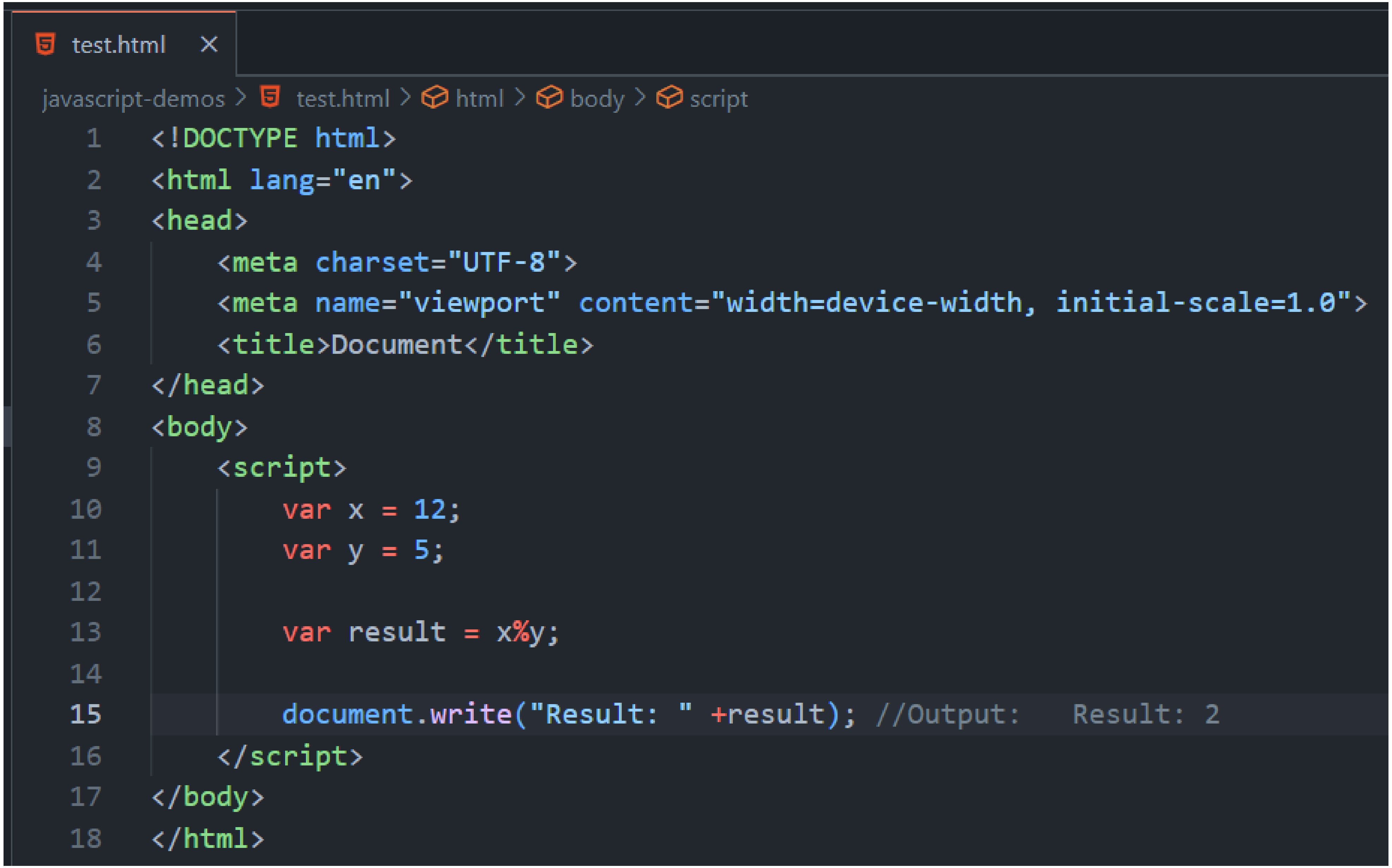
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 20;
        var y = 5;

        // Two variables are divided by the division operator.
        var result = x / y;

        document.write("Result: " +result); //Output-> Result: 4
    </script>
</body>
</html>
```



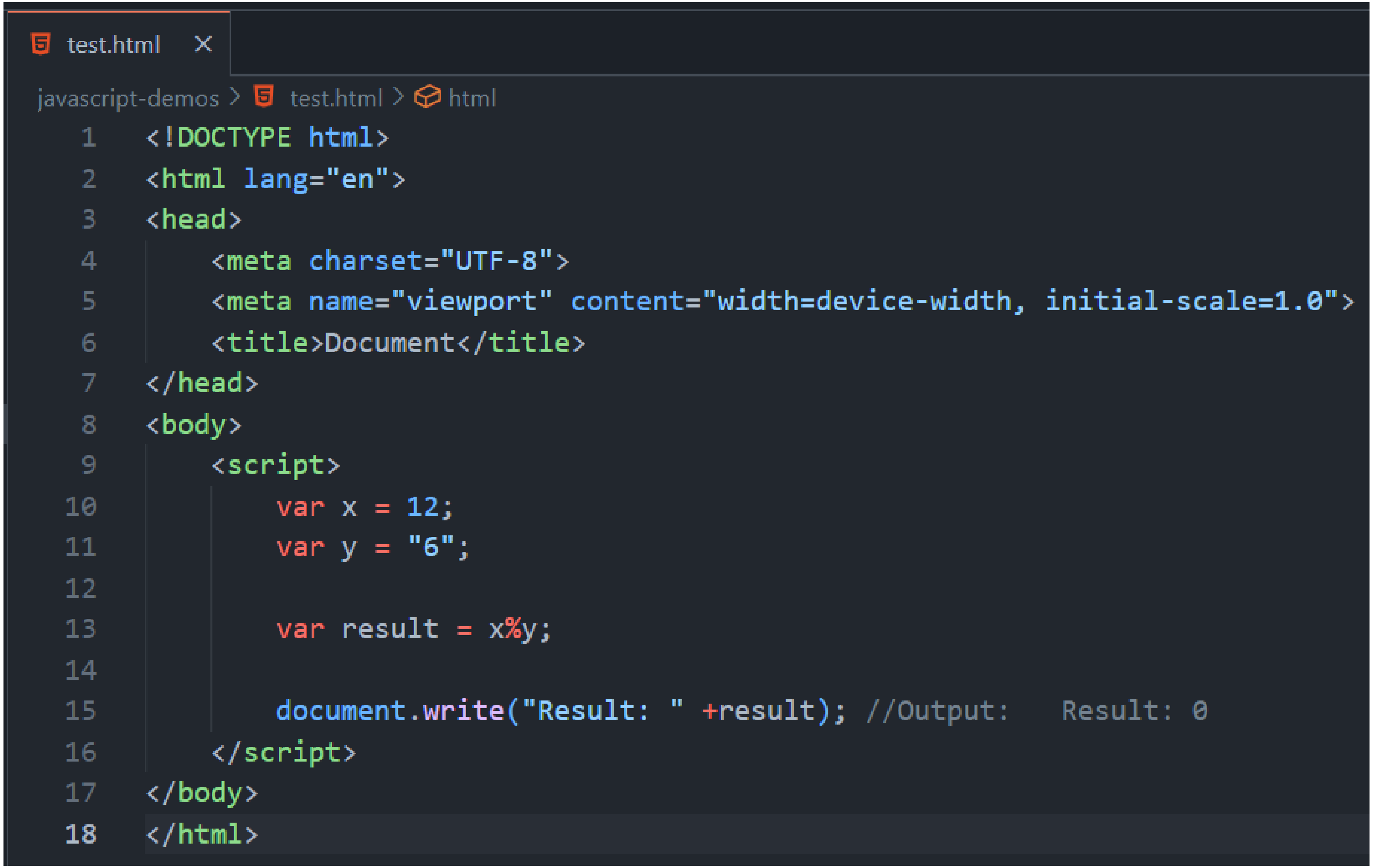
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 20;
        var y = 0;

        // Two variables are divided by the division operator.
        var result = x / y;

        document.write("Result: " +result); //Output-> Result: Infinity
        console.log("Result: " +result); //Output-> Result: Infinity
    </script>
</body>
</html>
```



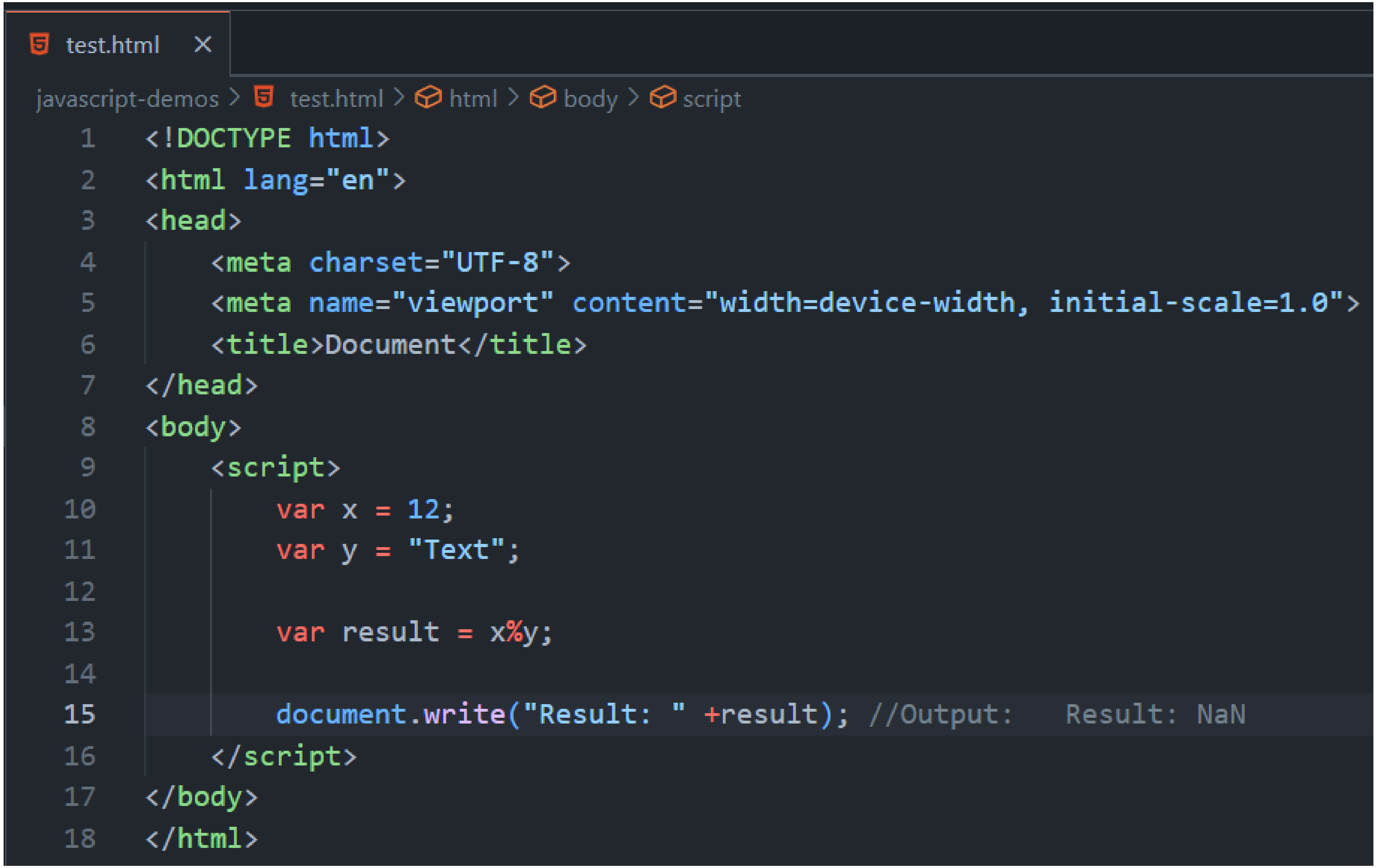
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 20;
        var y = "Text";

        // Two variables are divided by the division operator.
        var result = x / y;

        document.write("Result: " +result); //Output-> Result: NaN
        console.log("Result: " +result); //Output-> Result: NaN
    </script>
</body>
</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>
</head>
<body>
    <script>
        var x = 12;
        var y = 5;

        var result = x%y;

        document.write("Result: " +result); //Output:   Result: 2
    </script>
</body>
</html>
```

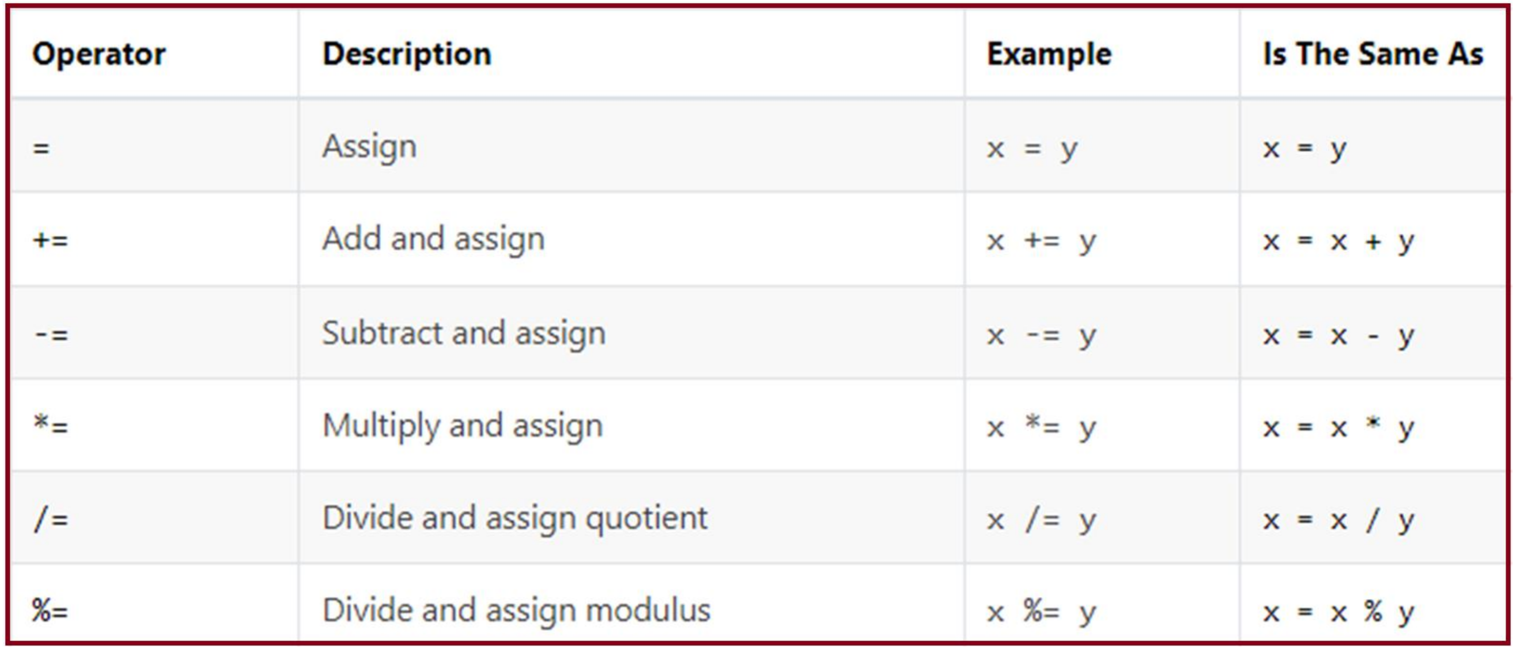


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 12;
        var y = "6";

        var result = x%y;

        document.write("Result: " +result); //Output:   Result: 0
    </script>
</body>
</html>
```

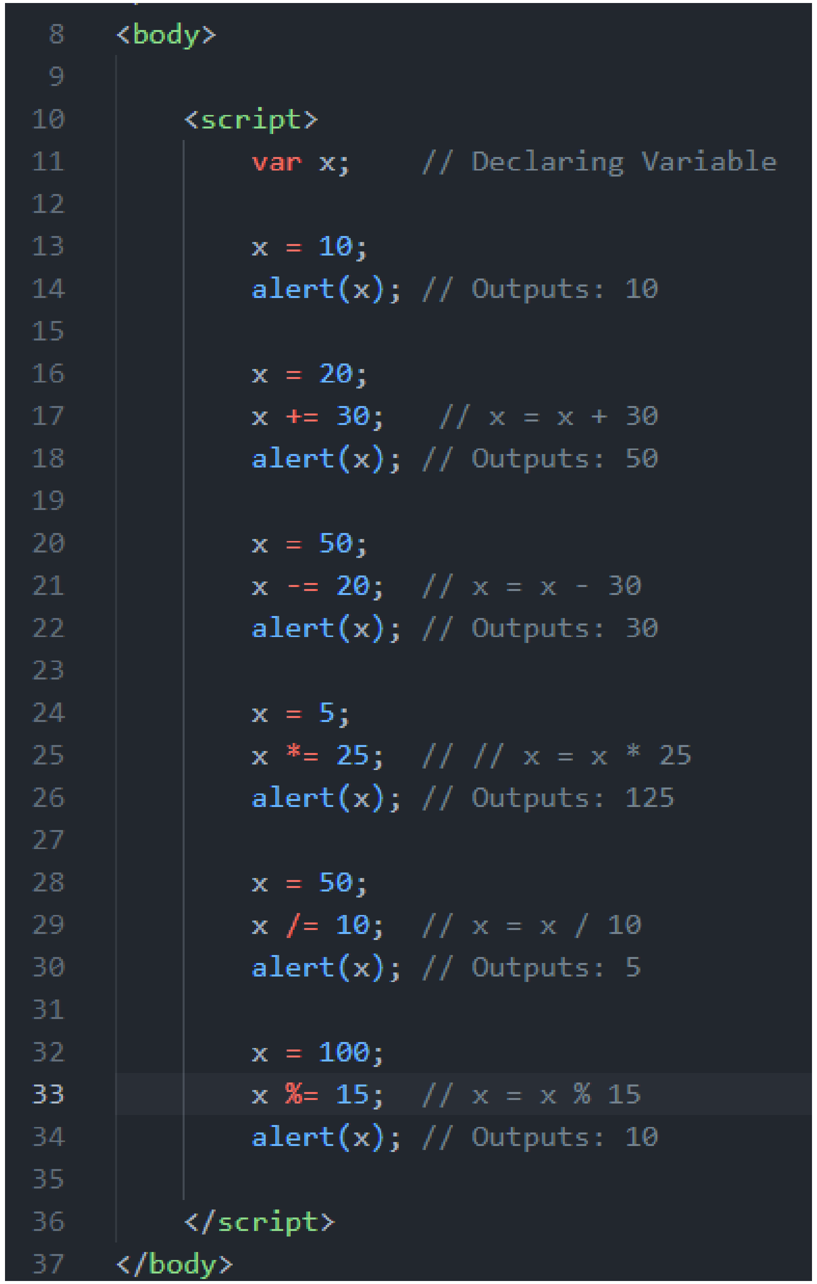


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        var x = 12;
        var y = "Text";

        var result = x%y;

        document.write("Result: " +result); //Output:   Result: NaN
    </script>
</body>
</html>
```

JavaScript Assignment Operators

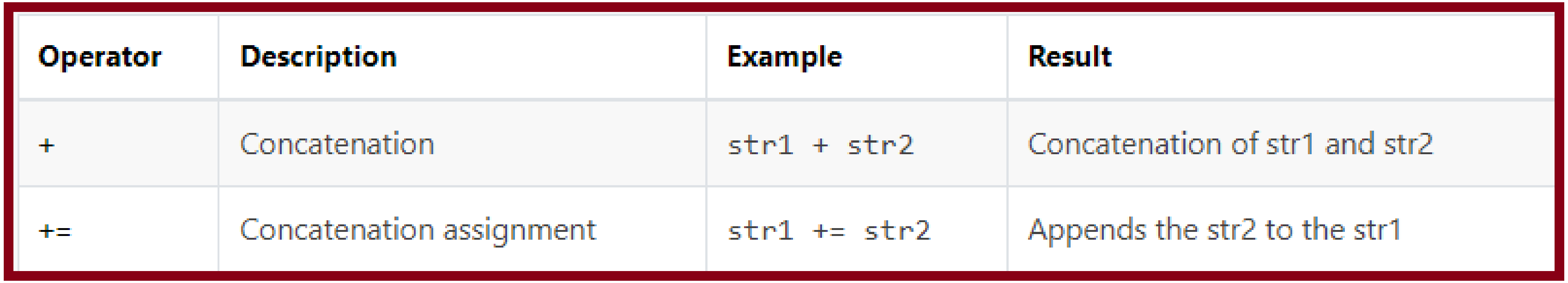


Operator	Description	Example	Is The Same As
=	Assign	x = y	x = y
+=	Add and assign	x += y	x = x + y
-=	Subtract and assign	x -= y	x = x - y
*=	Multiply and assign	x *= y	x = x * y
/=	Divide and assign quotient	x /= y	x = x / y
%=	Divide and assign modulus	x %= y	x = x % y

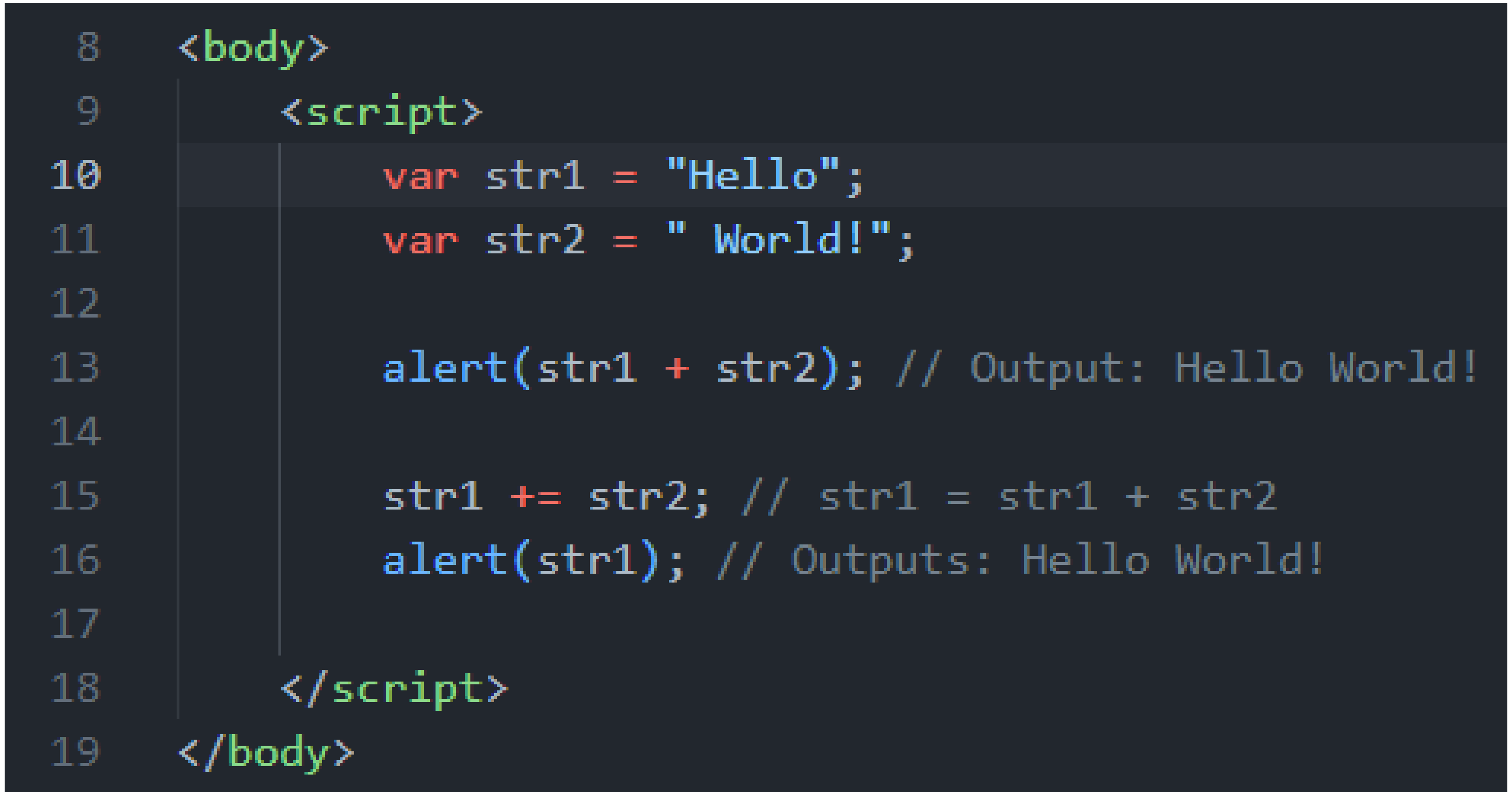


```
<body>

    <script>
        var x;    // Declaring Variable

        x = 10;
        alert(x); // Outputs: 10

        x = 20;
        x += 30;   // x = x + 30
        alert(x); // Outputs: 50

        x = 50;
        x -= 20;  // x = x - 30
        alert(x); // Outputs: 30

        x = 5;
        x *= 25;  // // x = x * 25
        alert(x); // Outputs: 125

        x = 50;
        x /= 10;  // x = x / 10
        alert(x); // Outputs: 5

        x = 100;
        x %= 15;  // x = x % 15
        alert(x); // Outputs: 10

    </script>
</body>
```

JavaScript Assignment Operators (String) …

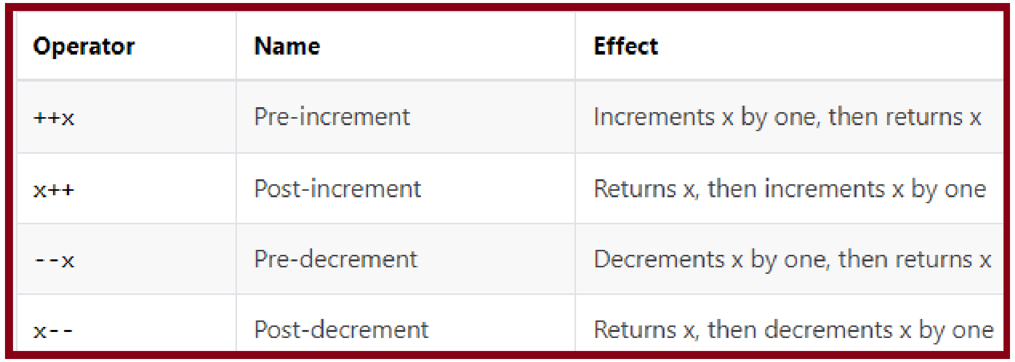


Operator	Description	Example	Result
+	Concatenation	`str1 + str2`	Concatenation of str1 and str2
+=	Concatenation assignment	`str1 += str2`	Appends the str2 to the str1

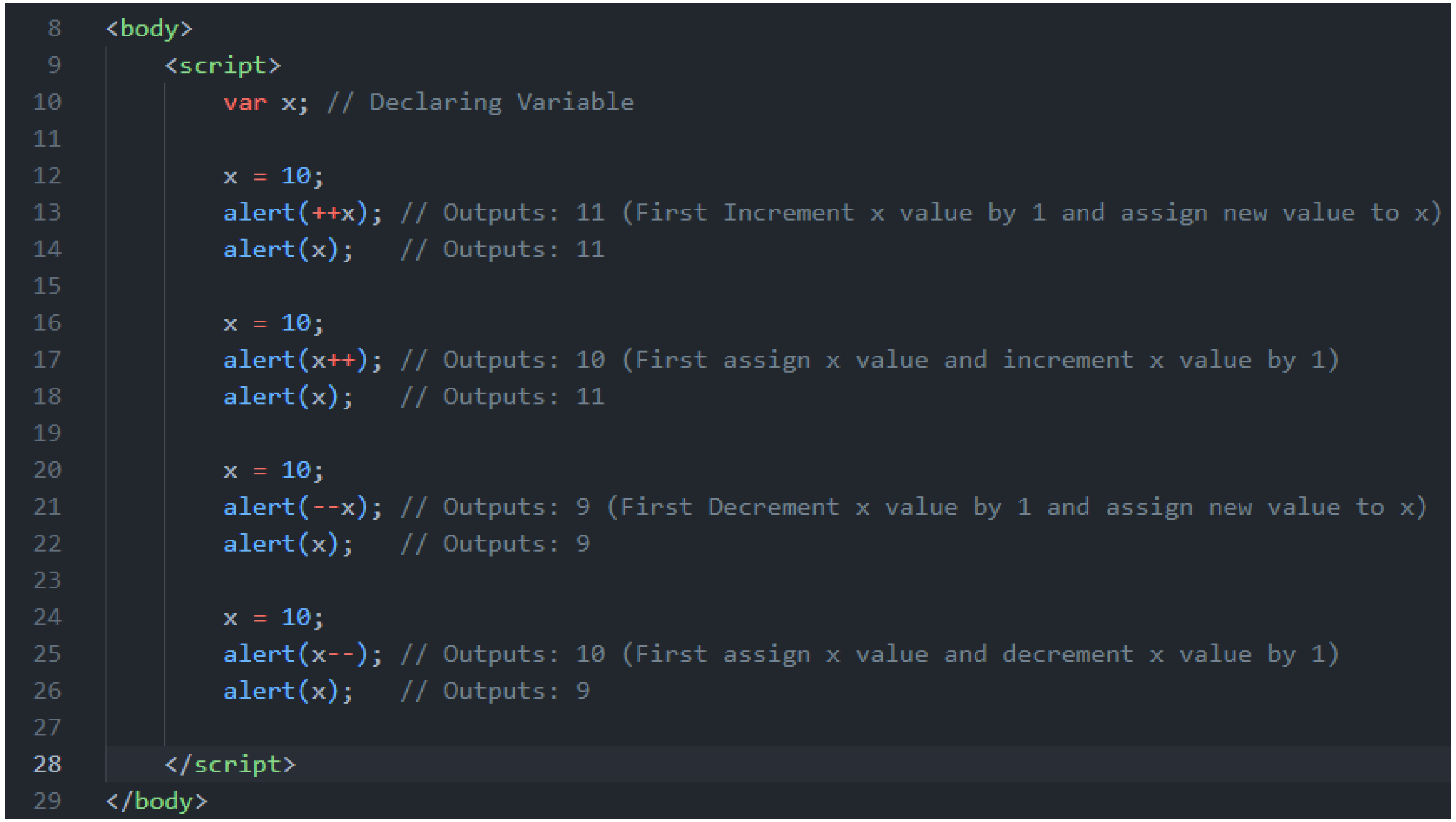


```
<body>
    <script>
        var str1 = "Hello";
        var str2 = " World!";

        alert(str1 + str2); // Output: Hello World!

        str1 += str2; // str1 = str1 + str2
        alert(str1); // Outputs: Hello World!

    </script>
</body>
```

Incrementing and Decrementing Operators In JavaScript

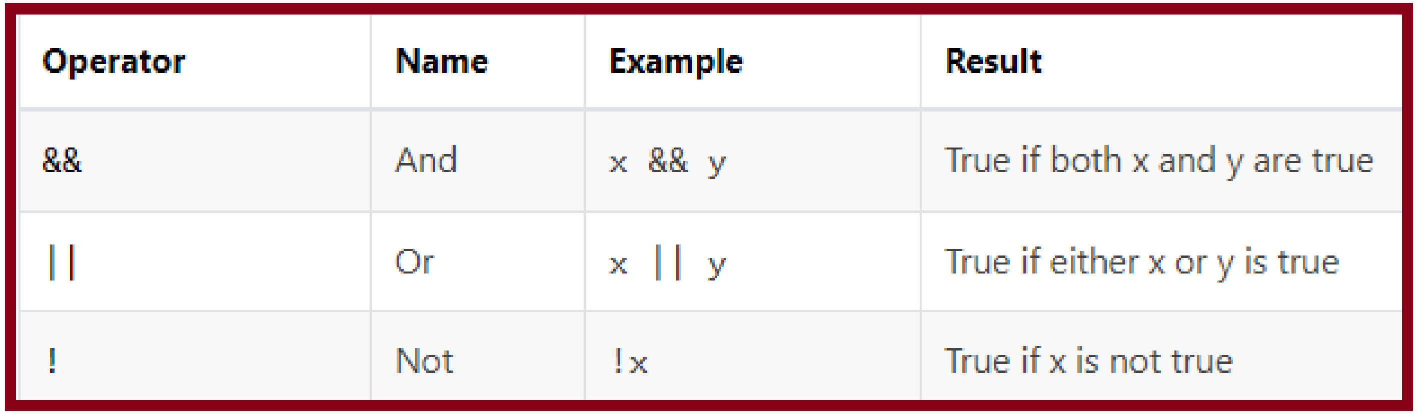


Operator	Name	Effect
++x	Pre-increment	Increments x by one, then returns x
x++	Post-increment	Returns x, then increments x by one
--x	Pre-decrement	Decrements x by one, then returns x
x--	Post-decrement	Returns x, then decrements x by one

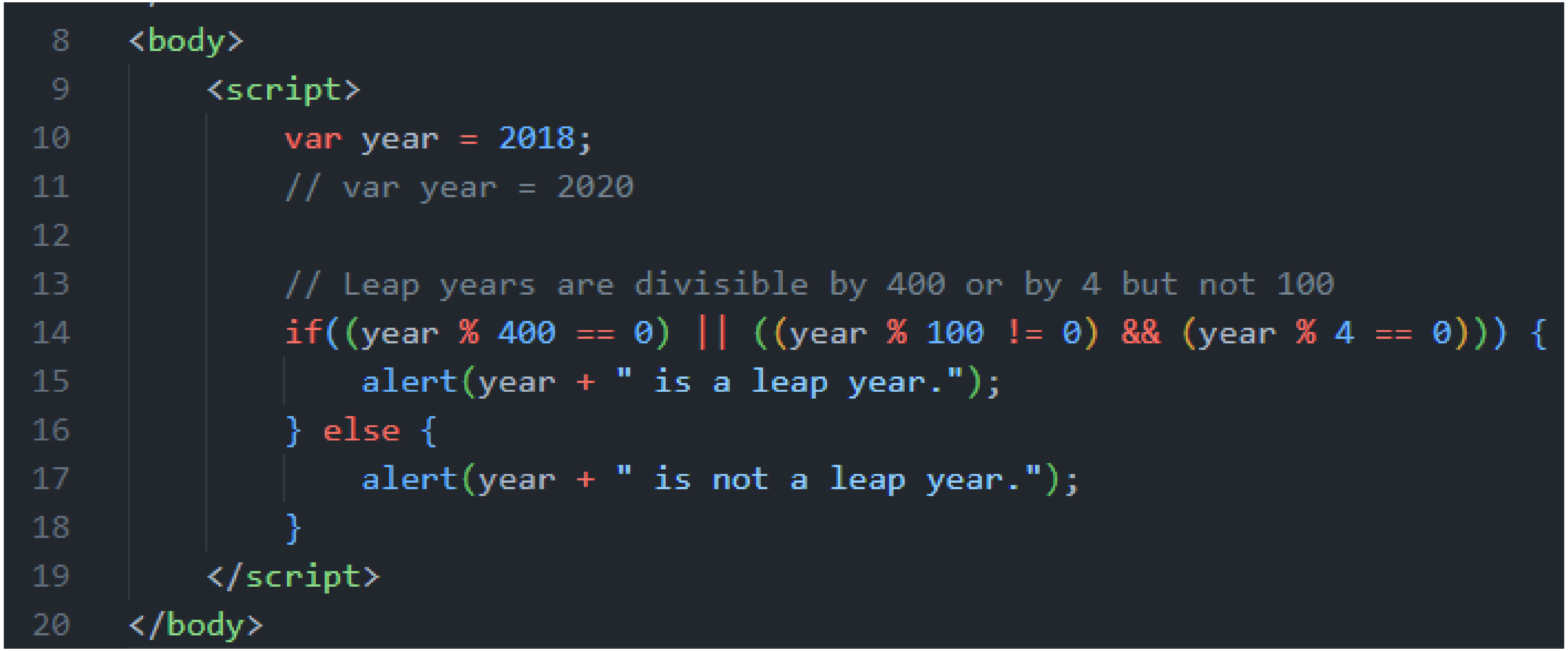


```
<body>
    <script>
        var x; // Declaring Variable

        x = 10;
        alert(++x); // Outputs: 11 (First Increment x value by 1 and assign new value to x)
        alert(x);   // Outputs: 11

        x = 10;
        alert(x++); // Outputs: 10 (First assign x value and increment x value by 1)
        alert(x);   // Outputs: 11

        x = 10;
        alert(--x); // Outputs: 9 (First Decrement x value by 1 and assign new value to x)
        alert(x);   // Outputs: 9

        x = 10;
        alert(x--); // Outputs: 10 (First assign x value and decrement x value by 1)
        alert(x);   // Outputs: 9

    </script>
</body>
```

JavaScript Logical Operators

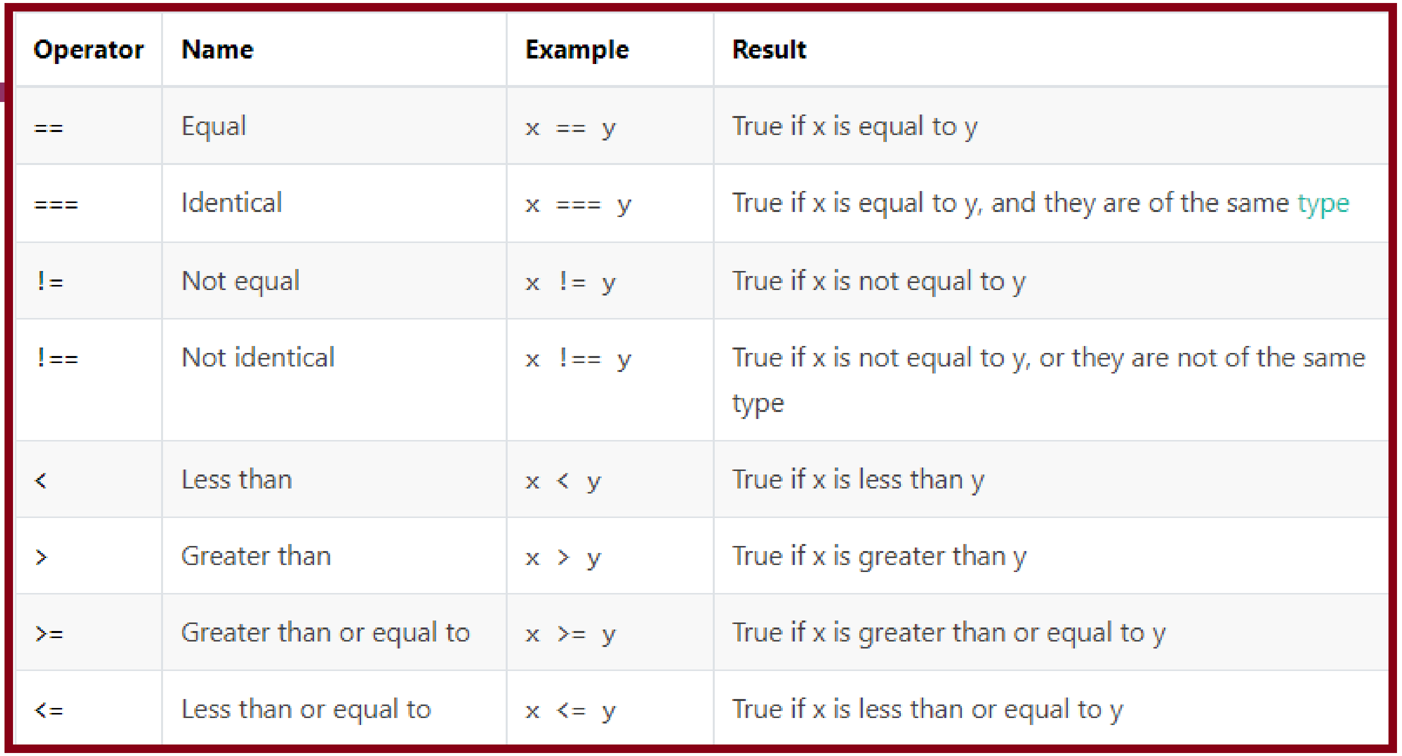


Operator	Name	Example	Result
&&	And	x && y	True if both x and y are true
||	Or	x || y	True if either x or y is true
!	Not	!x	True if x is not true

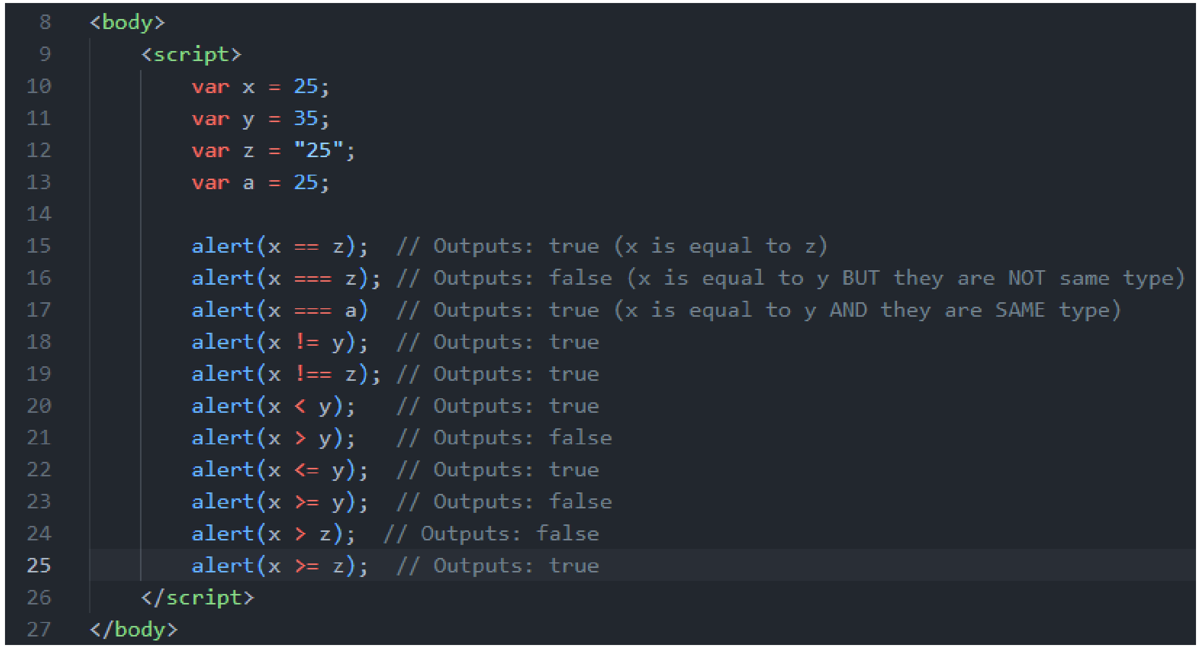


```
<body>
    <script>
        var year = 2018;
        // var year = 2020

        // Leap years are divisible by 400 or by 4 but not 100
        if((year % 400 == 0) || ((year % 100 != 0) && (year % 4 == 0))) {
            alert(year + " is a leap year.");
        } else {
            alert(year + " is not a leap year.");
        }
    </script>
</body>
```

JavaScript Comparison Operators



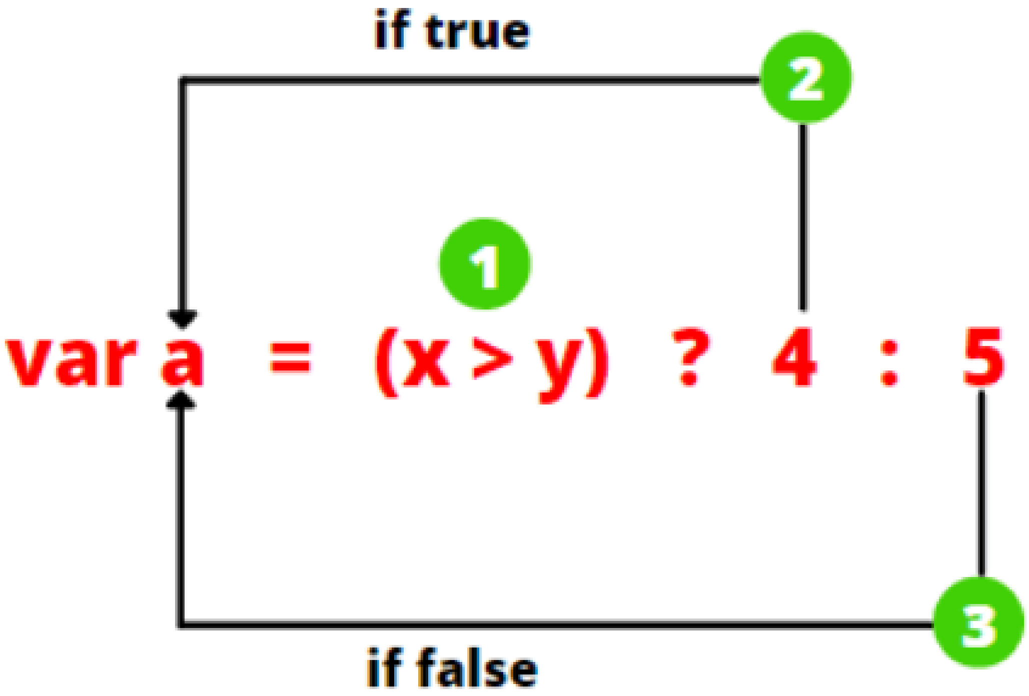
Operator	Name	Example	Result
==	Equal	x == y	True if x is equal to y
===	Identical	x === y	True if x is equal to y, and they are of the same type
!=	Not equal	x != y	True if x is not equal to y
!==	Not identical	x !== y	True if x is not equal to y, or they are not of the same type
<	Less than	x < y	True if x is less than y
>	Greater than	x > y	True if x is greater than y
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y
<=	Less than or equal to	x <= y	True if x is less than or equal to y



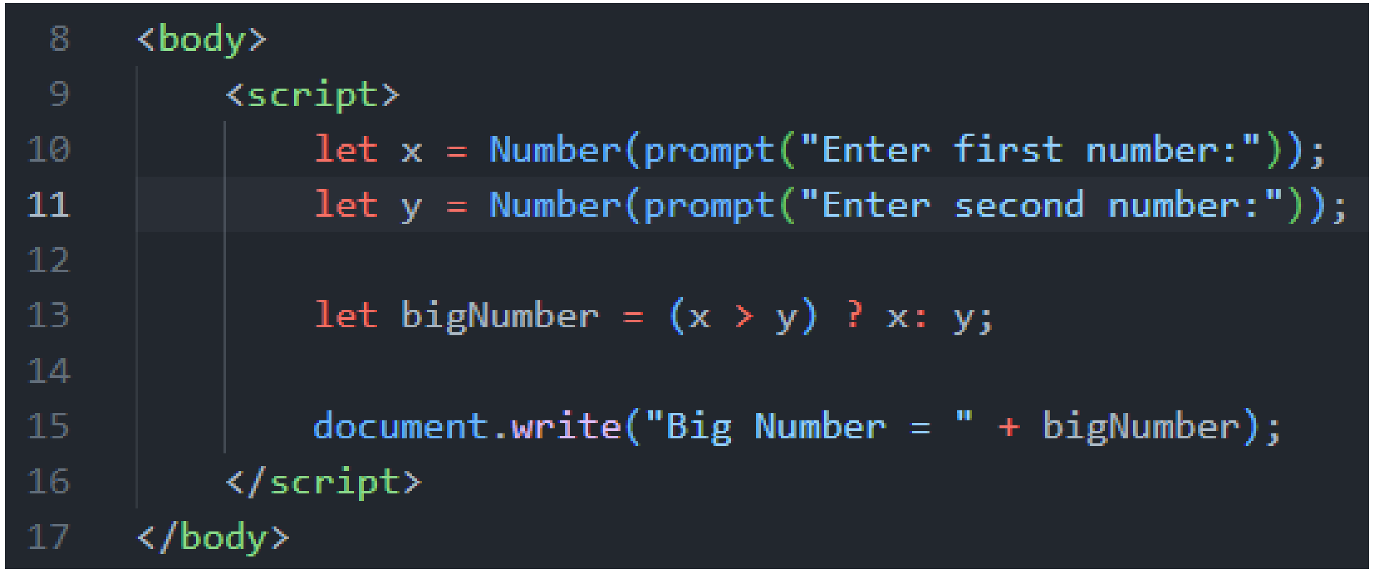
```
<body>
    <script>
        var x = 25;
        var y = 35;
        var z = "25";
        var a = 25;

        alert(x == z);  // Outputs: true (x is equal to z)
        alert(x === z); // Outputs: false (x is equal to y BUT they are NOT same type)
        alert(x === a)  // Outputs: true (x is equal to y AND they are SAME type)
        alert(x != y);  // Outputs: true
        alert(x !== z); // Outputs: true
        alert(x < y);   // Outputs: true
        alert(x > y);   // Outputs: false
        alert(x <= y);  // Outputs: true
        alert(x >= y);  // Outputs: false
        alert(x > z);  // Outputs: false
        alert(x >= z);  // Outputs: true
    </script>
</body>
```

Conditional (OR) Ternary Operator in JavaScript

- Conditional operator provides a one line approach for creating a simple conditional statement.
- It is often used as a shorthand method for if-else statement.
- The conditional operator (?:) is also known as ternary operator.
- Syntax



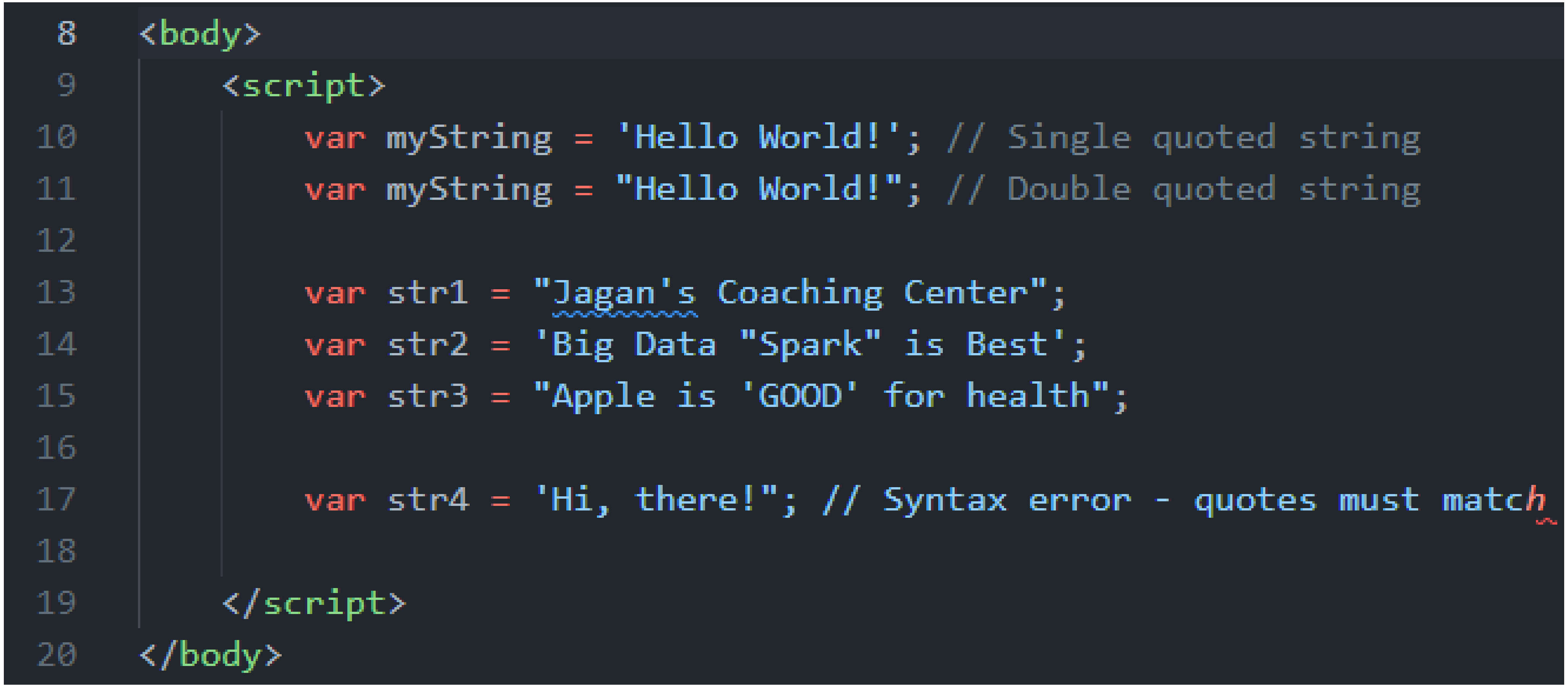
```
var a = (x > y) ? 4 : 5
```



```
<body>
    <script>
        let x = Number(prompt("Enter first number:"));
        let y = Number(prompt("Enter second number:"));

        let bigNumber = (x > y) ? x: y;

        document.write("Big Number = " + bigNumber);
    </script>
</body>
```

JavaScript Strings

- A string is a sequence of letters, numbers, special characters or combination of all.
- Strings can be created by enclosing the string literal (i.e. string characters) either within single quotes (') or double quotes (")



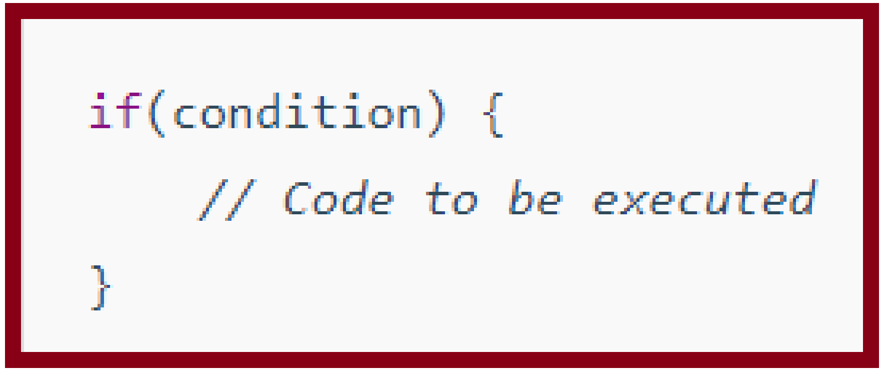
```
<body>
    <script>
        var myString = 'Hello World!'; // Single quoted string
        var myString = "Hello World!"; // Double quoted string

        var str1 = "Jagan's Coaching Center";
        var str2 = 'Big Data "Spark" is Best';
        var str3 = "Apple is 'GOOD' for health";

        var str4 = 'Hi, there!"; // Syntax error - quotes must match

    </script>
</body>
```

JavaScript Conditional(if…else) Statements

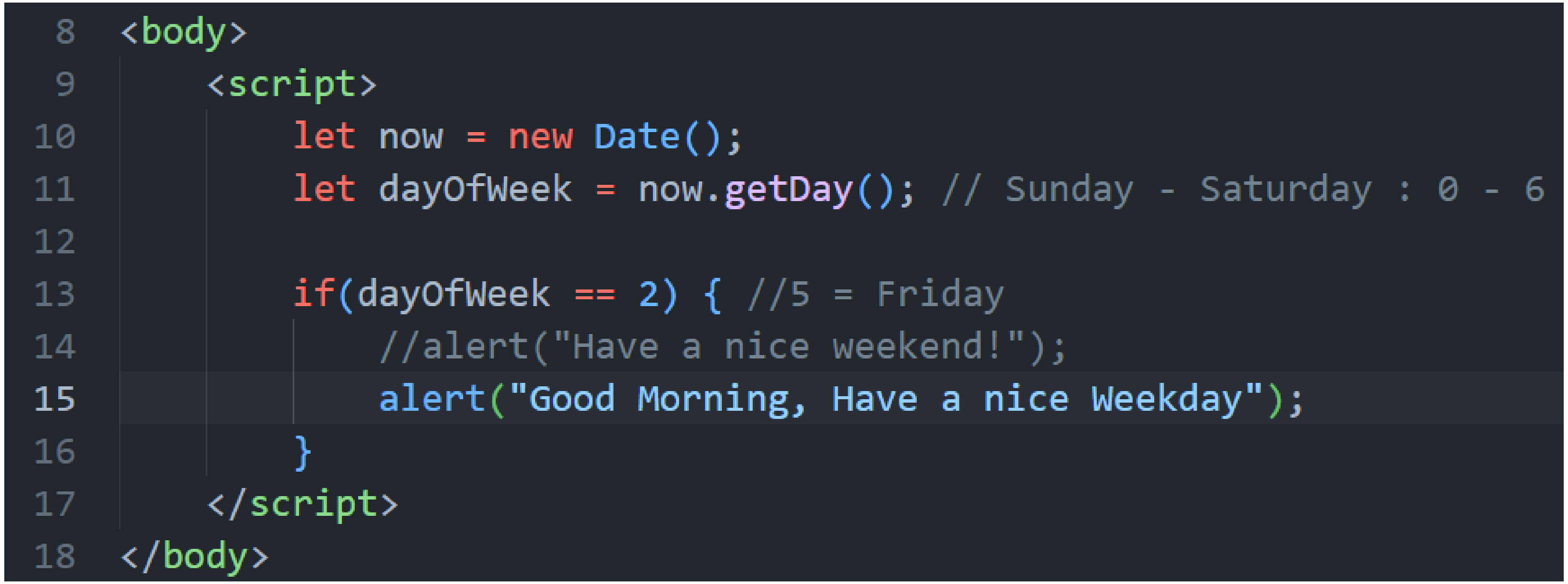
- JavaScript allows us to write the code that perform different actions based on some condition.
- Condition expressions evaluates to either true or false. Based on these results you can perform certain actions.
- Conditional statements in JavaScript
  - ✓ if statement
  - ✓ if...else statement
  - ✓ if...else if....else statement
  - ✓ switch...case statement

if Statement

The if statement is used to execute a block of code only if the specified condition evaluates to **true**



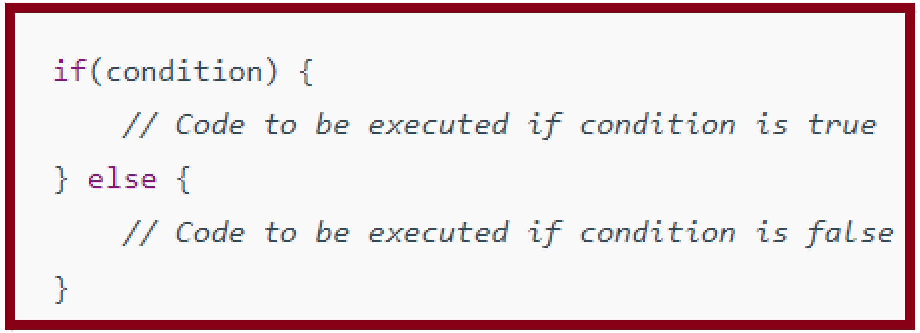
```
if(condition) {
    // Code to be executed
}
```



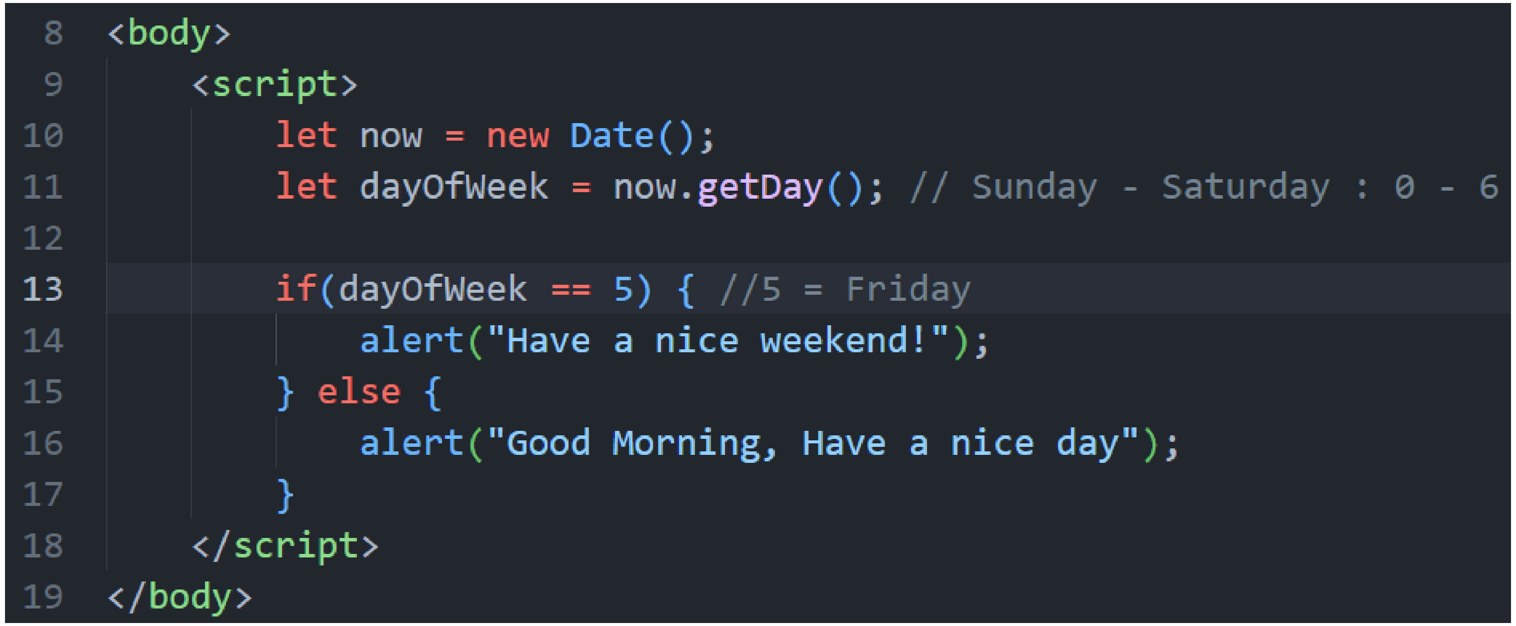
```
<body>
    <script>
        let now = new Date();
        let dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

        if(dayOfWeek == 2) { //5 = Friday
            //alert("Have a nice weekend!");
            alert("Good Morning, Have a nice Weekday");
        }
    </script>
</body>
```

if...else Statement

- We can enhance the decision making capabilities by providing an alternative choice through adding an **else** statement to the **if** statement.
- The **if...else** statement allows us to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false.



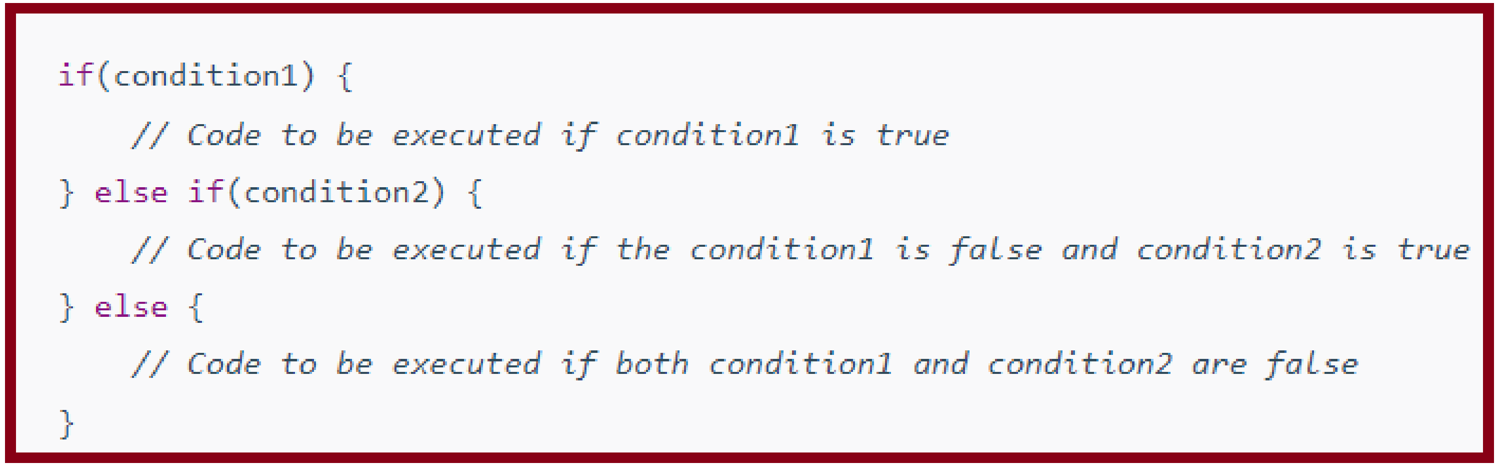
```
if(condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```



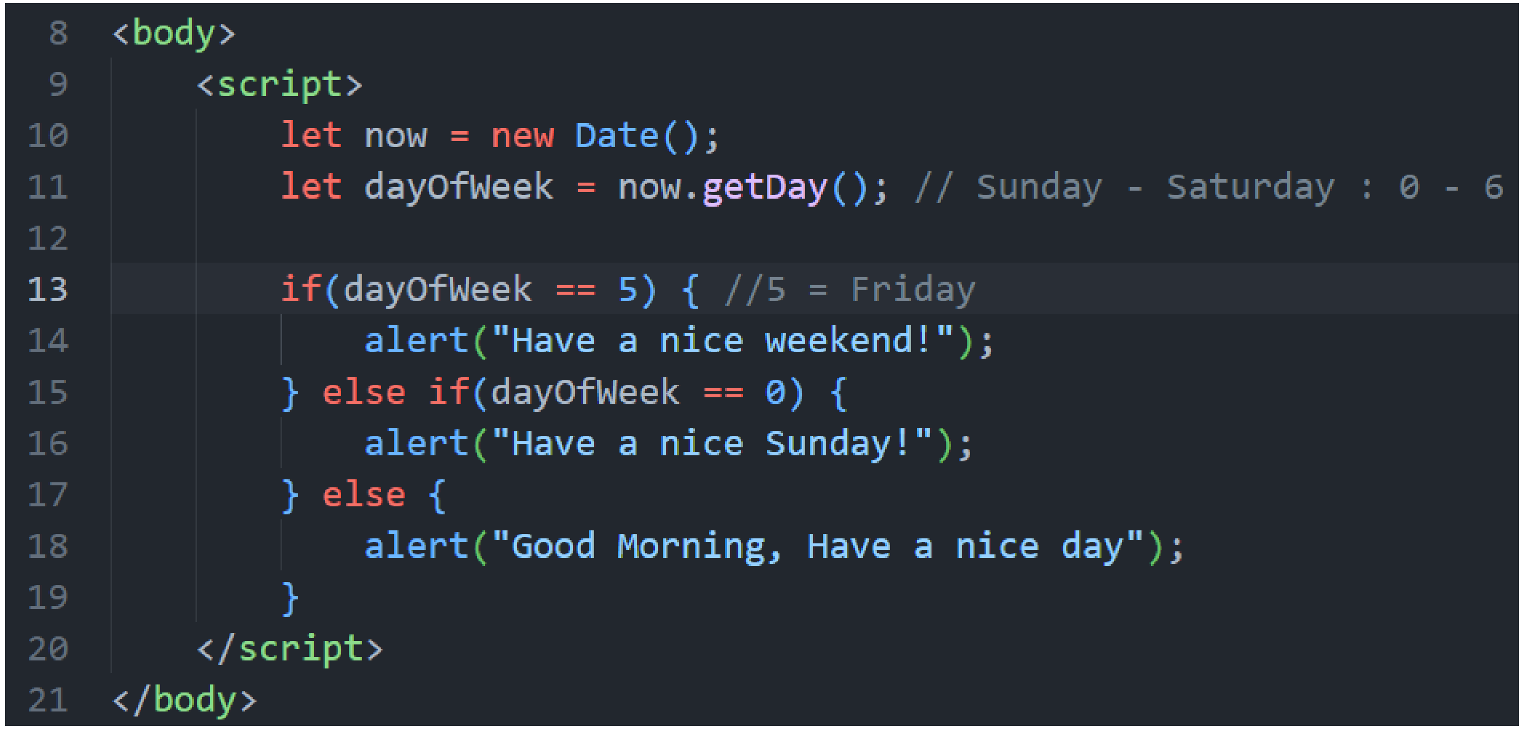
```
<body>
    <script>
        let now = new Date();
        let dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

        if(dayOfWeek == 5) { //5 = Friday
            alert("Have a nice weekend!");
        } else {
            alert("Good Morning, Have a nice day");
        }
    </script>
</body>
```

if...else if...else Statement

- if...else if...else a special statement used to combine multiple if...else statements

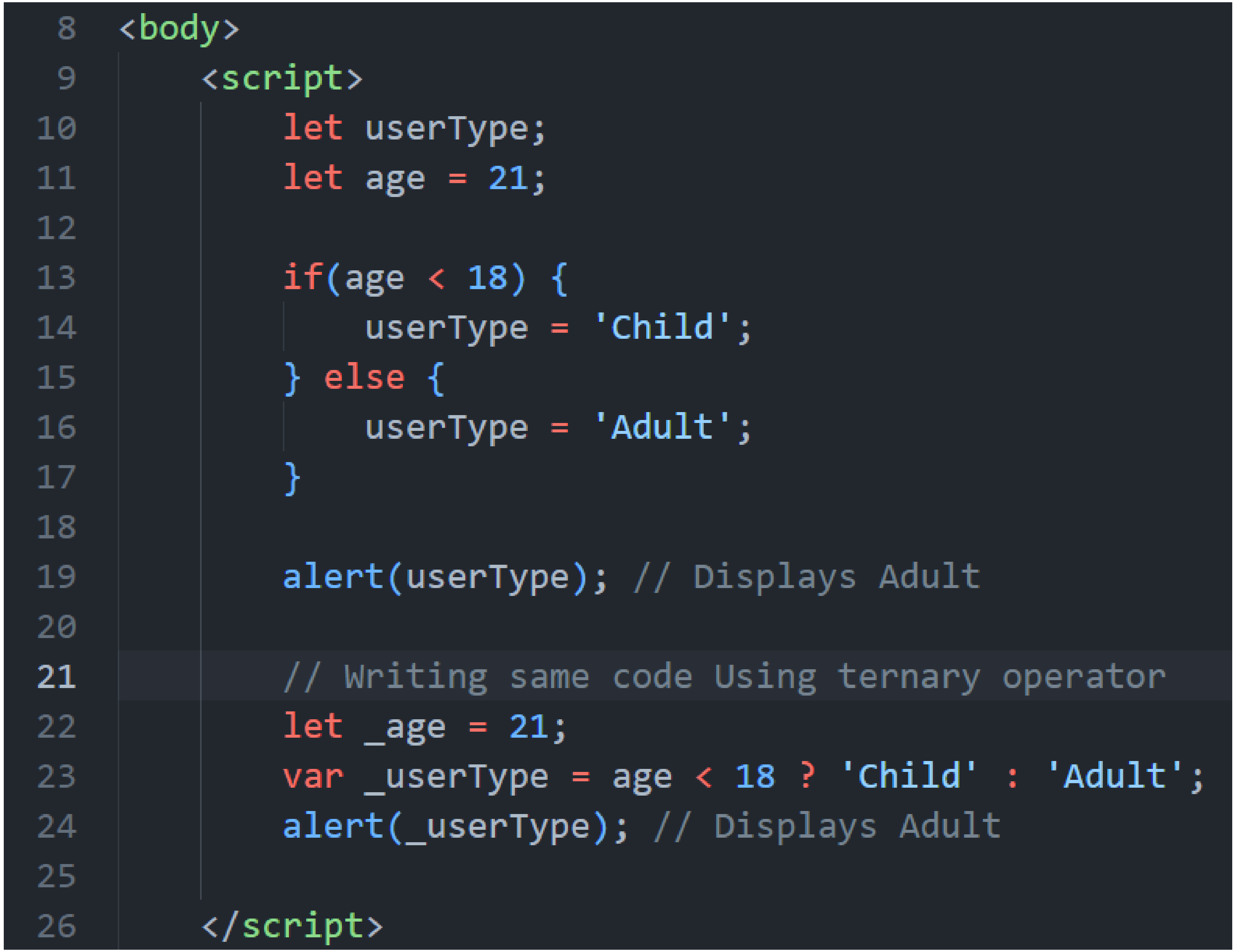


```
if(condition1) {
    // Code to be executed if condition1 is true
} else if(condition2) {
    // Code to be executed if the condition1 is false and condition2 is true
} else {
    // Code to be executed if both condition1 and condition2 are false
}
```



```
<body>
    <script>
        let now = new Date();
        let dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

        if(dayOfWeek == 5) { //5 = Friday
            alert("Have a nice weekend!");
        } else if(dayOfWeek == 0) {
            alert("Have a nice Sunday!");
        } else {
            alert("Good Morning, Have a nice day");
        }
    </script>
</body>
```

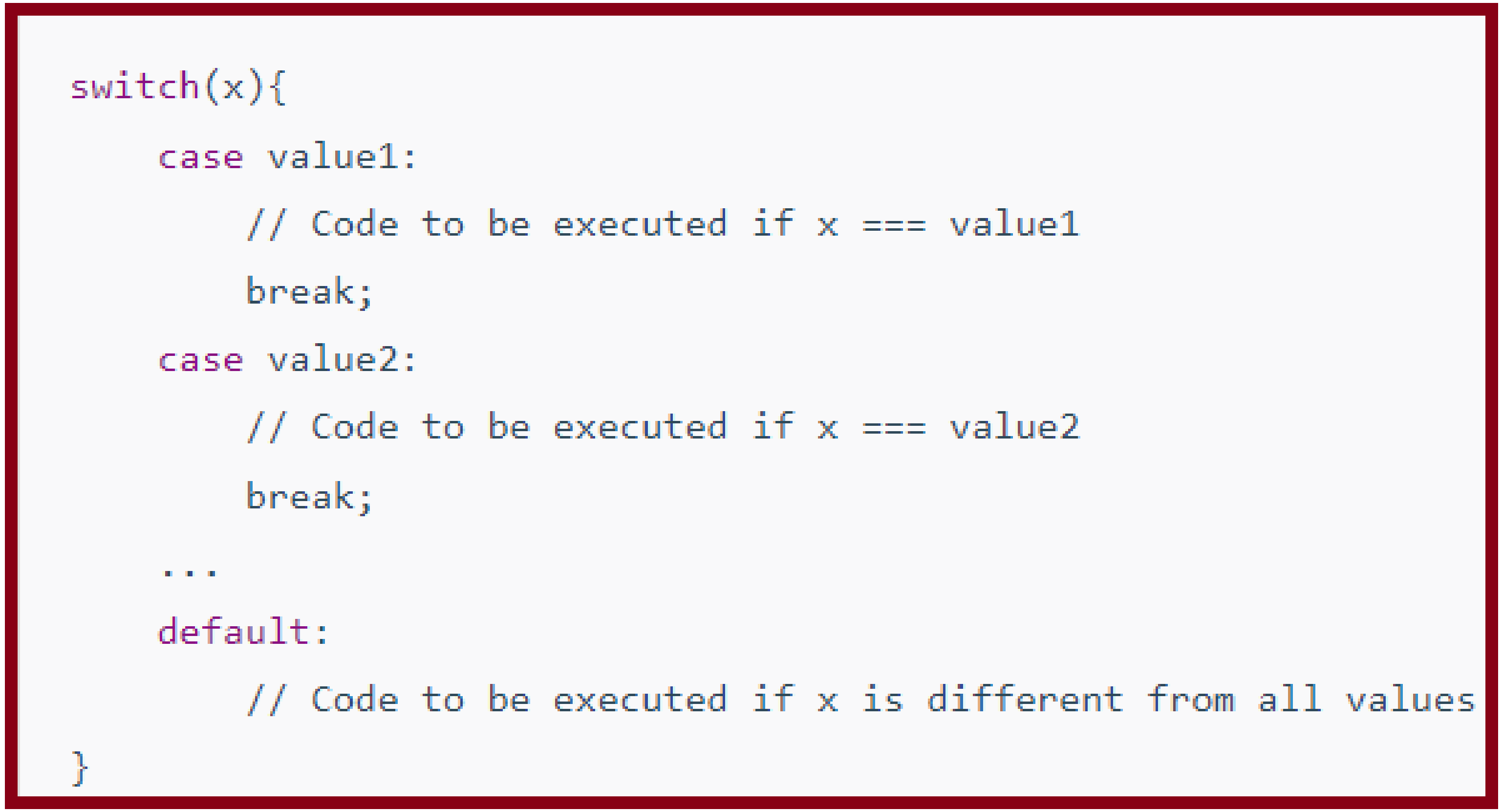
if...else Vs Ternary Operator



```
<body>
    <script>
        let userType;
        let age = 21;

        if(age < 18) {
            userType = 'Child';
        } else {
            userType = 'Adult';
        }

        alert(userType); // Displays Adult

        // Writing same code Using ternary operator
        let _age = 21;
        var _userType = age < 18 ? 'Child' : 'Adult';
        alert(_userType); // Displays Adult

    </script>
```

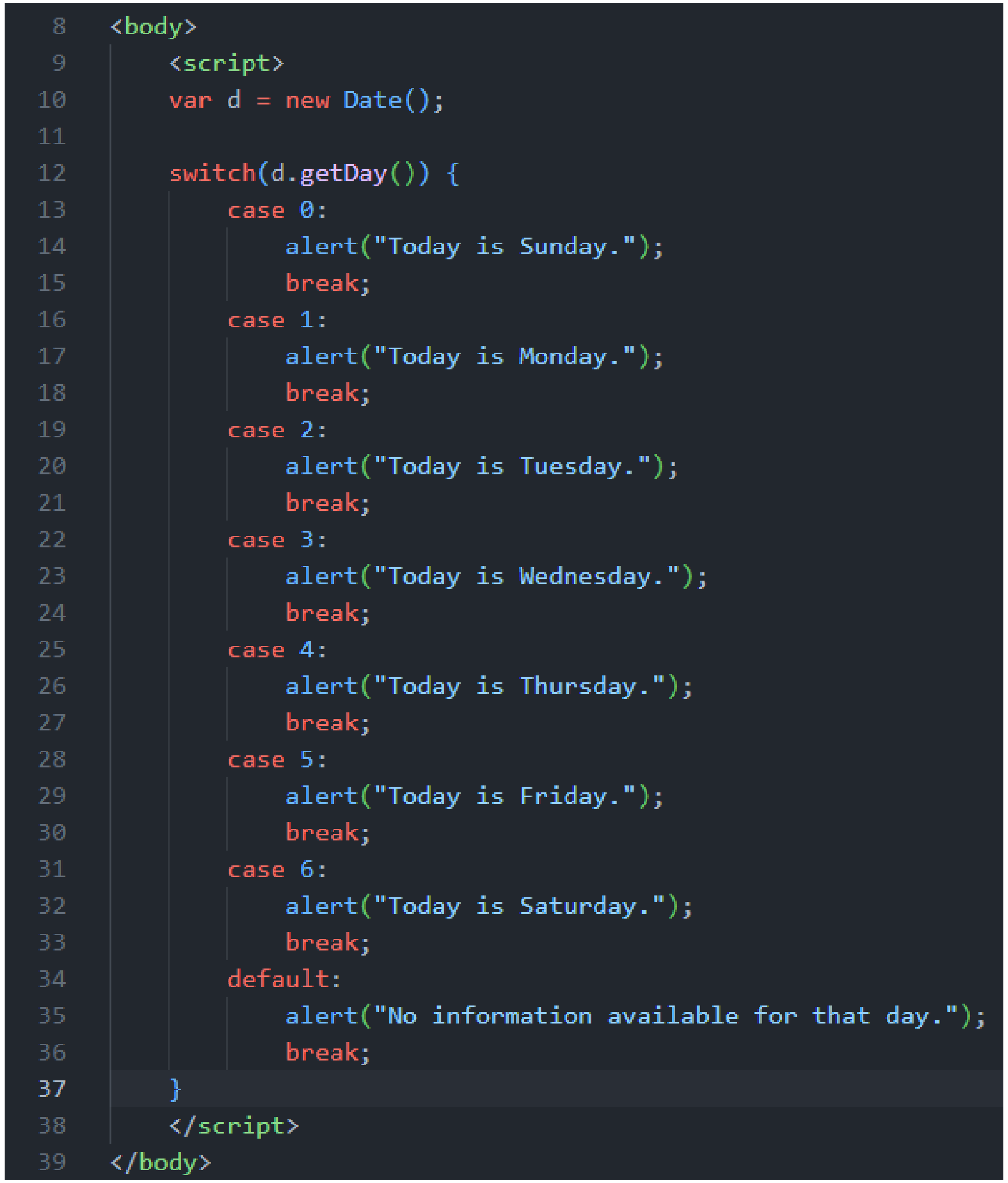
Switch...Case Statement

- switch..case statement is an alternative to the if...else if...else statement.
- The switch...case statement tests a variable or expression against a series of values until it finds a match, and then executes the block of code corresponding to that match.



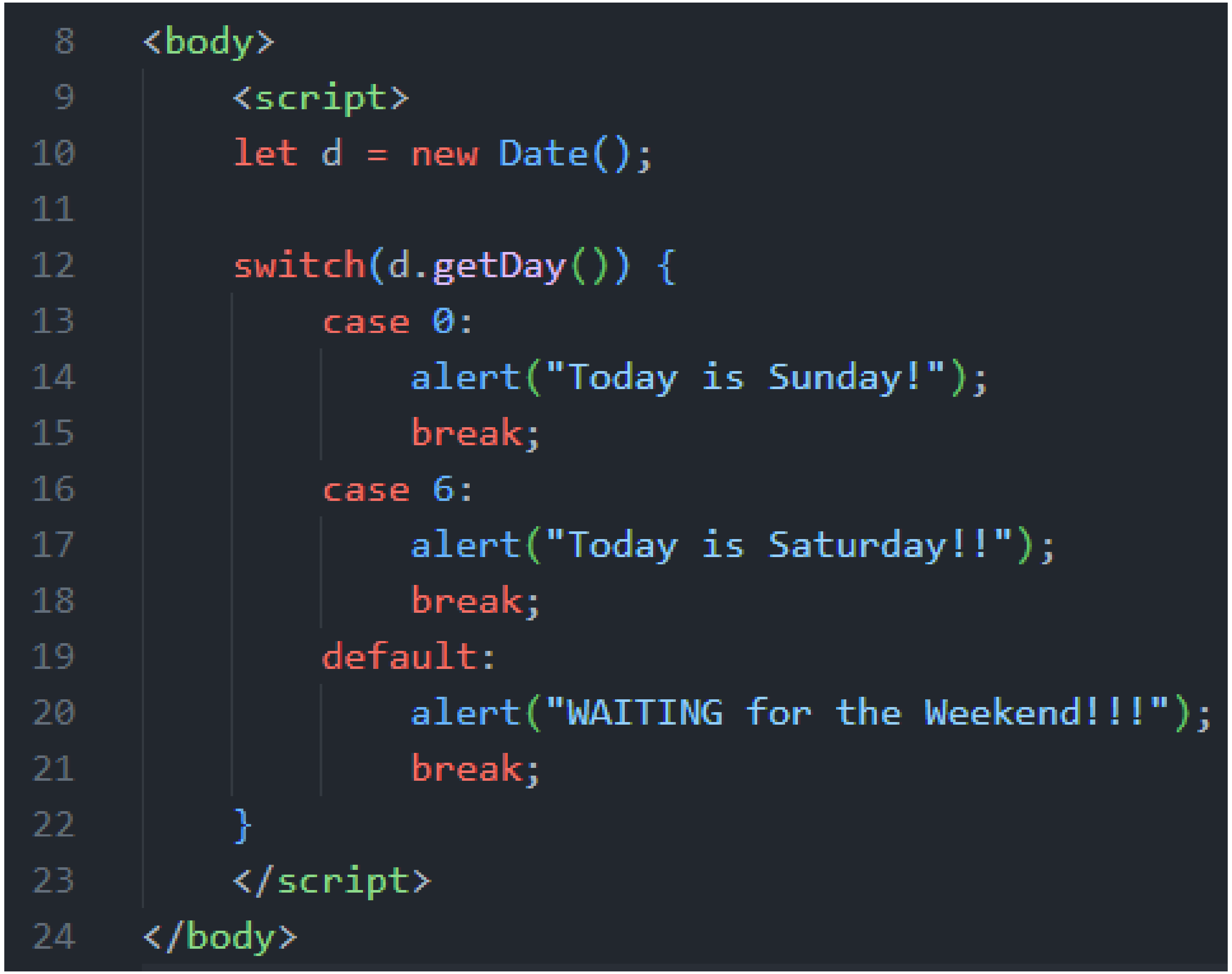
```
switch(x){
    case value1:
        // Code to be executed if x === value1
        break;
    case value2:
        // Code to be executed if x === value2
        break;
    ...
    default:
        // Code to be executed if x is different from all values
}
```

Switch...Case Statement… Demo-1



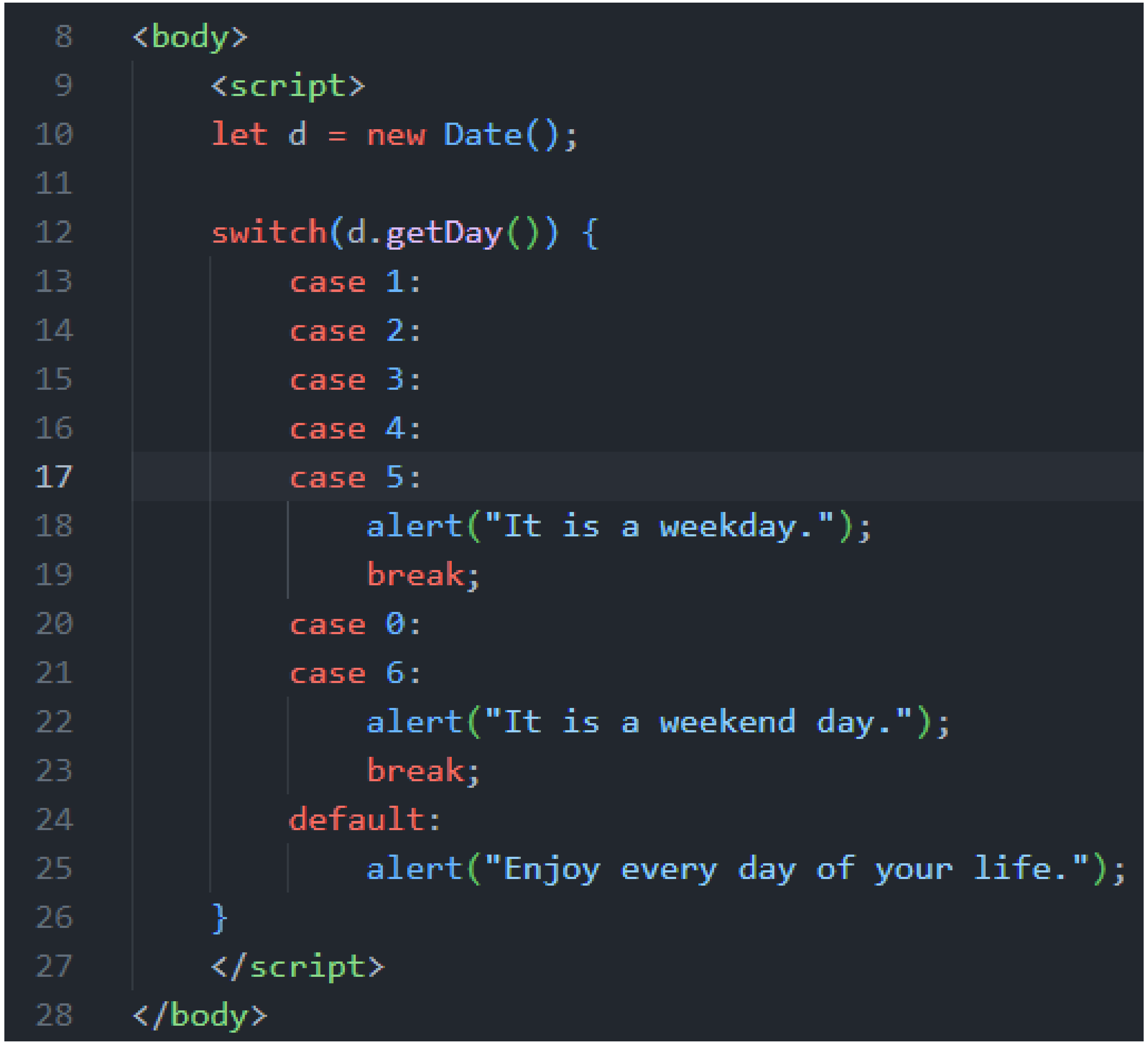
```
<body>
    <script>
    var d = new Date();

    switch(d.getDay()) {
        case 0:
            alert("Today is Sunday.");
            break;
        case 1:
            alert("Today is Monday.");
            break;
        case 2:
            alert("Today is Tuesday.");
            break;
        case 3:
            alert("Today is Wednesday.");
            break;
        case 4:
            alert("Today is Thursday.");
            break;
        case 5:
            alert("Today is Friday.");
            break;
        case 6:
            alert("Today is Saturday.");
            break;
        default:
            alert("No information available for that day.");
            break;
    }
    </script>
</body>
```

Switch...Case Statement… Demo - 2



```
<body>
    <script>
    let d = new Date();

    switch(d.getDay()) {
        case 0:
            alert("Today is Sunday!");
            break;
        case 6:
            alert("Today is Saturday!!");
            break;
        default:
            alert("WAITING for the Weekend!!!");
            break;
    }
    </script>
</body>
```

Multiple Cases Sharing Same Action … Demo - 3



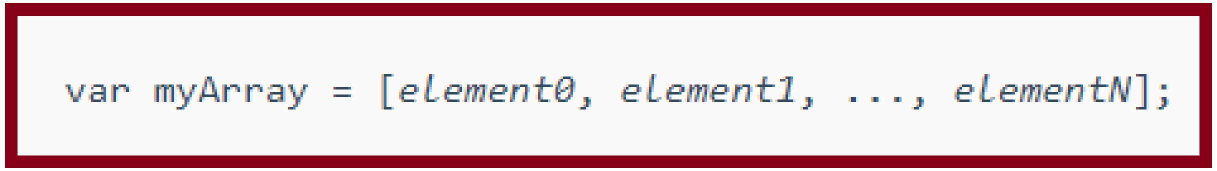
```
<body>
    <script>
    let d = new Date();

    switch(d.getDay()) {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            alert("It is a weekday.");
            break;
        case 0:
        case 6:
            alert("It is a weekend day.");
            break;
        default:
            alert("Enjoy every day of your life.");
    }
    </script>
</body>
```

JavaScript Arrays

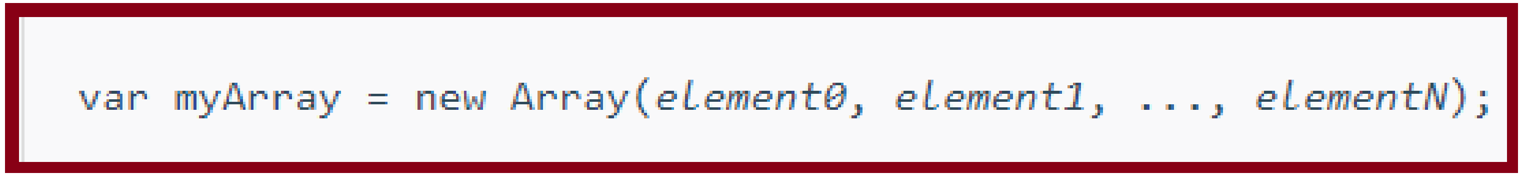
- Arrays are complex data types that allow us to store more than one value OR a group of values under a single variable name.
- JavaScript arrays can store any valid value, including strings, numbers, objects, functions, and even other arrays.
- Using Arrays, its possible to create more complex data structures such as an array of objects OR array of arrays.
- Array is a ordered collection of values. Each value in array is called element, and each element has a numeric position called index.

Creating Array

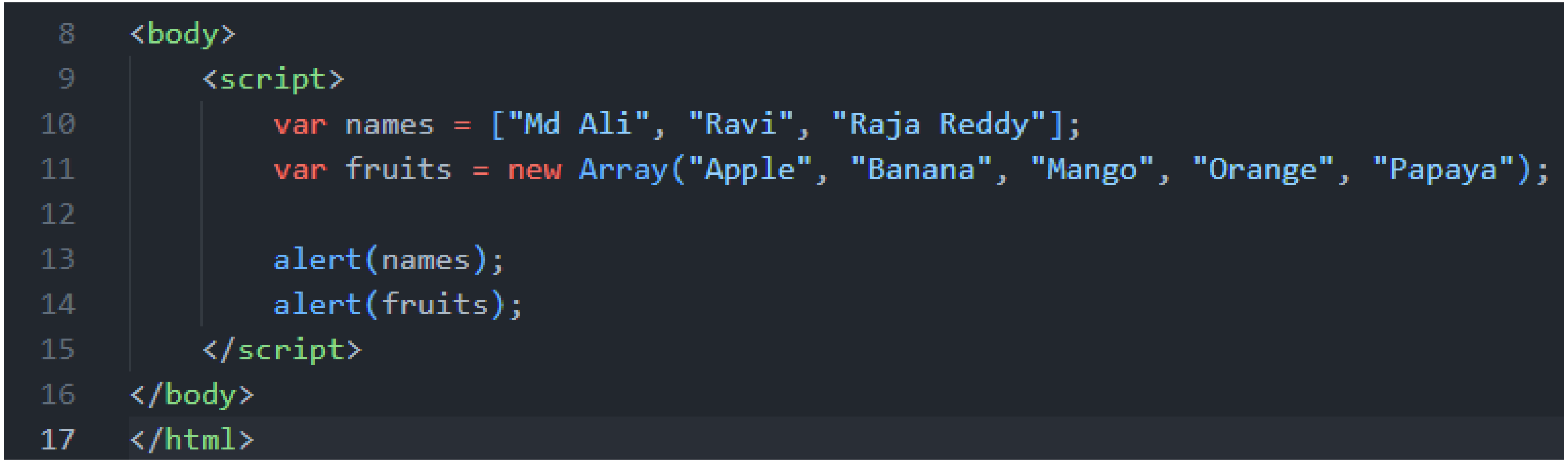
- Comma-separated list of values in square brackets ([])
- Using the Array() constructor



```
var myArray = [element0, element1, ..., elementN];
```



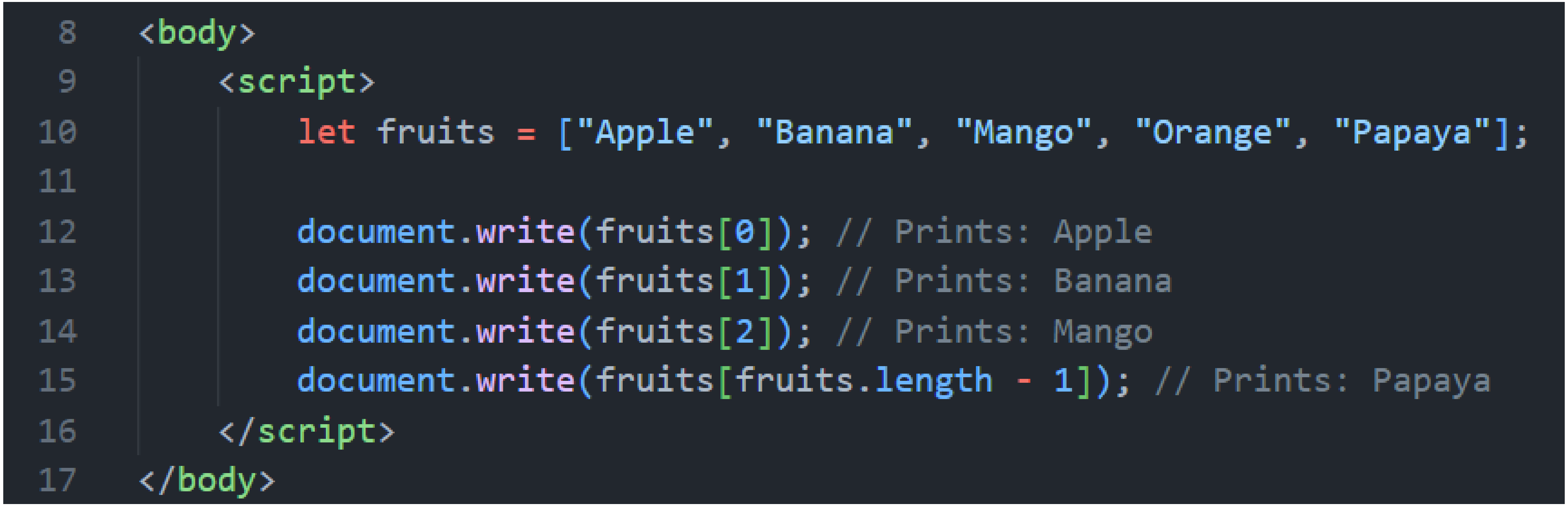
```
var myArray = new Array(element0, element1, ..., elementN);
```



```
<body>
    <script>
        var names = ["Md Ali", "Ravi", "Raja Reddy"];
        var fruits = new Array("Apple", "Banana", "Mango", "Orange", "Papaya");

        alert(names);
        alert(fruits);
    </script>
</body>
</html>
```

Accessing the Elements of an Array

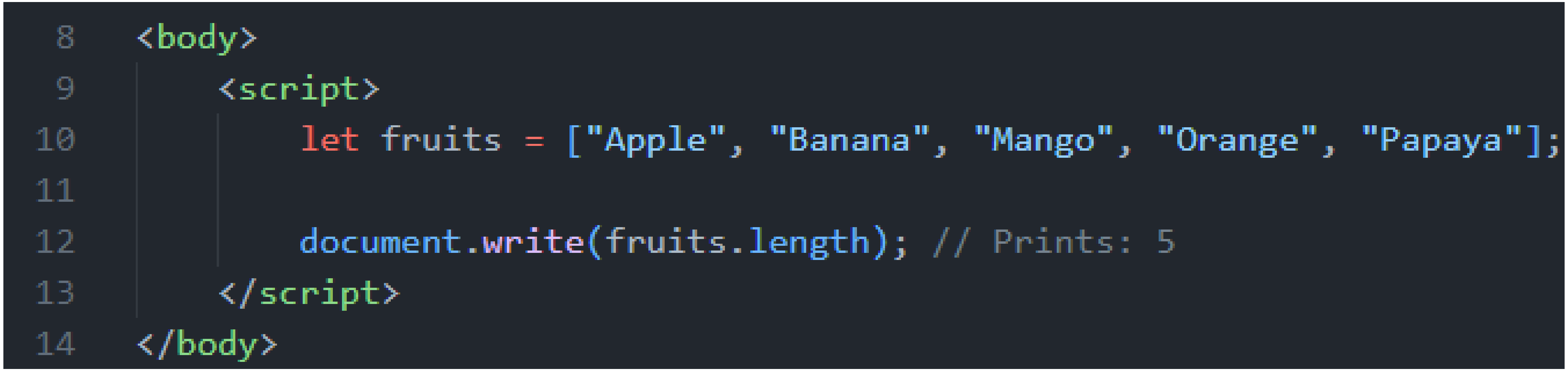
- Array indexes are zero-based i.e., first element of an array is stored at index 0, second element is stored at index 1, and so on.
- JavaScript arrays are special type of objects and typeof operator will return "object" for arrays.



```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

        document.write(fruits[0]); // Prints: Apple
        document.write(fruits[1]); // Prints: Banana
        document.write(fruits[2]); // Prints: Mango
        document.write(fruits[fruits.length - 1]); // Prints: Papaya
    </script>
</body>
```

Length of an Array

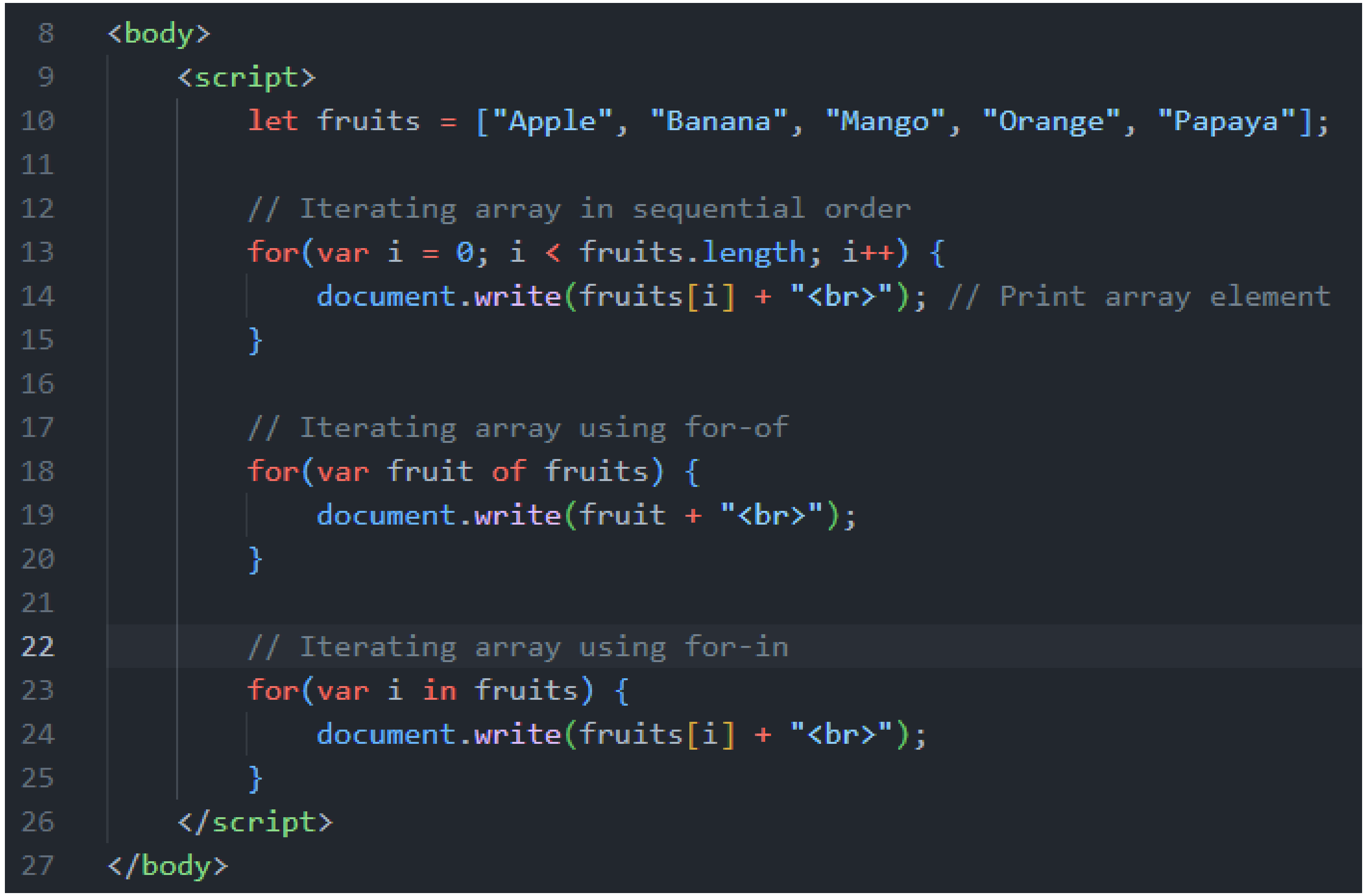
- length property returns the length of an array
- Length is total number of elements present in array



```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

        document.write(fruits.length); // Prints: 5
    </script>
</body>
```

Iterating Array Elements

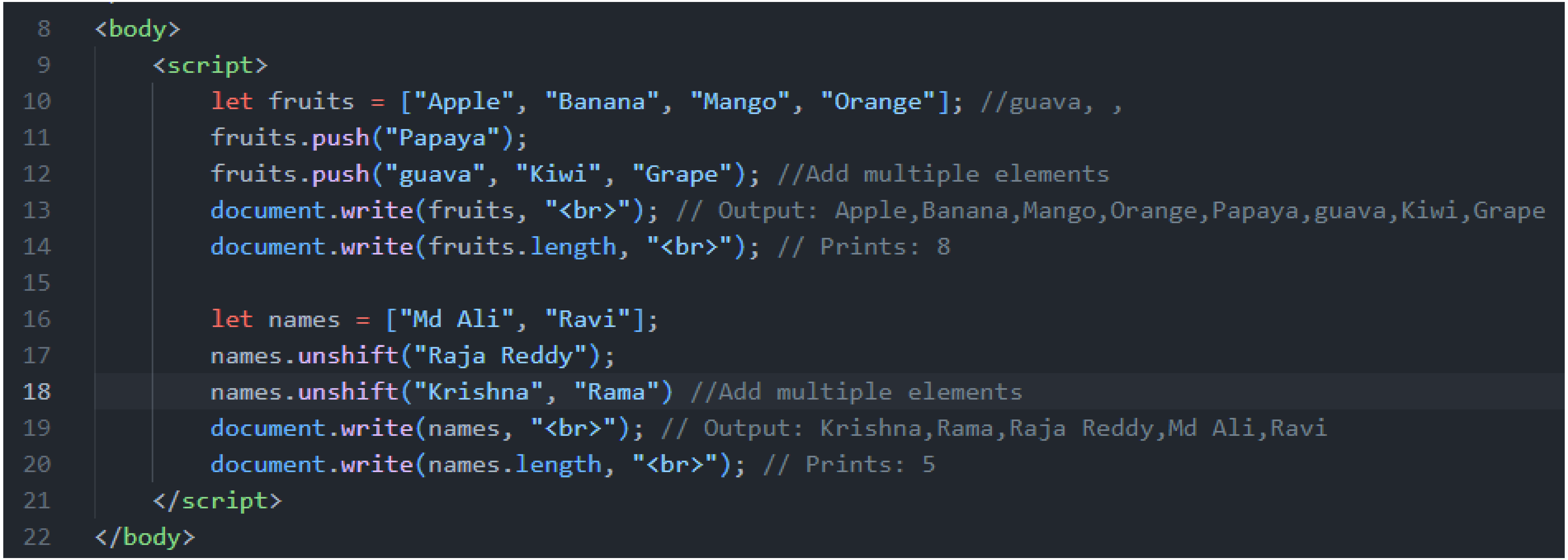
- for loop is to access each element of array in sequential order.
- ECMAScript 6 has a simple way to iterate array elements using **for-of** loop.
  - for-of loop we don't have to initialize and keep track of the loop counter variable (i).
- We can also iterate array elements using for-in loop



```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

        // Iterating array in sequential order
        for(var i = 0; i < fruits.length; i++) {
            document.write(fruits[i] + "<br>"); // Print array element
        }

        // Iterating array using for-of
        for(var fruit of fruits) {
            document.write(fruit + "<br>");
        }

        // Iterating array using for-in
        for(var i in fruits) {
            document.write(fruits[i] + "<br>");
        }
    </script>
</body>
```

Adding New Elements to Array

- push() method is used to add a new element at the end of an array.
- unshift() method is used to add a new element at the beginning of an array

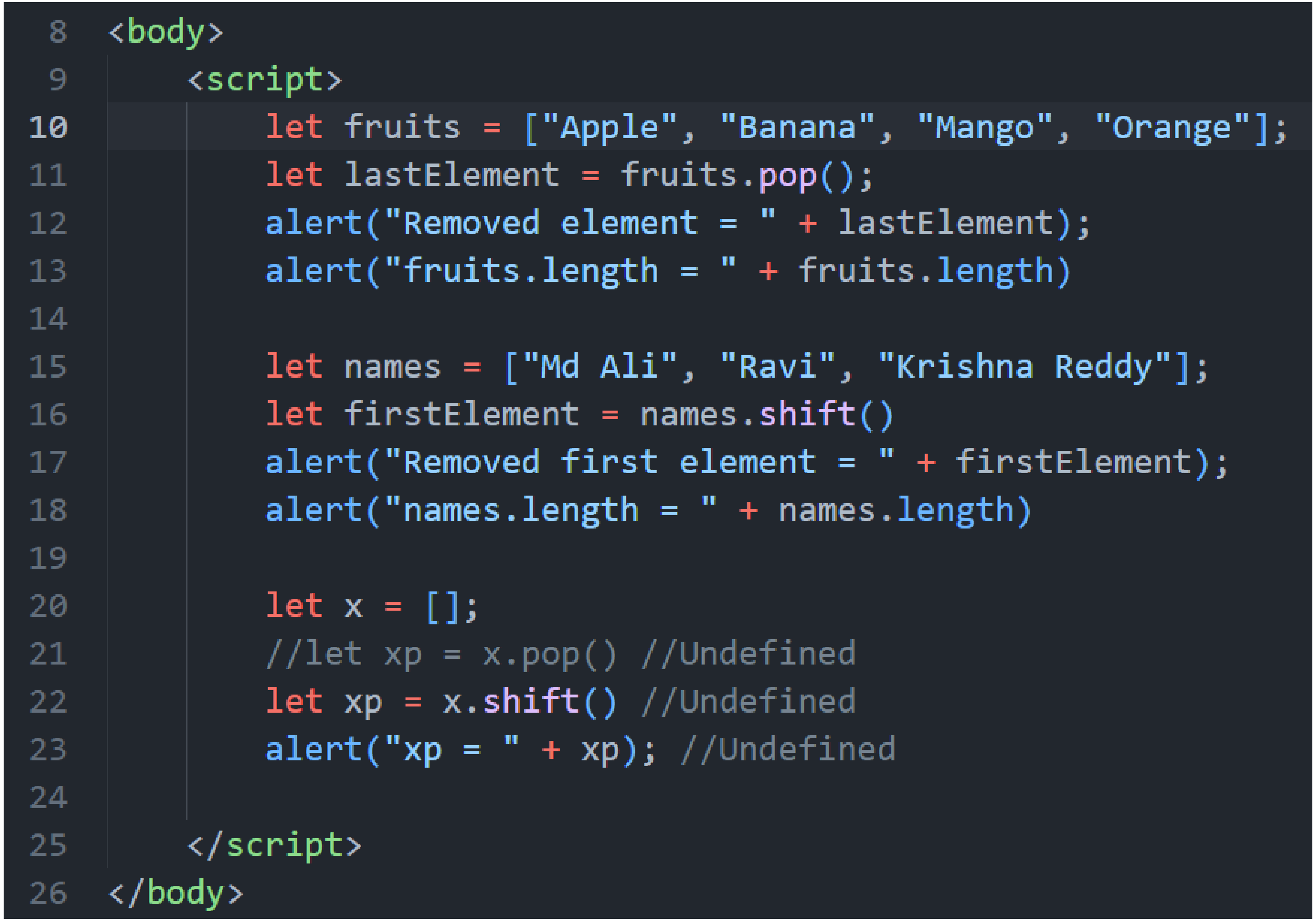


```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange"]; //guava, ,
        fruits.push("Papaya");
        fruits.push("guava", "Kiwi", "Grape"); //Add multiple elements
        document.write(fruits, "<br>"); // Output: Apple,Banana,Mango,Orange,Papaya,guava,Kiwi,Grape
        document.write(fruits.length, "<br>"); // Prints: 8

        let names = ["Md Ali", "Ravi"];
        names.unshift("Raja Reddy");
        names.unshift("Krishna", "Rama") //Add multiple elements
        document.write(names, "<br>"); // Output: Krishna,Rama,Raja Reddy,Md Ali,Ravi
        document.write(names.length, "<br>"); // Prints: 5
    </script>
</body>
```

Removing Elements from Array

- To remove the last element from array we can use the pop() method.
- pop() method returns the value that was popped out.
- We can remove the first element from an array using the shift() method.
- shift() method also returns the value that was shifted out
- NOTE: push() and pop() methods runs faster than unshift() and shift(). Because push() and pop() methods simply add and remove elements at the end of an array therefore the elements do not move, whereas unshift() and shift() add and remove elements at the beginning of the array that require re-indexing of whole array.

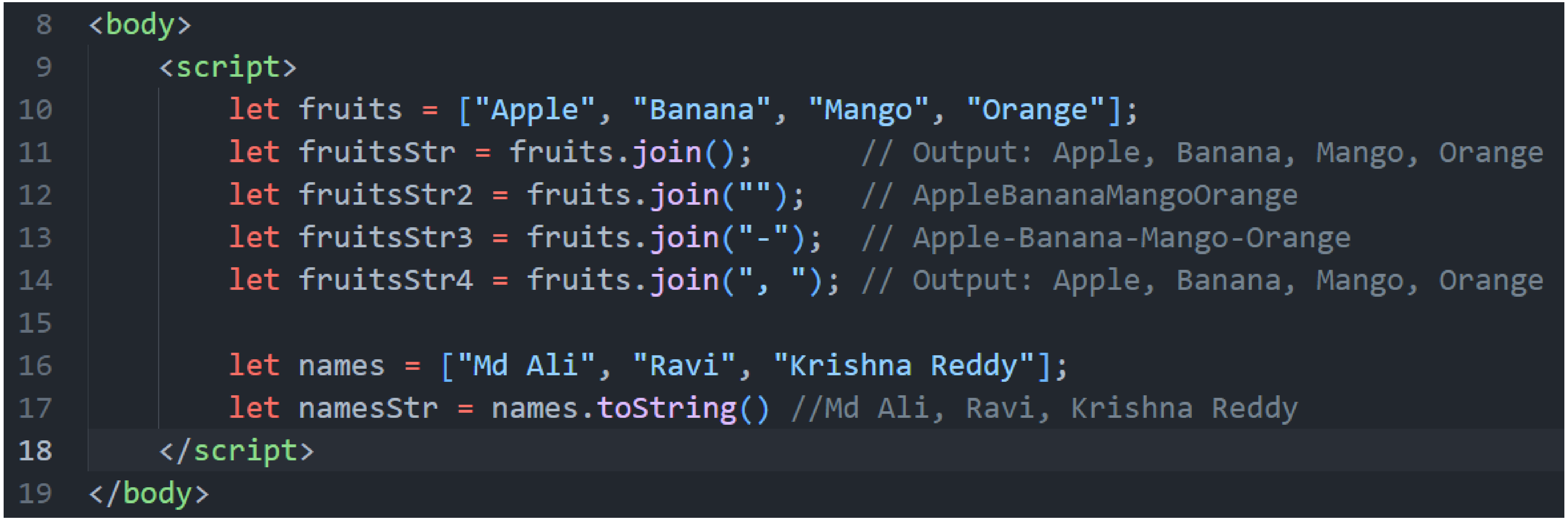
Removing Elements from Array - Demo



```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange"];
        let lastElement = fruits.pop();
        alert("Removed element = " + lastElement);
        alert("fruits.length = " + fruits.length)

        let names = ["Md Ali", "Ravi", "Krishna Reddy"];
        let firstElement = names.shift()
        alert("Removed first element = " + firstElement);
        alert("names.length = " + names.length)

        let x = [];
        //let xp = x.pop() //Undefined
        let xp = x.shift() //Undefined
        alert("xp = " + xp); //Undefined

    </script>
</body>
```

Creating a String from an Array

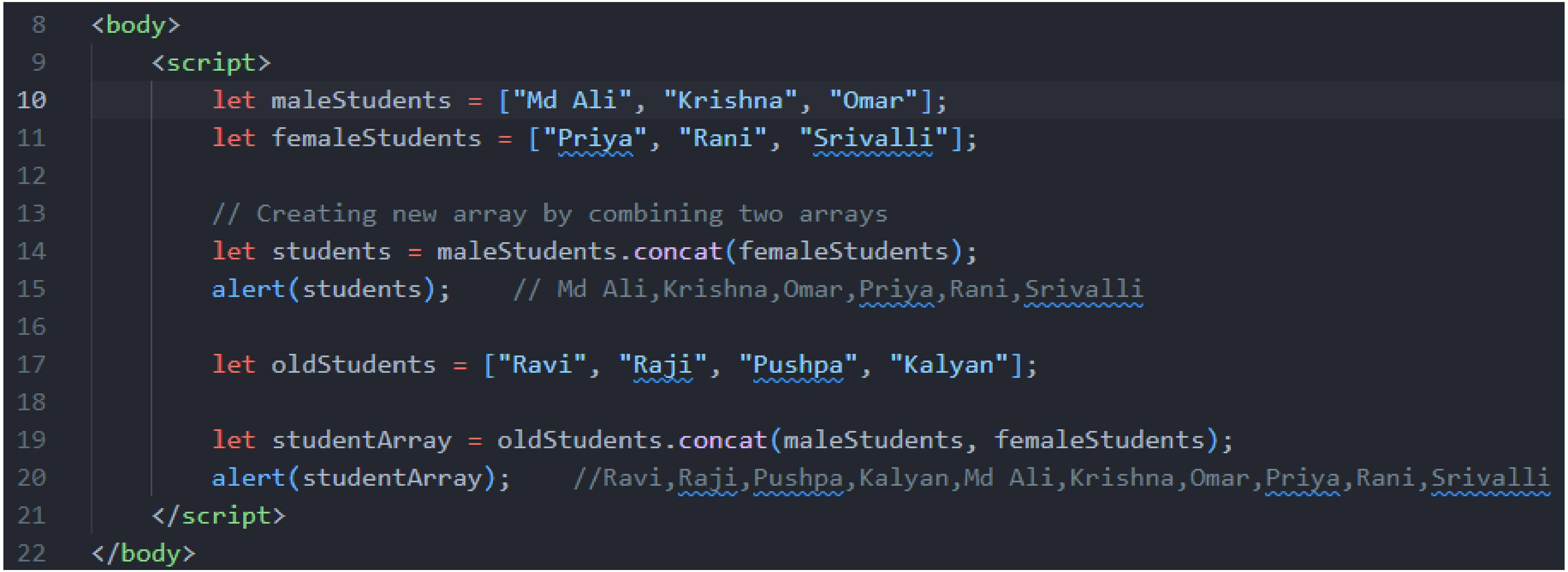
- join() method is used to joining the elements of an array and Create a string.
- join() method takes an optional parameter which is a separator string that is added in between each element.
- comma (,) is the default separator.
- We can also convert an array to a comma-separated string using the toString(). but this method does not accept the separator parameter like join()



```
<body>
    <script>
        let fruits = ["Apple", "Banana", "Mango", "Orange"];
        let fruitsStr = fruits.join();      // Output: Apple, Banana, Mango, Orange
        let fruitsStr2 = fruits.join("");   // AppleBananaMangoOrange
        let fruitsStr3 = fruits.join("-");  // Apple-Banana-Mango-Orange
        let fruitsStr4 = fruits.join(", "); // Output: Apple, Banana, Mango, Orange

        let names = ["Md Ali", "Ravi", "Krishna Reddy"];
        let namesStr = names.toString() //Md Ali, Ravi, Krishna Reddy
    </script>
</body>
```

Merging Two or More Arrays

- concat() method is used to merge two or more arrays.
- concat() method does not change the existing arrays, instead it returns a new array.
- concat() method can take any number of array arguments

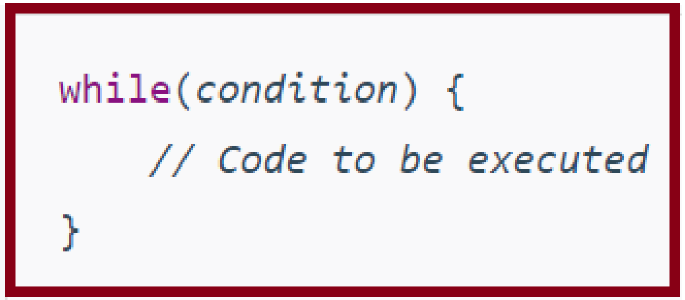


```
<body>
    <script>
        let maleStudents = ["Md Ali", "Krishna", "Omar"];
        let femaleStudents = ["Priya", "Rani", "Srivalli"];

        // Creating new array by combining two arrays
        let students = maleStudents.concat(femaleStudents);
        alert(students);    // Md Ali,Krishna,Omar,Priya,Rani,Srivalli

        let oldStudents = ["Ravi", "Raji", "Pushpa", "Kalyan"];

        let studentArray = oldStudents.concat(maleStudents, femaleStudents);
        alert(studentArray);    //Ravi,Raji,Pushpa,Kalyan,Md Ali,Krishna,Omar,Priya,Rani,Srivalli
    </script>
</body>
```

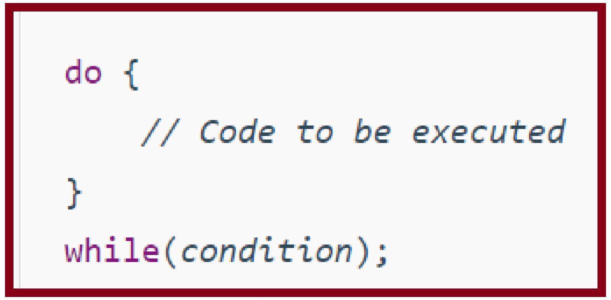
Loops in JavaScript

- Loops are used to execute the same block of code again and again, as long as a certain condition is met.
- different types of loops
  - **while -** loops through a block of code as long as the condition specified evaluates to true.
  - **do…while -** loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
  - **for -** loops through a block of code until the counter reaches a specified number.
  - **for…in -** loops through the properties of an object.
  - **for…of -** loops over iterable objects such as arrays, strings, etc.

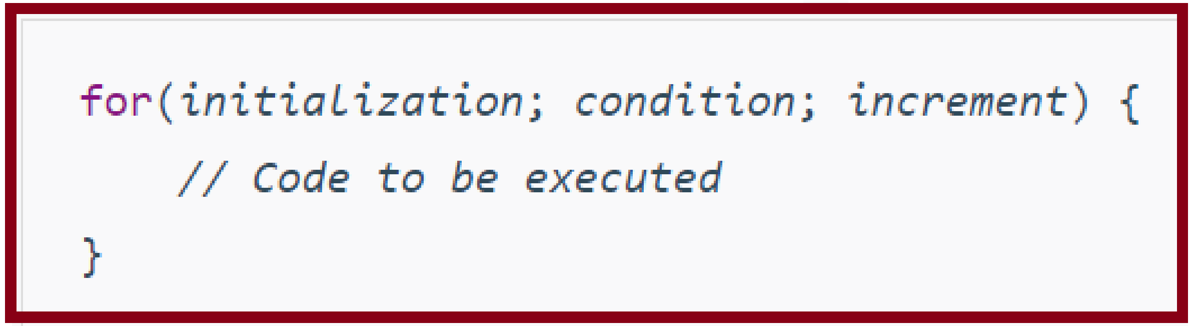
Loops Syntax



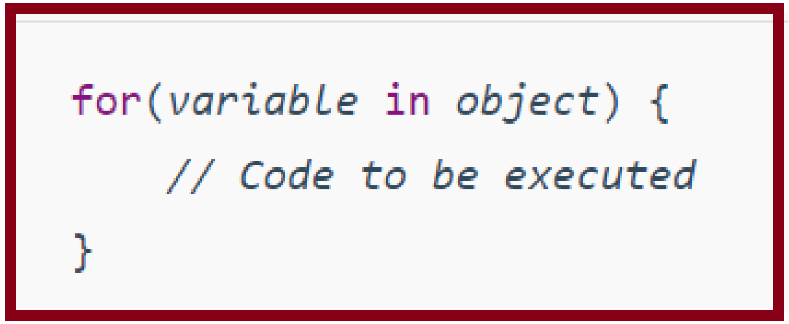
```
while(condition) {
    // Code to be executed
}
```



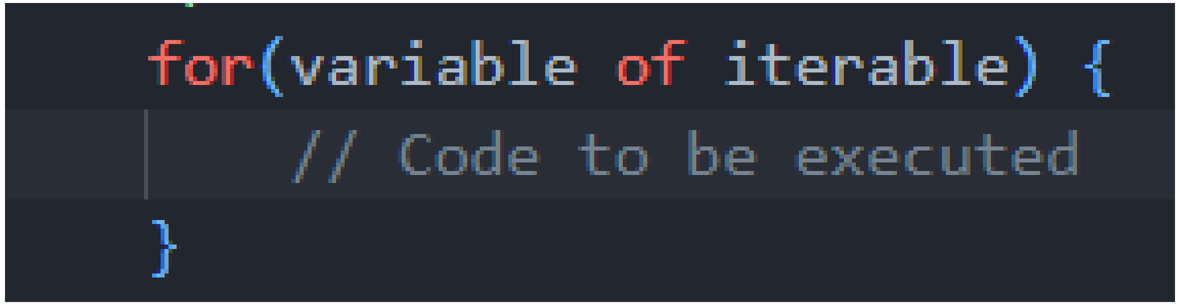
```
do {
    // Code to be executed
}
while(condition);
```



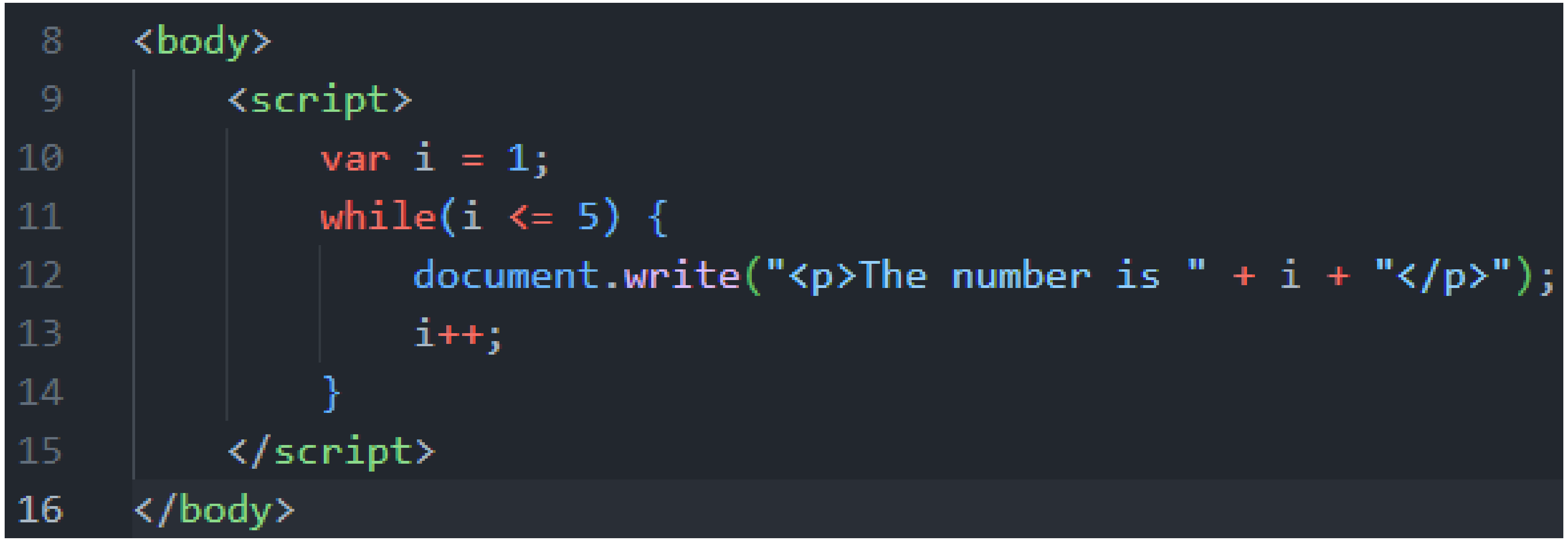
```
for(initialization; condition; increment) {
    // Code to be executed
}
```



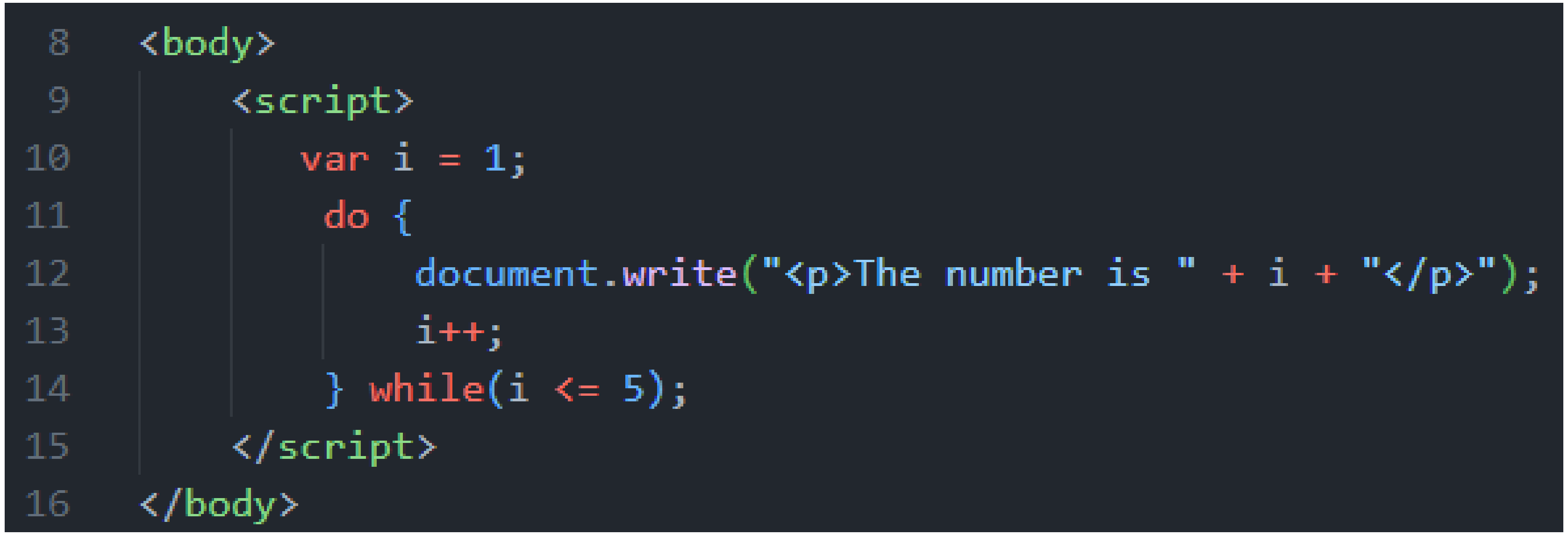
```
for(variable in object) {
    // Code to be executed
}
```



```
for(variable of iterable) {
    // Code to be executed
}
```



```
<body>
    <script>
        var i = 1;
        while(i <= 5) {
            document.write("<p>The number is " + i + "</p>");
            i++;
        }
    </script>
</body>
```



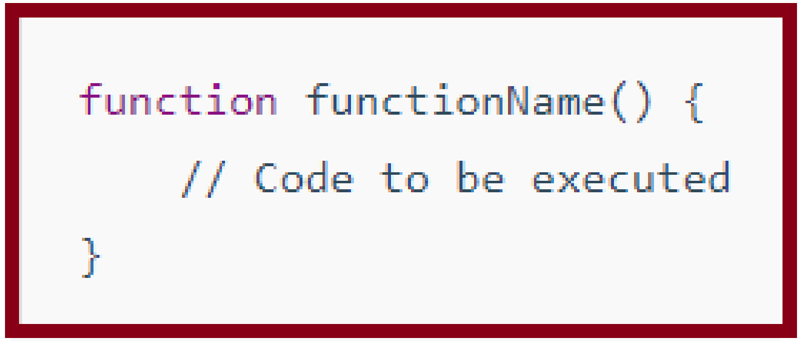
```
<body>
    <script>
       var i = 1;
        do {
            document.write("<p>The number is " + i + "</p>");
            i++;
        } while(i <= 5);
    </script>
</body>
```

JavaScript Functions

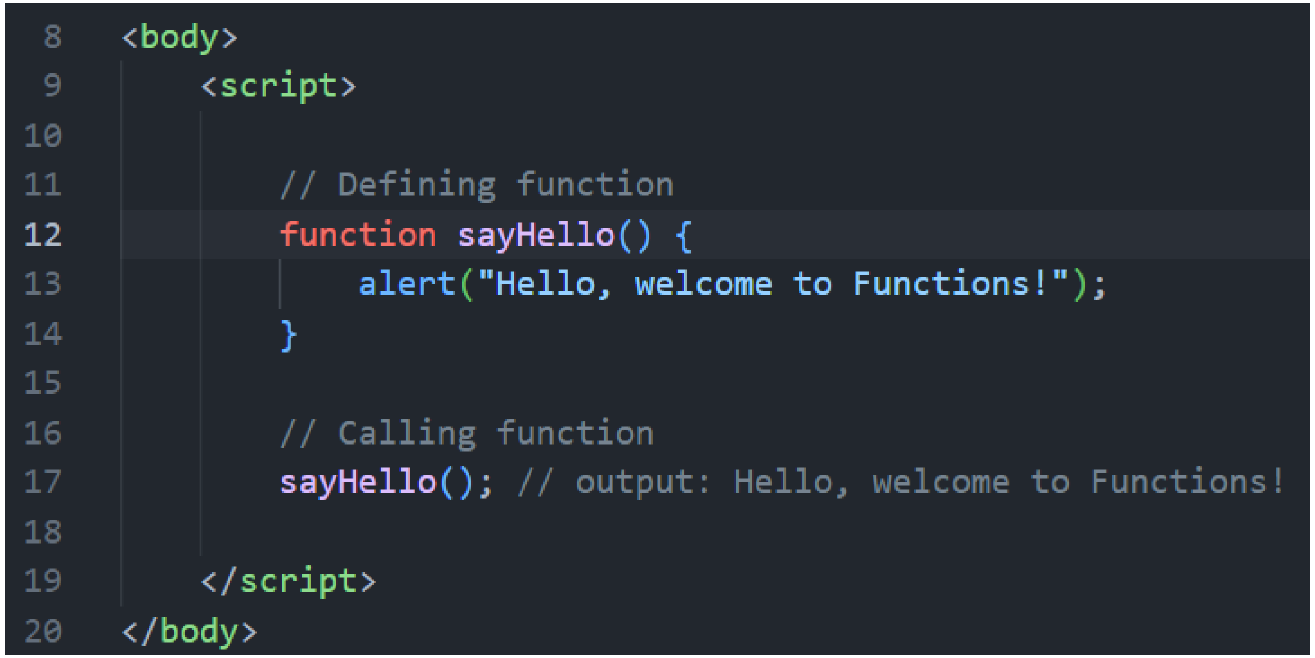
- function is a group of statements that perform specific tasks and can be kept and maintained separately form main program
- Functions provide a way to create reusable code
- Advantages of Functions
  - ✓ Functions reduces the repetition of code within a program
  - ✓ Functions makes the code much easier to maintain
  - ✓ Functions makes it easier to eliminate the errors

Defining and Calling a Function

- **function** keyword used to define a function, followed by **name** of the function, followed by **parentheses** i.e. () and we place function's **Body** between curly brackets {}



```
function functionName() {
    // Code to be executed
}
```



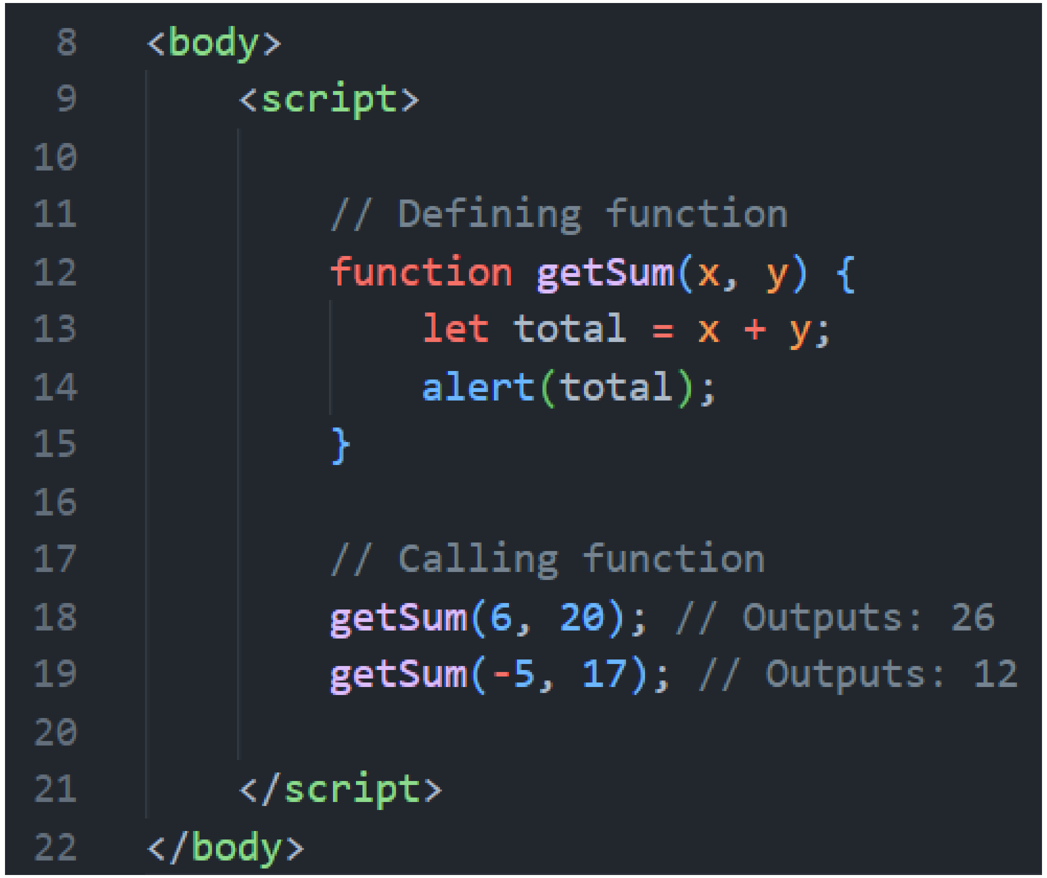
```
<body>
    <script>

        // Defining function
        function sayHello() {
            alert("Hello, welcome to Functions!");
        }

        // Calling function
        sayHello(); // output: Hello, welcome to Functions!

    </script>
</body>
```

NOTE

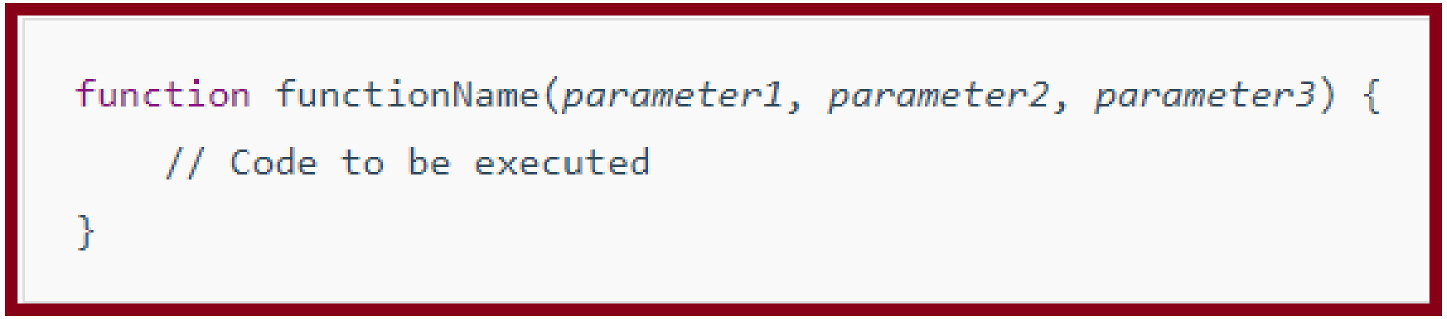
- ✓ Once a function is defined it can be called (invoked) from anywhere in the document, by typing its name followed by a set of parentheses, like sayHello()
- ✓ A function name must start with a letter or underscore character not with a number, followed by the more letters, numbers, or underscore characters.
- ✓ Function names are case sensitive, just like variable names.

Adding Parameters to Functions

- We can specify list of parameters when we define a function to accept input values at run time.
- Parameters are like placeholder variables within a function; they're replaced at run time by the values (Argument) to be provided at the time of calling function.



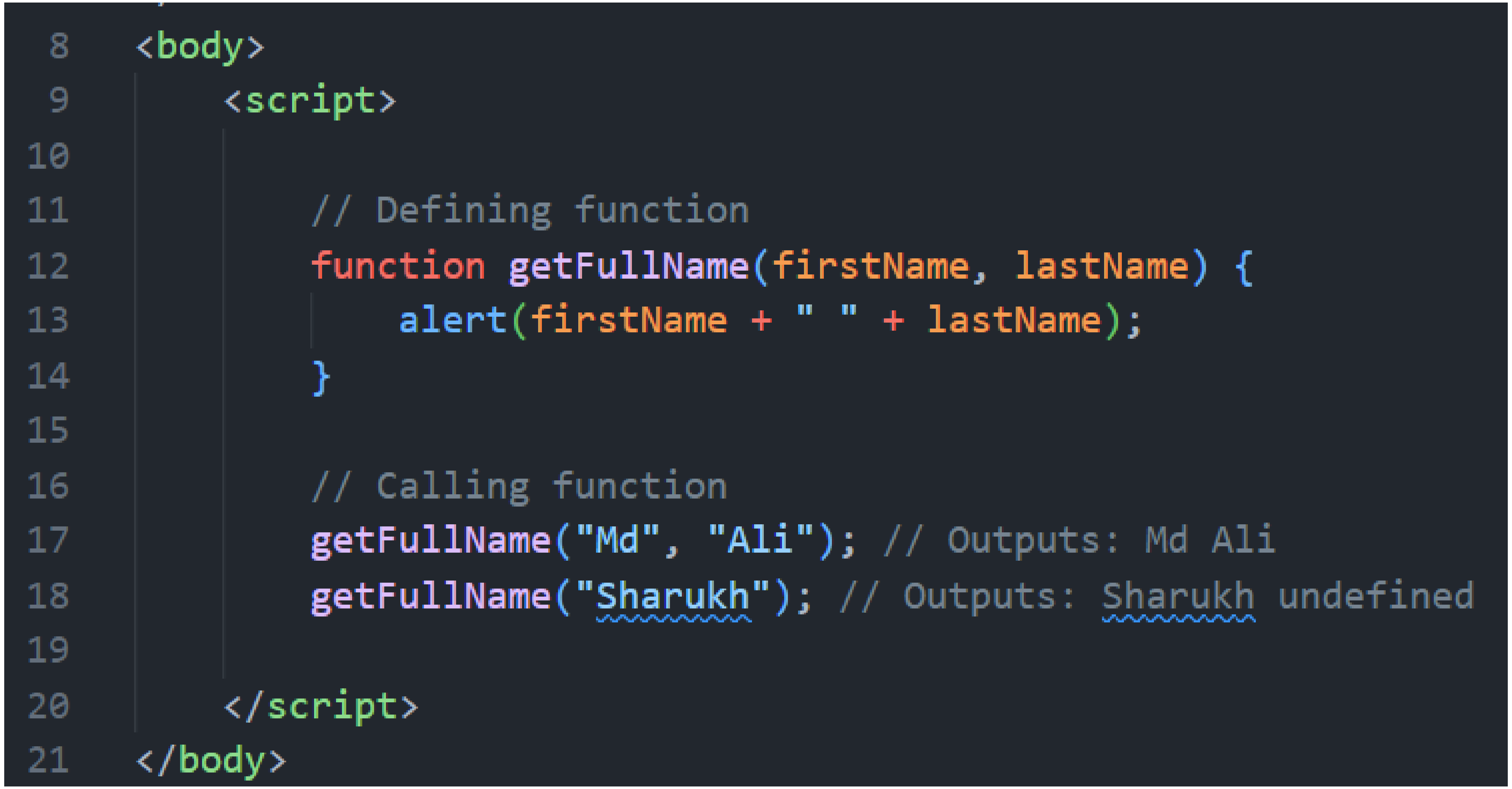
```
<body>
    <script>

        // Defining function
        function getSum(x, y) {
            let total = x + y;
            alert(total);
        }

        // Calling function
        getSum(6, 20); // Outputs: 26
        getSum(-5, 17); // Outputs: 12

    </script>
</body>
```



```
function functionName(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

NOTE

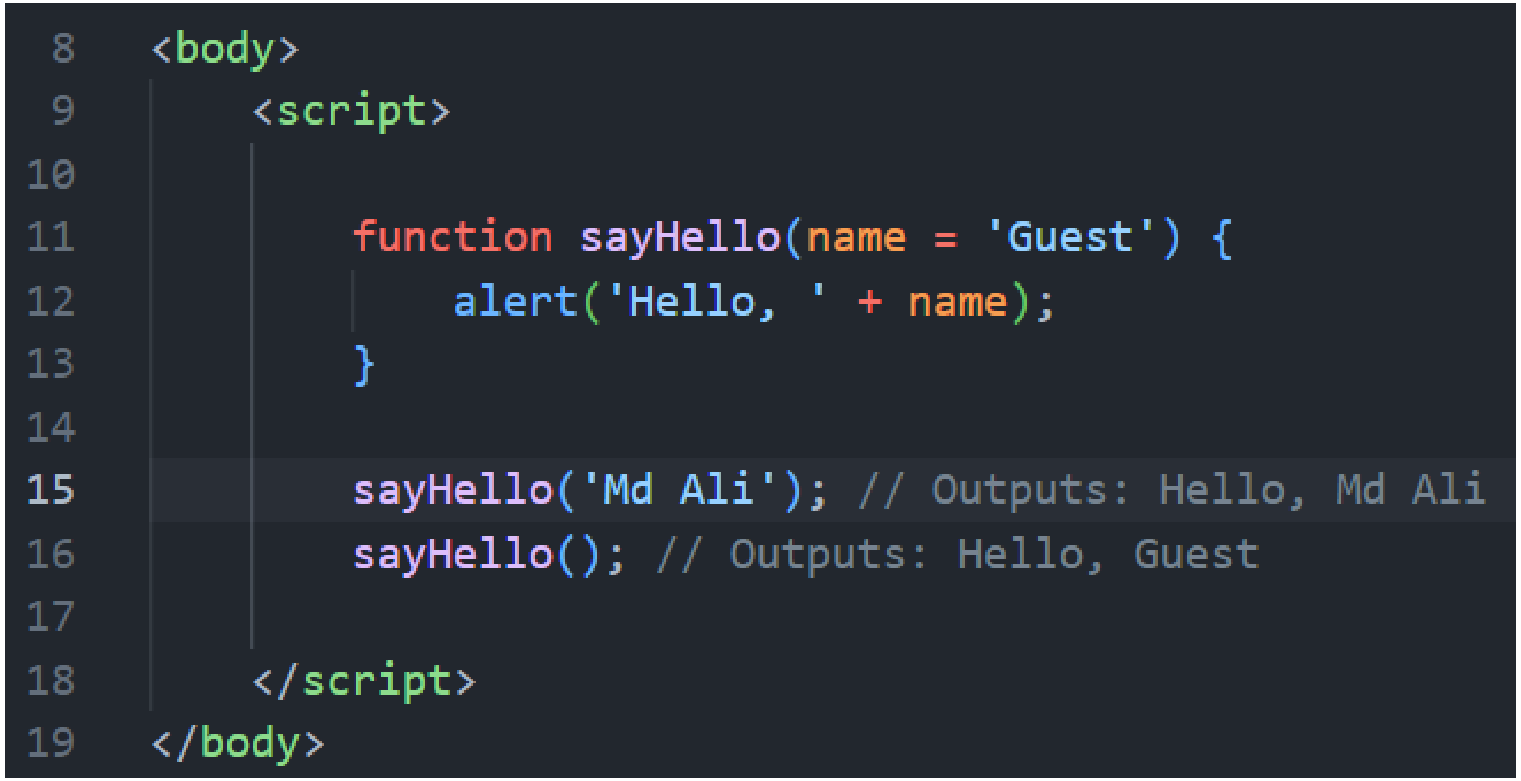
- We can define as many parameters as we like.
- For each parameter we specify, a corresponding argument needs to be passed to the function when it is called, otherwise its value becomes undefined.



```
<body>
    <script>

        // Defining function
        function getFullName(firstName, lastName) {
            alert(firstName + " " + lastName);
        }

        // Calling function
        getFullName("Md", "Ali"); // Outputs: Md Ali
        getFullName("Sharukh"); // Outputs: Sharukh undefined

    </script>
</body>
```

Default Values for Function Parameters (ES6)

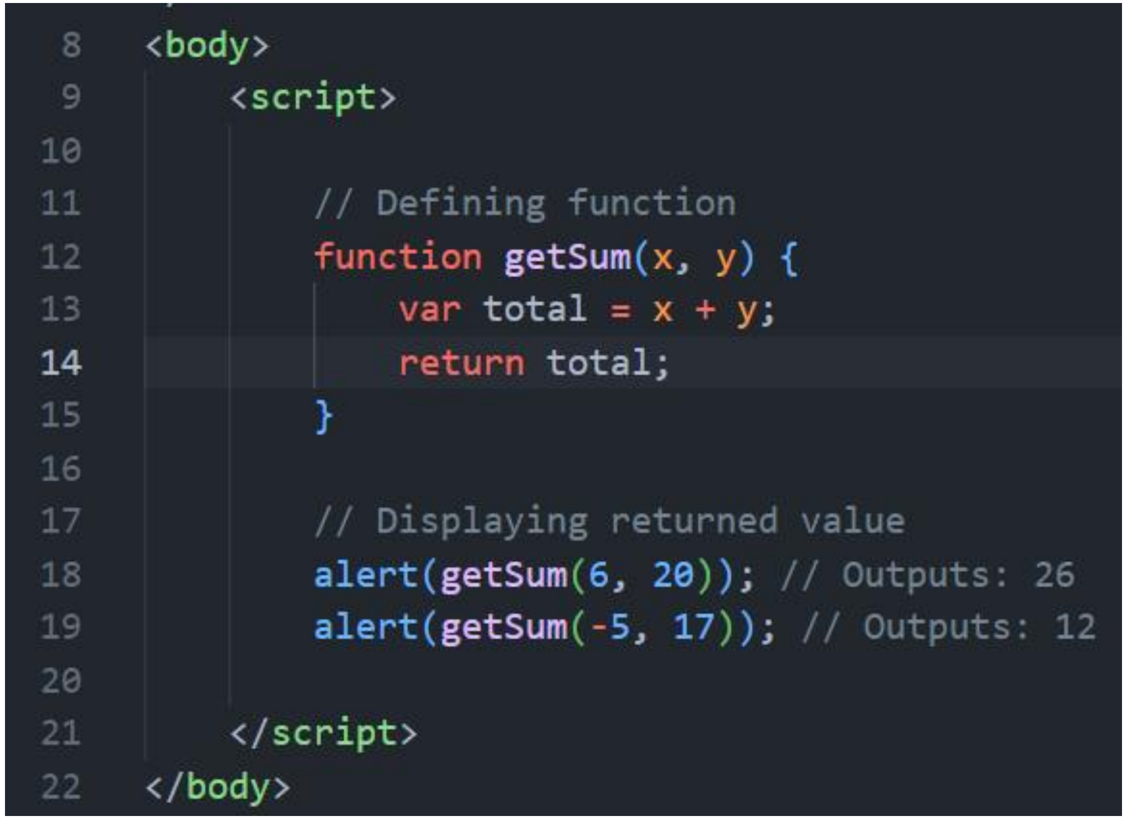
- With ES6, now we can specify default values to the function parameters. This means that if no arguments are provided to function when it is called these default parameters values will be used.
- This is one of the most awaited features in JavaScript.



```
<body>
    <script>

        function sayHello(name = 'Guest') {
            alert('Hello, ' + name);
        }

        sayHello('Md Ali'); // Outputs: Hello, Md Ali
        sayHello(); // Outputs: Hello, Guest

    </script>
</body>
```

Returning Values from a Function

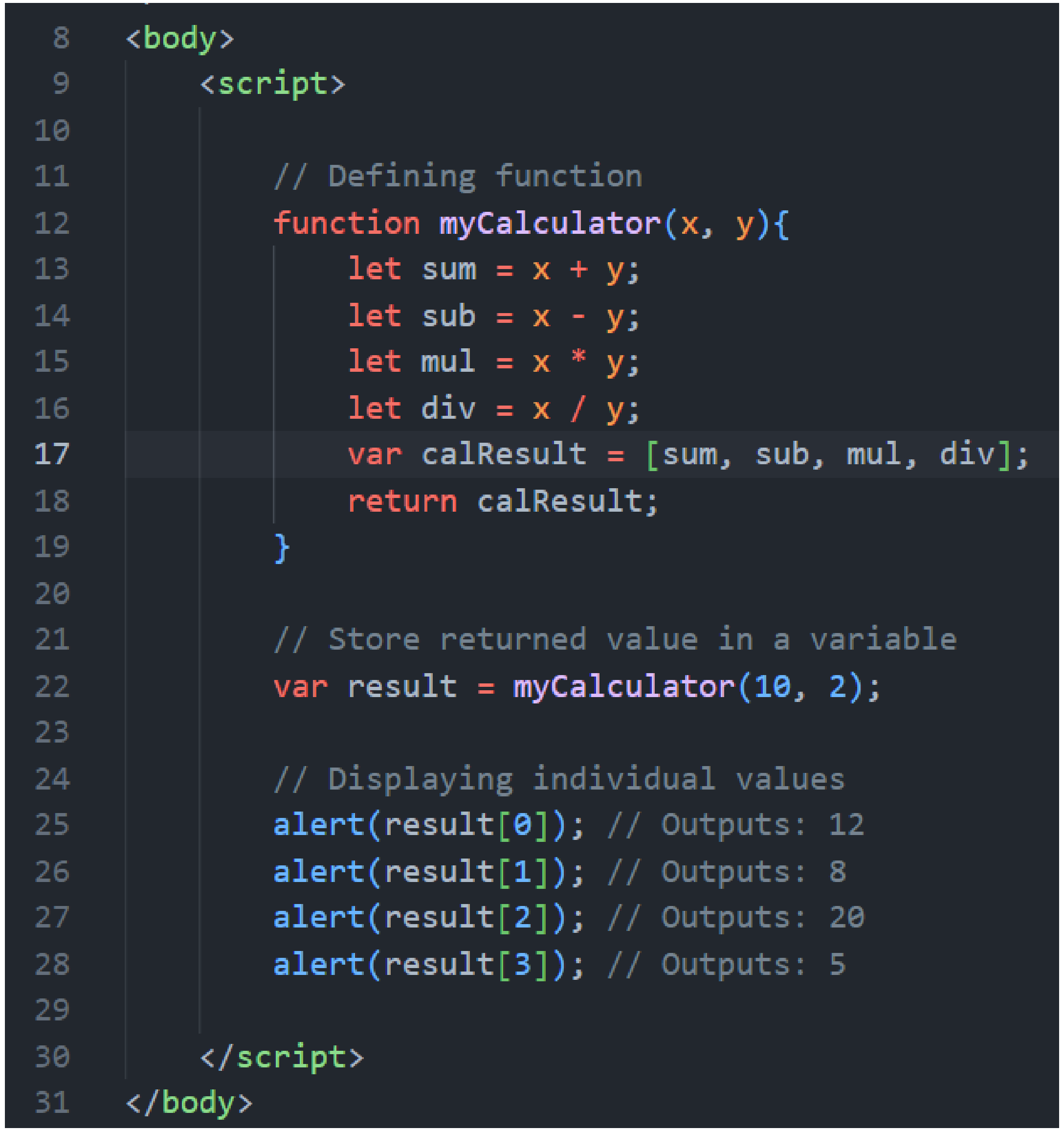
- A function can return a value back to the script that called the function as a result using the return statement.
- The value may be of any type, including arrays and objects.
- The return statement usually placed as the last line of the function before the closing curly bracket and ends it with a semicolon



```
<body>
    <script>

        // Defining function
        function getSum(x, y) {
            var total = x + y;
            return total;
        }

        // Displaying returned value
        alert(getSum(6, 20)); // Outputs: 26
        alert(getSum(-5, 17)); // Outputs: 12

    </script>
</body>
```

NOTE

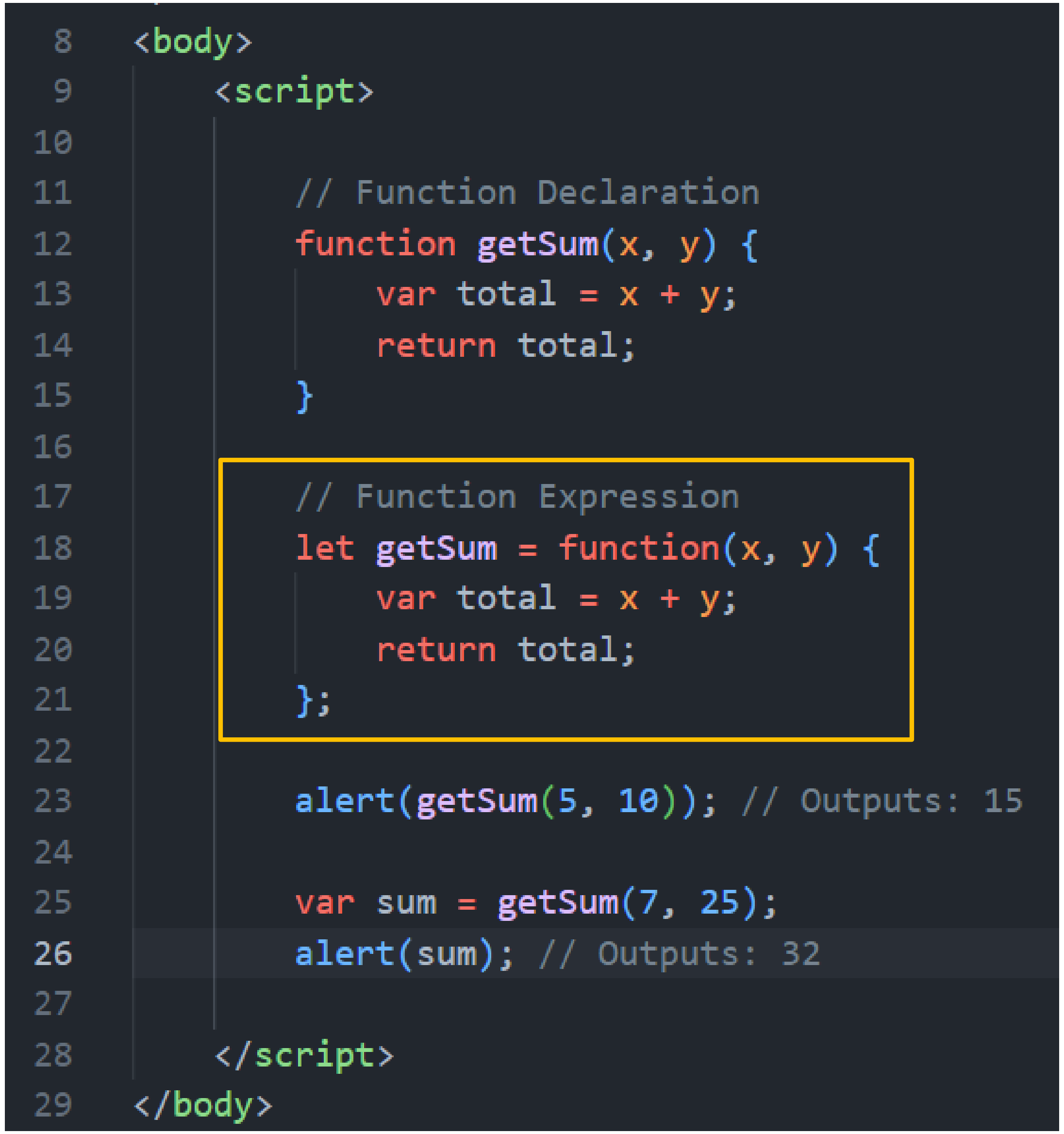
- A function **CAN NOT** return multiple values. But we can obtain similar results by returning an array of values



```
<body>
    <script>

        // Defining function
        function myCalculator(x, y){
            let sum = x + y;
            let sub = x - y;
            let mul = x * y;
            let div = x / y;
            var calResult = [sum, sub, mul, div];
            return calResult;
        }

        // Store returned value in a variable
        var result = myCalculator(10, 2);

        // Displaying individual values
        alert(result[0]); // Outputs: 12
        alert(result[1]); // Outputs: 8
        alert(result[2]); // Outputs: 20
        alert(result[3]); // Outputs: 5

    </script>
</body>
```

Function Expressions

- Another way of creating a function is called a function expression.



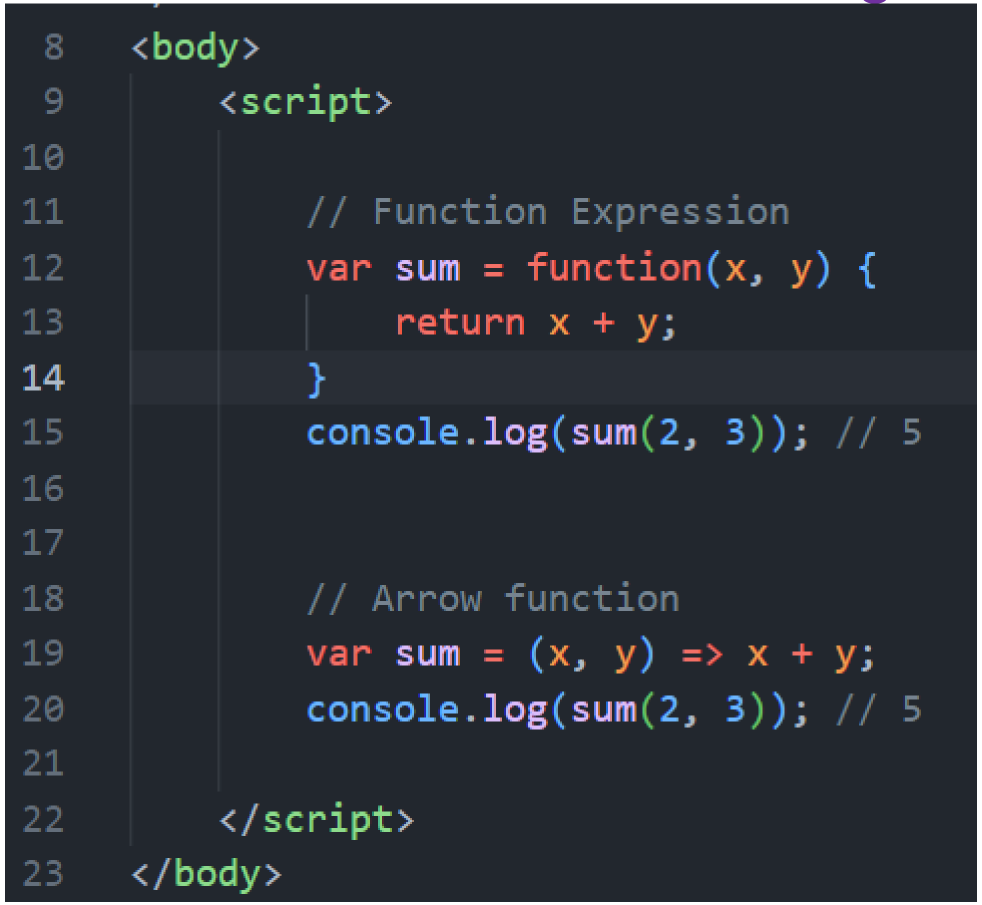
```
<body>
    <script>

        // Function Declaration
        function getSum(x, y) {
            var total = x + y;
            return total;
        }

        // Function Expression
        let getSum = function(x, y) {
            var total = x + y;
            return total;
        };

        alert(getSum(5, 10)); // Outputs: 15

        var sum = getSum(7, 25);
        alert(sum); // Outputs: 32

    </script>
</body>
```

NOTE:

✓ There is no need to put a semicolon after the closing curly bracket in a function declaration. But function expressions, on the other hand, should always end with a semicolon.

Arrow Functions

- Arrow Functions are another feature in ES6.
- It provides a more concise syntax for writing function expressions without using function and return keywords.
- Arrow functions are defined using a new syntax, the fat arrow (=>) notation

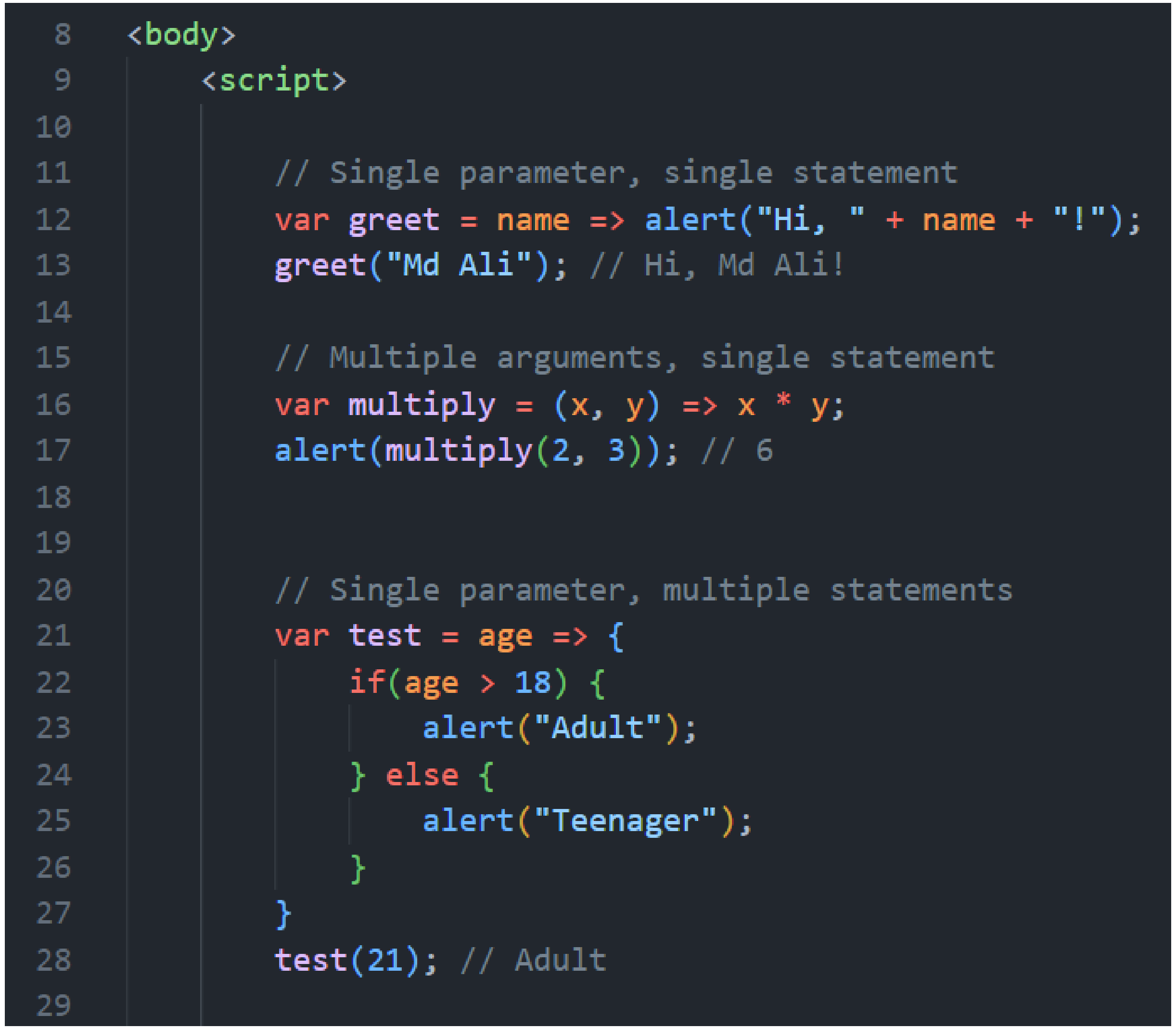


```
<body>
    <script>

        // Function Expression
        var sum = function(x, y) {
            return x + y;
        }
        console.log(sum(2, 3)); // 5


        // Arrow function
        var sum = (x, y) => x + y;
        console.log(sum(2, 3)); // 5

    </script>
</body>
```

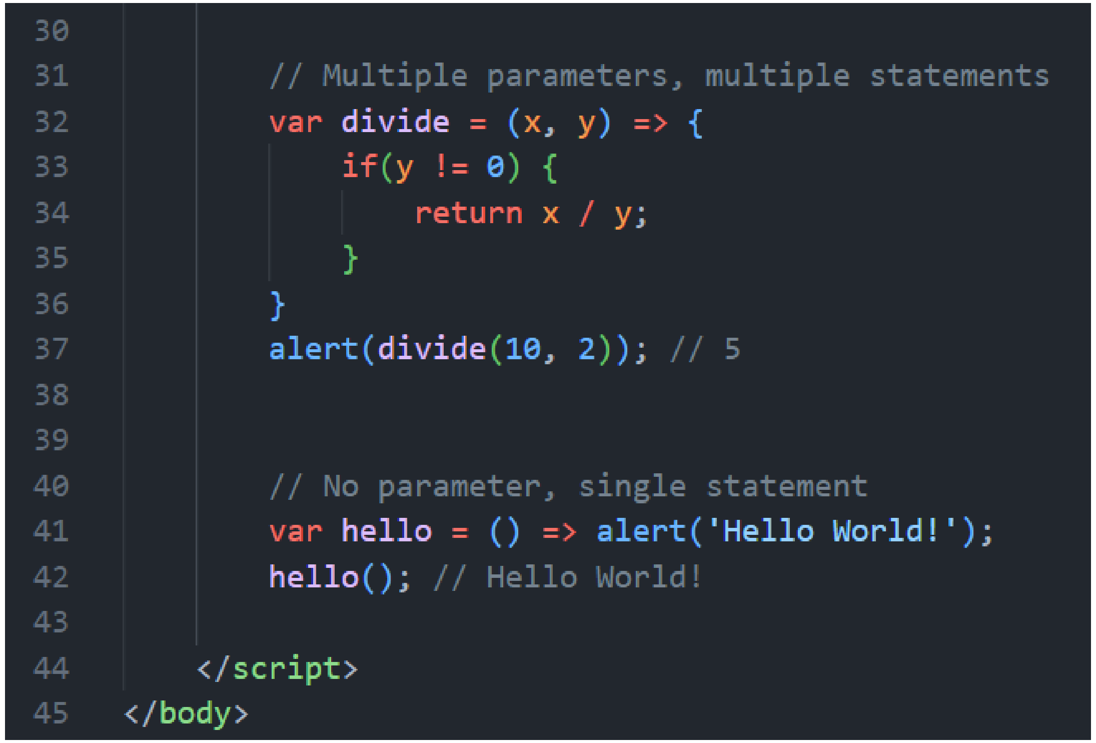
NOTEs

- ✓ There is no function and return keyword in arrow function declaration.
- ✓ We can also skip the parentheses i.e. () in case when there is exactly one parameter, but we always need to use parentheses when we have zero or more than one parameter.
- ✓ if there's more than one expression in the function body, we need to wrap it in braces ({}). In this case we also need to use the return statement to return a value.

Different ways Of defining arrow functions



```
<body>
    <script>

        // Single parameter, single statement
        var greet = name => alert("Hi, " + name + "!");
        greet("Md Ali"); // Hi, Md Ali!

        // Multiple arguments, single statement
        var multiply = (x, y) => x * y;
        alert(multiply(2, 3)); // 6


        // Single parameter, multiple statements
        var test = age => {
            if(age > 18) {
                alert("Adult");
            } else {
                alert("Teenager");
            }
        }
        test(21); // Adult

```



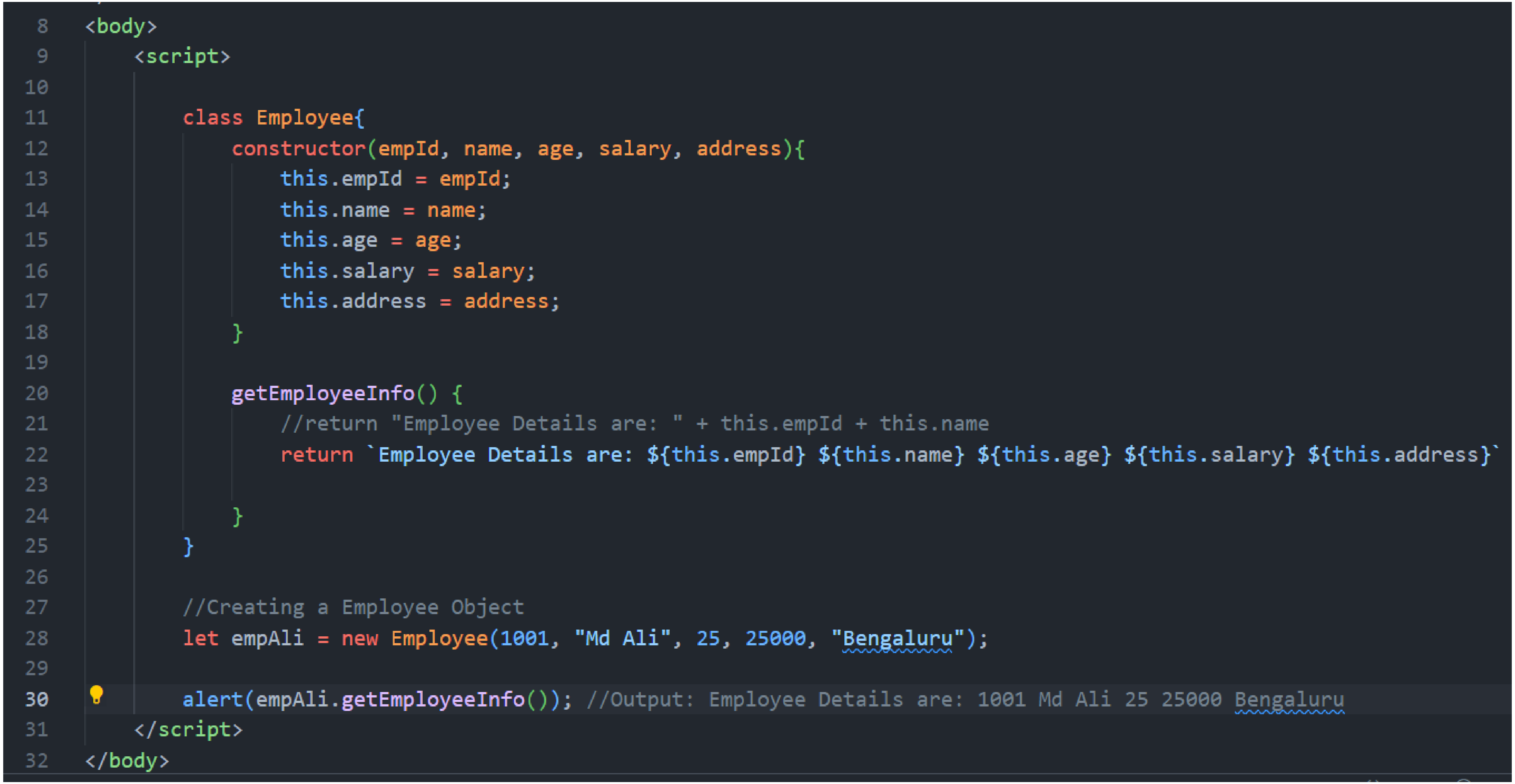
```

        // Multiple parameters, multiple statements
        var divide = (x, y) => {
            if(y != 0) {
                return x / y;
            }
        }
        alert(divide(10, 2)); // 5


        // No parameter, single statement
        var hello = () => alert('Hello World!');
        hello(); // Hello World!

    </script>
</body>
```

JavaScript Classes

- Classes were introduced in EcmaScript 2015 (ES6) to provide a cleaner way to follow object-oriented programming patterns.
- JavaScript class is a blueprint used to create objects in Javascript.
- Syntax

```
class Classname {
    constructor() { ... }
}
```

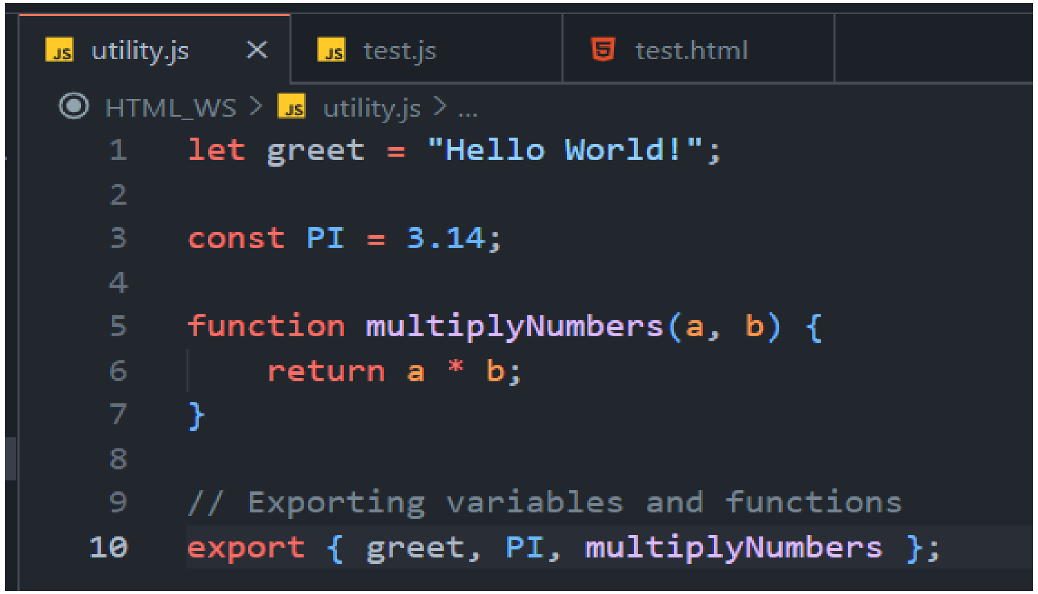
- The class keyword is used to create classes in javascript.
- The Classname is the name of the class. It is user-defined. (Conventional method is to start the class name with a capital letter).
- The constructor() is a method used to initialize the object properties. It is executed automatically when a new object of the given class is created. (It may take parameters).



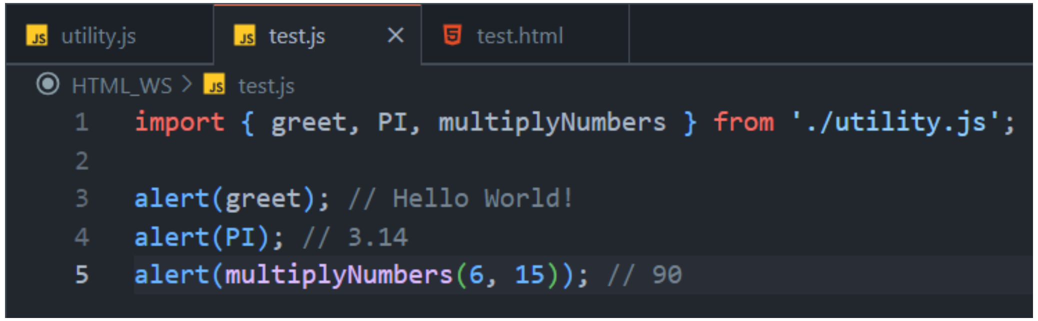
```
<body>
    <script>

        class Employee{
            constructor(empId, name, age, salary, address){
                this.empId = empId;
                this.name = name;
                this.age = age;
                this.salary = salary;
                this.address = address;
            }

            getEmployeeInfo() {
                //return "Employee Details are: " + this.empId + this.name
                return `Employee Details are: ${this.empId} ${this.name} ${this.age} ${this.salary} ${this.address}`

            }
        }

        //Creating a Employee Object
        let empAli = new Employee(1001, "Md Ali", 25, 25000, "Bengaluru");

        alert(empAli.getEmployeeInfo()); //Output: Employee Details are: 1001 Md Ali 25 25000 Bengaluru
    </script>
</body>
```

Modules

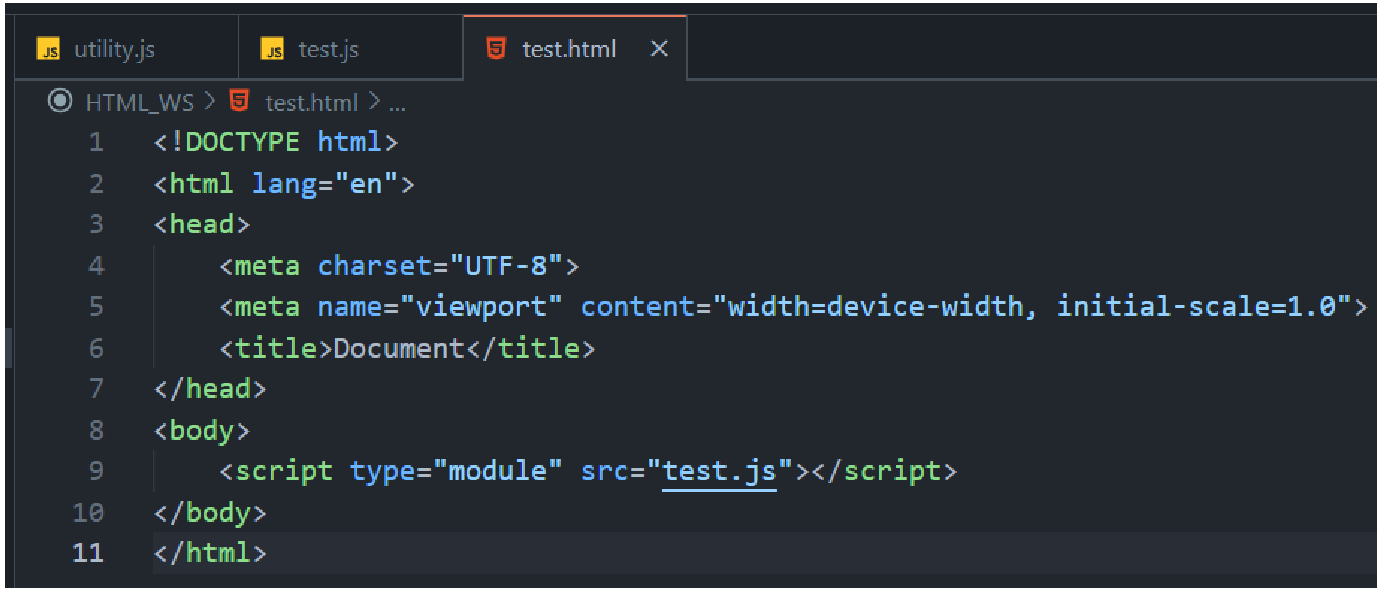
- Prior to ES6, there were no native support for modules in JavaScript.
- ES6 introduces file based module, each module represented by a .js file. We can use export or import statement in a module to export or import variables, functions, classes or any other entity to/from other modules or files.



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

const PI = 3.14;

function multiplyNumbers(a, b) {
    return a * b;
}

// Exporting variables and functions
export { greet, PI, multiplyNumbers };
```



```
import { greet, PI, multiplyNumbers } from './utility.js';

alert(greet); // Hello World!
alert(PI); // 3.14
alert(multiplyNumbers(6, 15)); // 90
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="module" src="test.js"></script>
</body>
</html>
```