UNIT-2
JAVASCRIPT

J Vishnu Priyanka
Assistant Professor©
Dept of CSE
RGUKT-SRIKAKULAM.

INTRODUCTION

- **JavaScript is the premier client-side *interpreted scripting language.***
- Netscape initially introduced the language under the name **LiveScript in an early beta release of Navigator 2.0 in 1995, and the focus was on form validation.**
- **After that, the language was renamed JavaScript.**
- **After Netscape introduced JavaScript in version 2.0 of their browser, Microsoft introduced a clone of JavaScript called JScript in Internet Explorer 3.0.**

JavaScript gives web developers a programming language for use in web pages & allows them to do the following:

JavaScript gives HTML designers a programming tool

JavaScript can be used to validate data

JavaScript can read and write HTML elements

Create pop-up windows

Perform mathematical calculations on data

React to events, such as a user rolling over an image or clicking a button

Retrieve the current date and time from a user's computer or the last time a document was modified

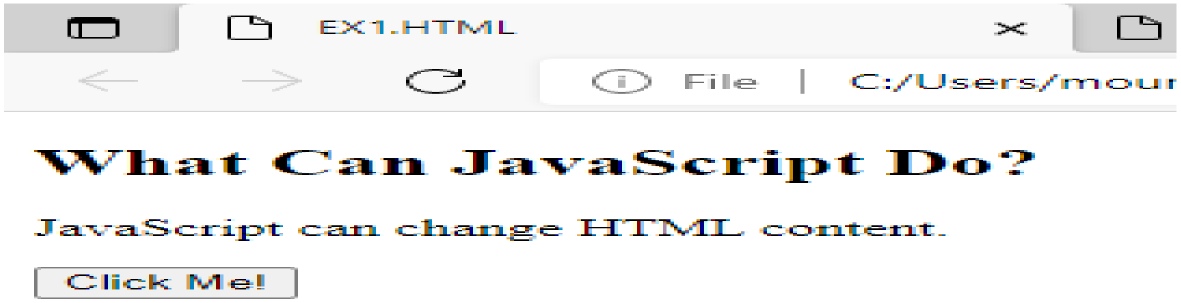
Determine the user's screen size, browser version, or screen resolution

JavaScript can put dynamic text into an HTML page

JavaScript can be used to create cookies

Changing the Elements:

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
  onclick='document.getElementById("demo").innerHTML =
  "Hello JavaScript!"'>Click Me!</button>

</body>
</html>
```

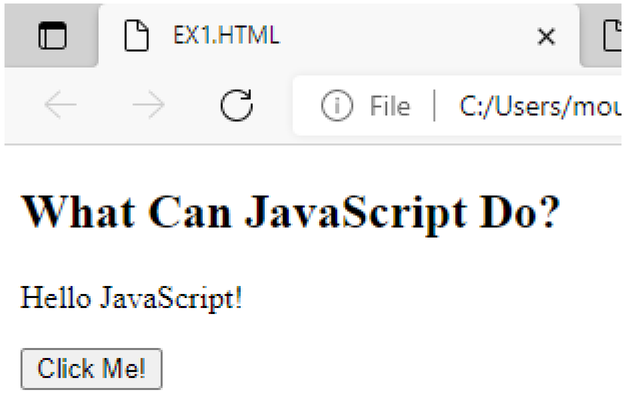


EX1.HTML

File | C:/Users/mour

What Can JavaScript Do?

JavaScript can change HTML content.



EX1.HTML

File | C:/Users/mou

What Can JavaScript Do?

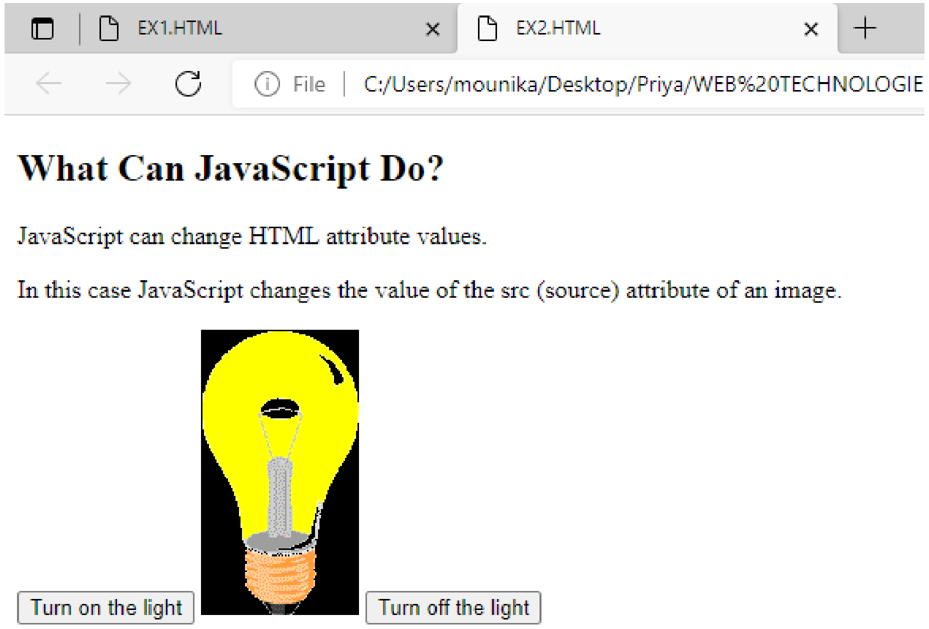
Hello JavaScript!

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.PNG'">Turn on the
  light</button>

<img id="myImage" src="pic_bulboff.PNG" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.PNG'">Turn off the
  light</button>

</body>
</html>
```

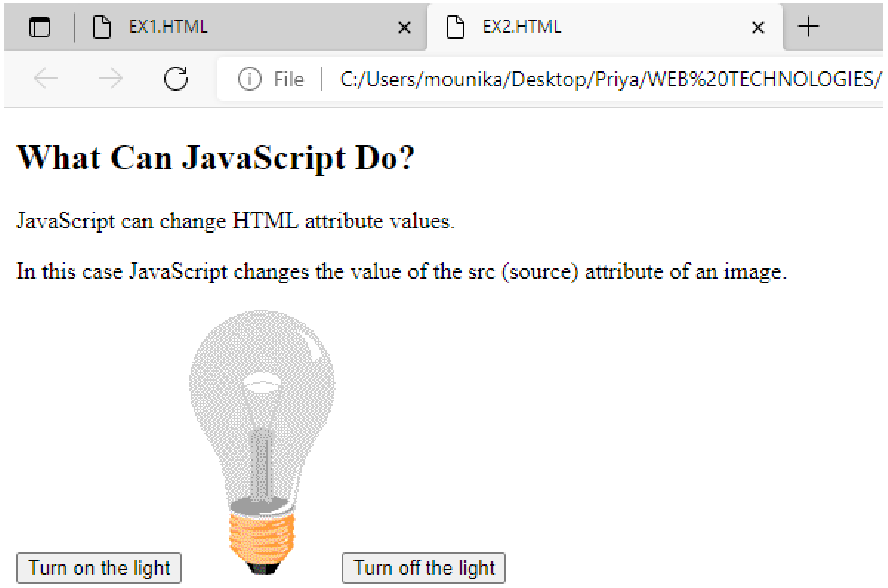


EX1.HTML
EX2.HTML
File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIE

What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



EX1.HTML
EX2.HTML
File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIES/

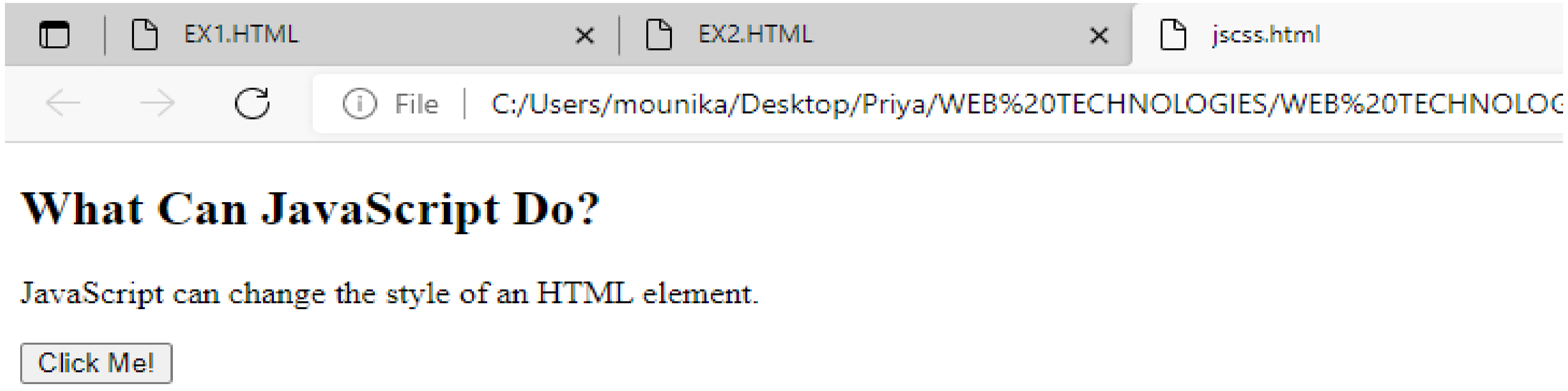
What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

JavaScript Can Change HTML Styles (CSS)

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change the style of an
  HTML element.</p>

<button type="button"
  onclick="document.getElementById('demo').style.fontS
  ize='35px'">Click Me!</button>

</body>
</html>
```



EX1.HTML

EX2.HTML

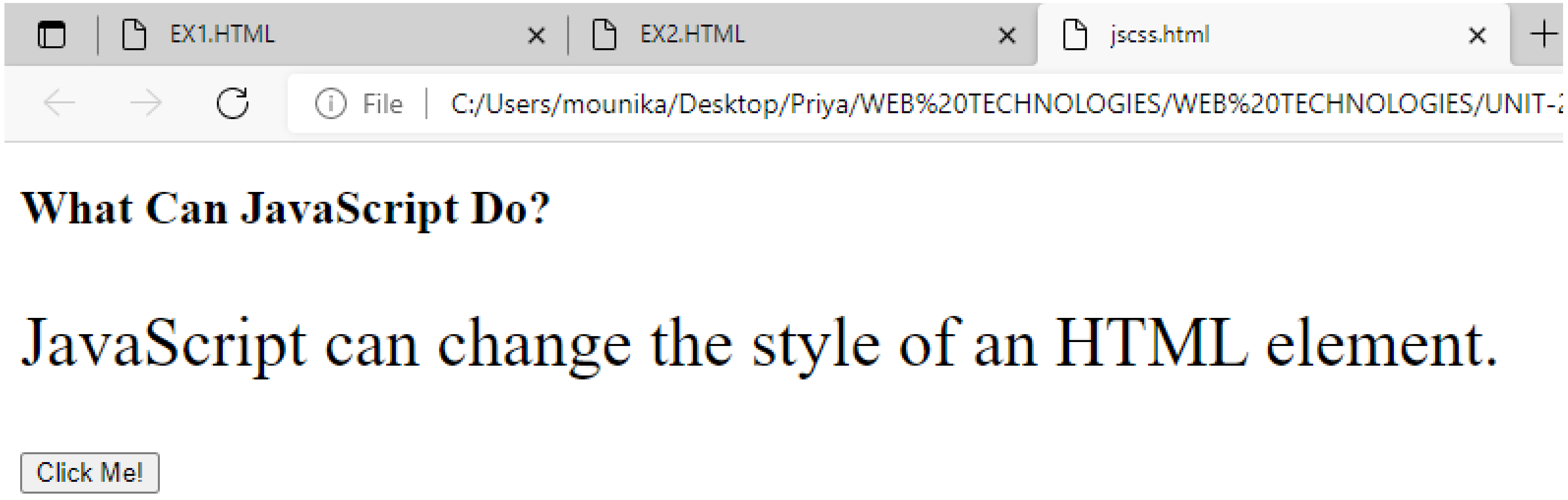
jscss.html

File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIES/WEB%20TECHNOLOG

What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!



EX1.HTML

EX2.HTML

jscss.html

File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIES/WEB%20TECHNOLOGIES/UNIT-

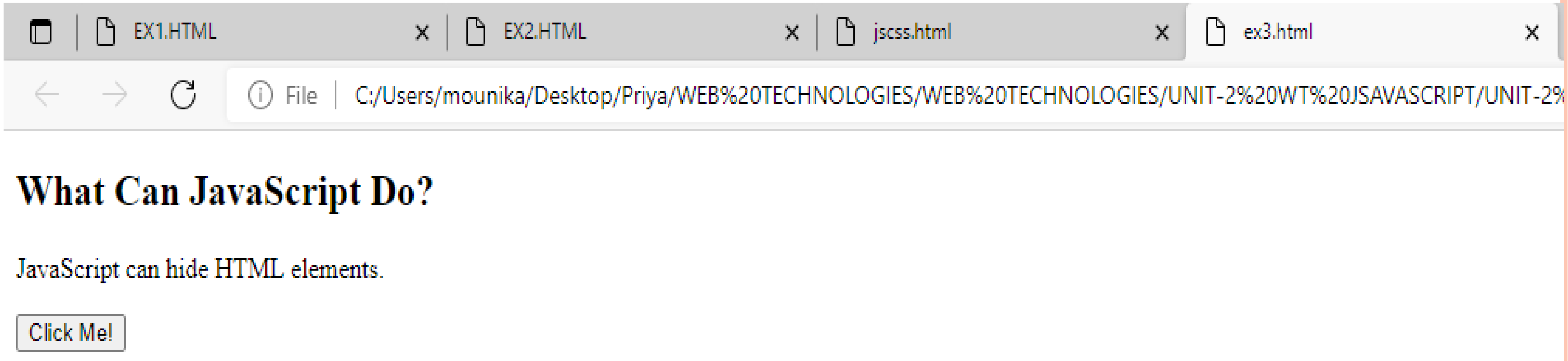
What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!

JavaScript Can Hide HTML Elements

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button"
   onclick="document.getElementById('demo').style.display='
   none'">Click Me!</button>

</body>
</html>
```



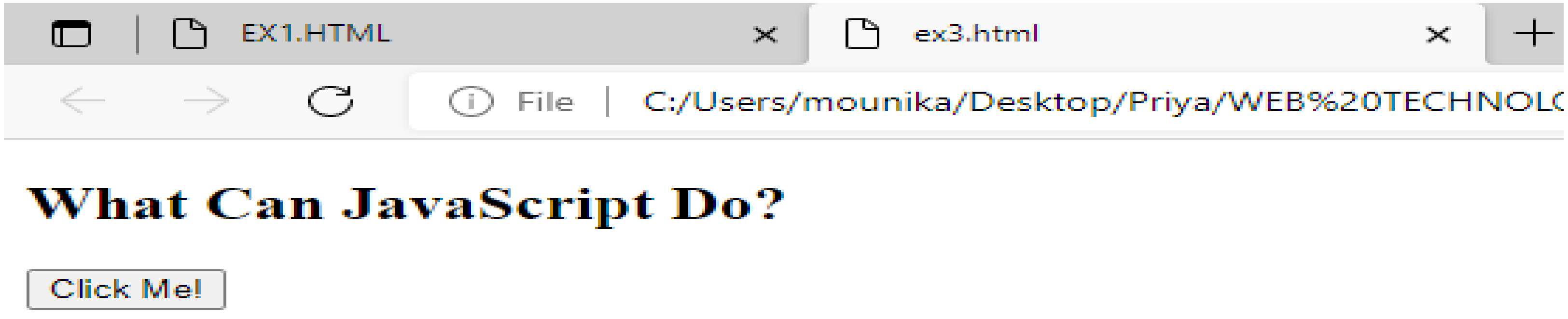
EX1.HTML | EX2.HTML | jscss.html | ex3.html

File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIES/WEB%20TECHNOLOGIES/UNIT-2%20WT%20JSAVASCRIPT/UNIT-2%

What Can JavaScript Do?

JavaScript can hide HTML elements.

Click Me!



EX1.HTML | ex3.html

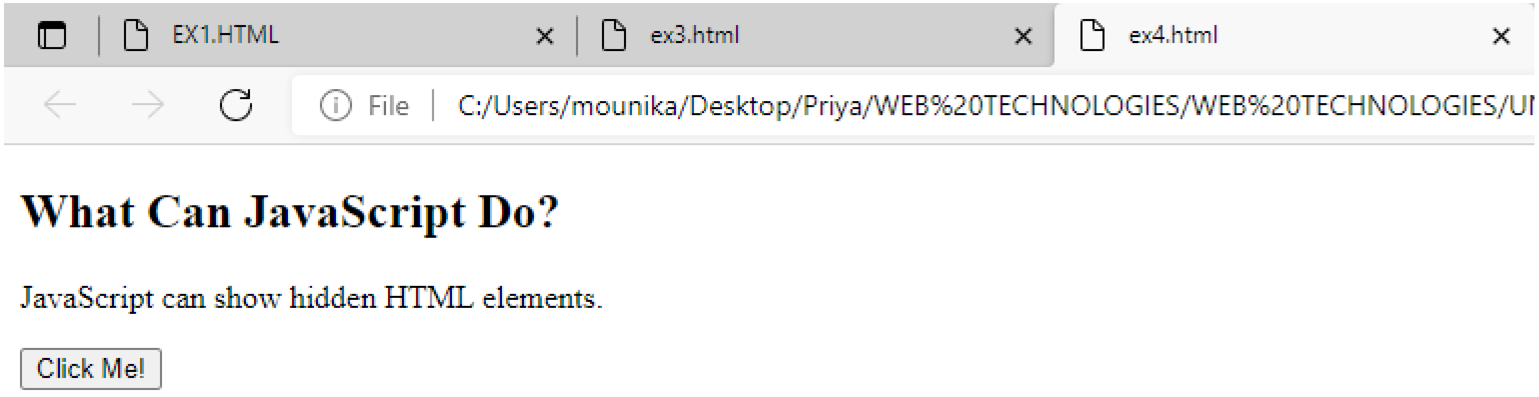
File | C:/Users/mounika/Desktop/Priya/WEB%20TECHNOL

What Can JavaScript Do?

Click Me!

JavaScript Can Show HTML Elements

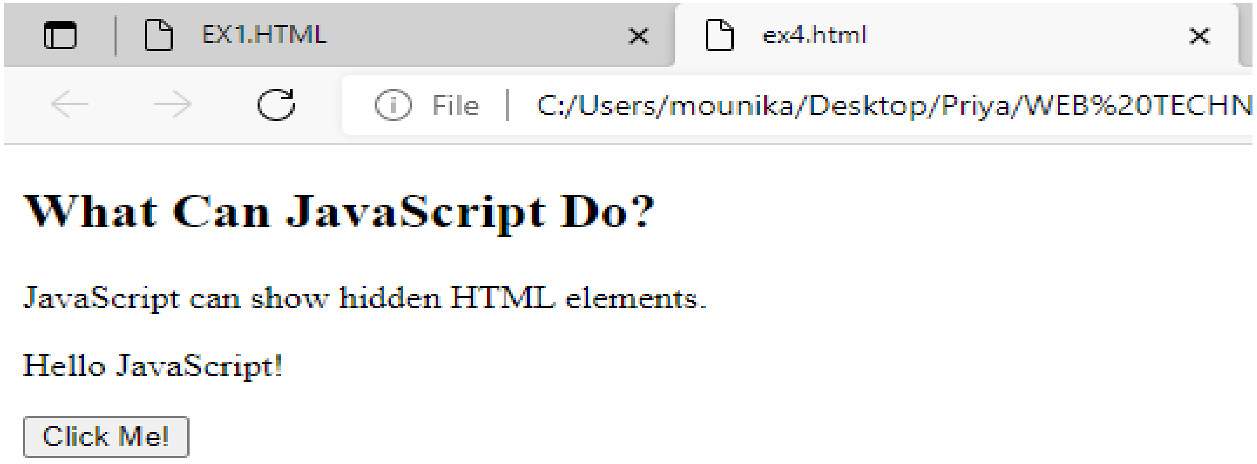
```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button"
   onclick="document.getElementById('demo').style.display='block'">
   Click Me!</button>

</body>
</html>
```



EX1.HTML
ex3.html
ex4.html
File
C:/Users/mounika/Desktop/Priya/WEB%20TECHNOLOGIES/WEB%20TECHNOLOGIES/UI

What Can JavaScript Do?

JavaScript can show hidden HTML elements.



What Can JavaScript Do?

JavaScript can show hidden HTML elements.

Hello JavaScript!

The <script> Tag

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "My First
   JavaScript";
</script>

</body>
</html>
```

JavaScript in <head>

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

JavaScript Output

- JavaScript can "display" data in different ways:
- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Example:

```
<!DOCTYPE html>
    <html>
    <body>

    <h1>My First Web Page</h1>
    <p>My First Paragraph</p>

    <p id="demo"></p>

    <script>
    document.getElementById("demo").innerHTML = 5 +
    6;
    </script>

    </body>
    </html>
```

Using document.write()

For testing purposes, it is convenient to use document.write():

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.
It will overwrite the whole document.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

JavaScript Statements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a computer.</p>

<p id="demo"></p>

<script>
var x, y, z;  // Statement 1
x = 5;    // Statement 2
y = 6;    // Statement 3
z = x + y;  // Statement 4

document.getElementById("demo").innerHTML =
"The value of z is " + z + ".";
</script>

</body>
</html>
```

Advantages of JavaScript:

Less server interaction

Immediate feedback to the visitors

Increased interactivity

Java Script is relatively secure.

Points to remember:

JavaScript is case-sensitive

Each line of code is terminated by a semicolon

Variables are declared using the keyword **var**

JavaScript and HTML Page

- Having written some JavaScript, we need to include it in an HTML page.
- We can"t execute these scripts from a command line, as the interpreter is part of the browser.
- The script is included in the web page and run by the browser, usually as soon as the page has been loaded.
- The browser is able to debug the script and can display errors.

Embedding JavaScript in HTML file:

- If we are writing small scripts, or only use our scripts in few pages, then the easiest way is to include the script in the HTML code. The syntax is shown below:

```
<html>
   <head>
        <script language=”javascript”>
<!- -
Javascript code here
//- - >
   </head>
   <body>
……
   </body>
</html>
```

Using External JavaScript in HTML file:

- If we use lot of scripts, or our scripts are complex then including the code inside the web page will make the source file difficult to read and debug.
- A better idea is to put the JavaScript code in a separate file and include that code in the HTML file.
- By convention, JavaScript programs are stored in files with the **.js extension.**

```
<html>
<head>
<script language="javascript" src="sample.js"> </script>
</head>
<body>
……
</body>
</html>
```

The script tag:

The <SCRIPT> tag is an extension to HTML that can enclose any number of JavaScript statements as shown here:

```
<SCRIPT>
    JavaScript statements...
</SCRIPT>
```

A document can have multiple SCRIPT tags, and each can enclose any number of JavaScript statements.

FIRST JAVASCRIPT PROGRAM

```
<!DOCTYPE html>
<html>
    <body>
          <script>
     document.write(“Hello, World!!”);
     alert(“Hello, World!!”);
         </script>
    </body>
</html>
```

JAVASCRIPT CODE IN A FILE

- The SRC attribute of the <SCRIPT> tag lets you specify a file as the JavaScript source (rather than embedding the JavaScript in the HTML).

- This attribute is especially useful for sharing functions among many different pages.

Example:

```
<!DOCTYPE HTML>
<html>
<body>
 <p>Before the script...</p>
<script> alert( 'Hello, world!' );
</script>
 <p>...After the script.</p>
 </body>
 </html>
```

POPUP BOXES IN JAVASCRIPT

alert("string") opens box containing the message

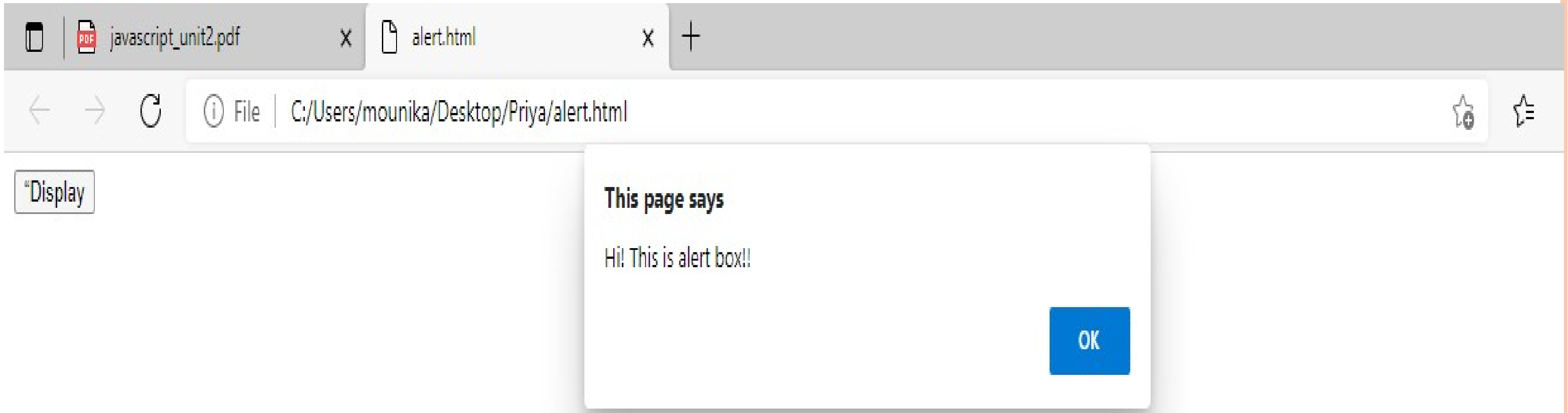
confirm("string") displays a message box with OK and CANCEL buttons

prompt("string") displays a prompt window with field for the user to enter a text string

Example:

```
<html>
<head>
<script language="javascript">
function show_alert()
{
alert("Hi! This is alert box!!");
}
</script>
</head>
<body>
<input type="button"onclick="show_alert()" value="Display alert box" >
</input>
</body>
</html>
```

OUTPUT

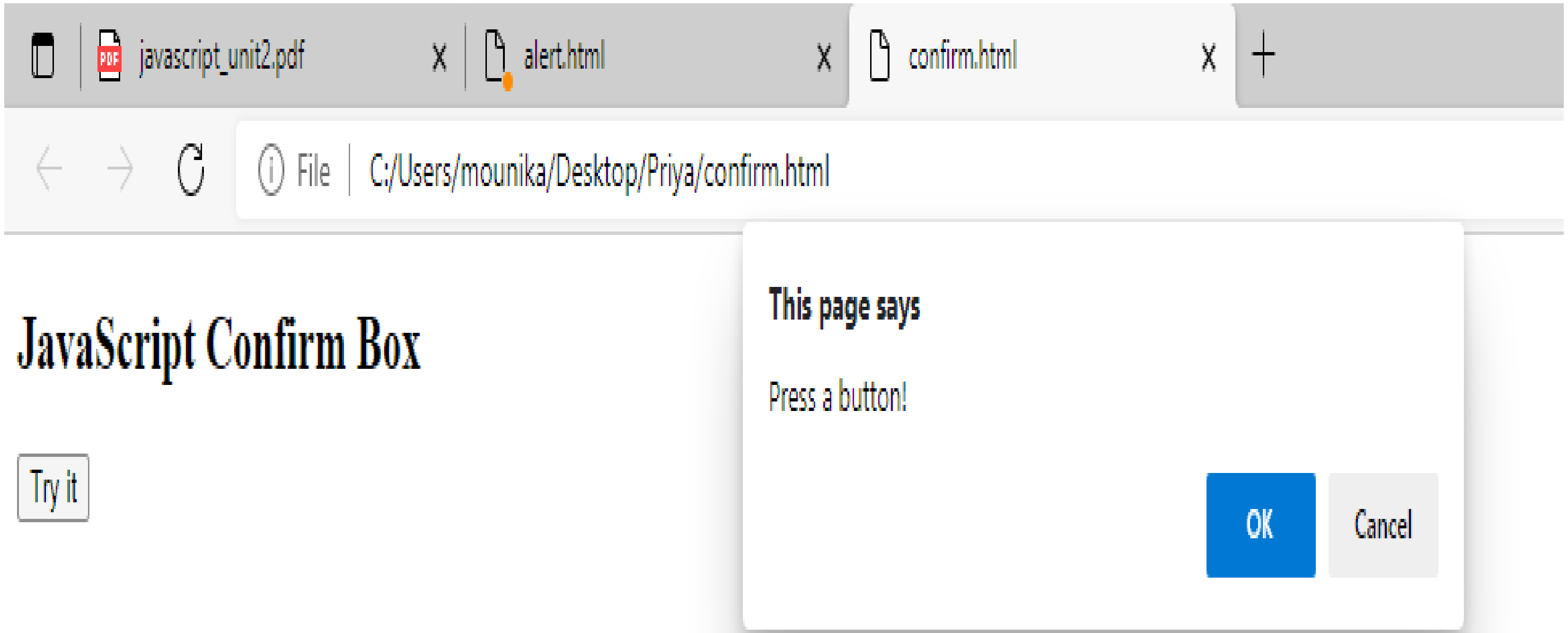


This page says

Hi! This is alert box!!

Confirm("string")

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Confirm Box</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
```


javascript_unit2.pdf

alert.html

confirm.html

File C:/Users/mounika/Desktop/Priya/confirm.html

JavaScript Confirm Box

Try it

This page says

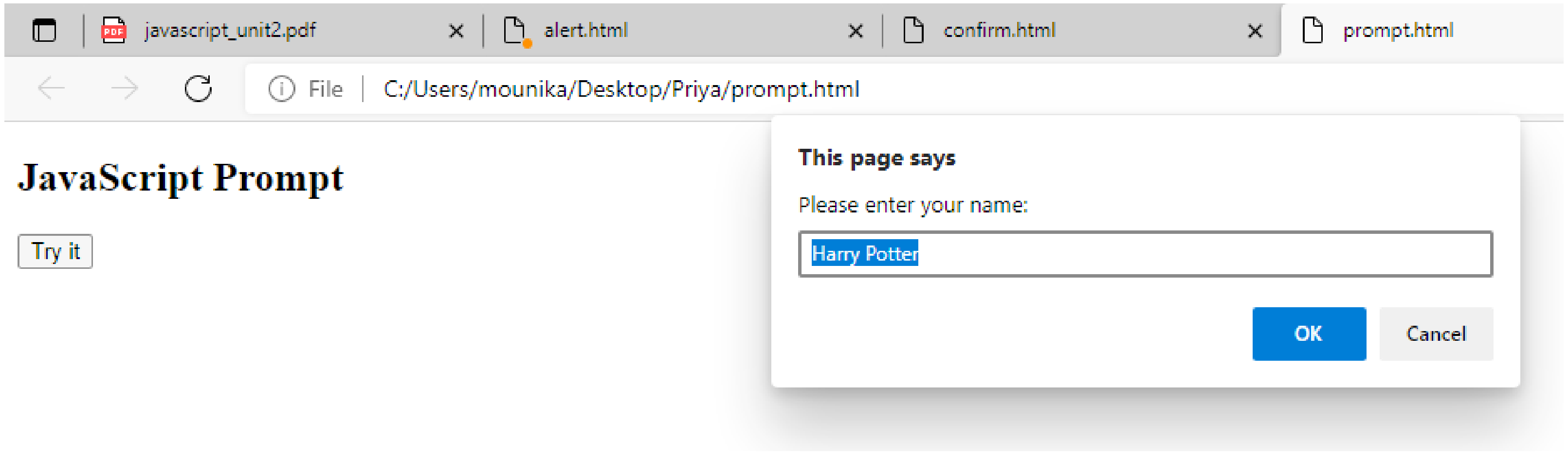
Press a button!

OK Cancel

PROMPT("STRING")

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Prompt</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let text;
let person = prompt("Please enter your name:", "Harry Potter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
```

```
text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```



javascript_unit2.pdf | alert.html | confirm.html | prompt.html

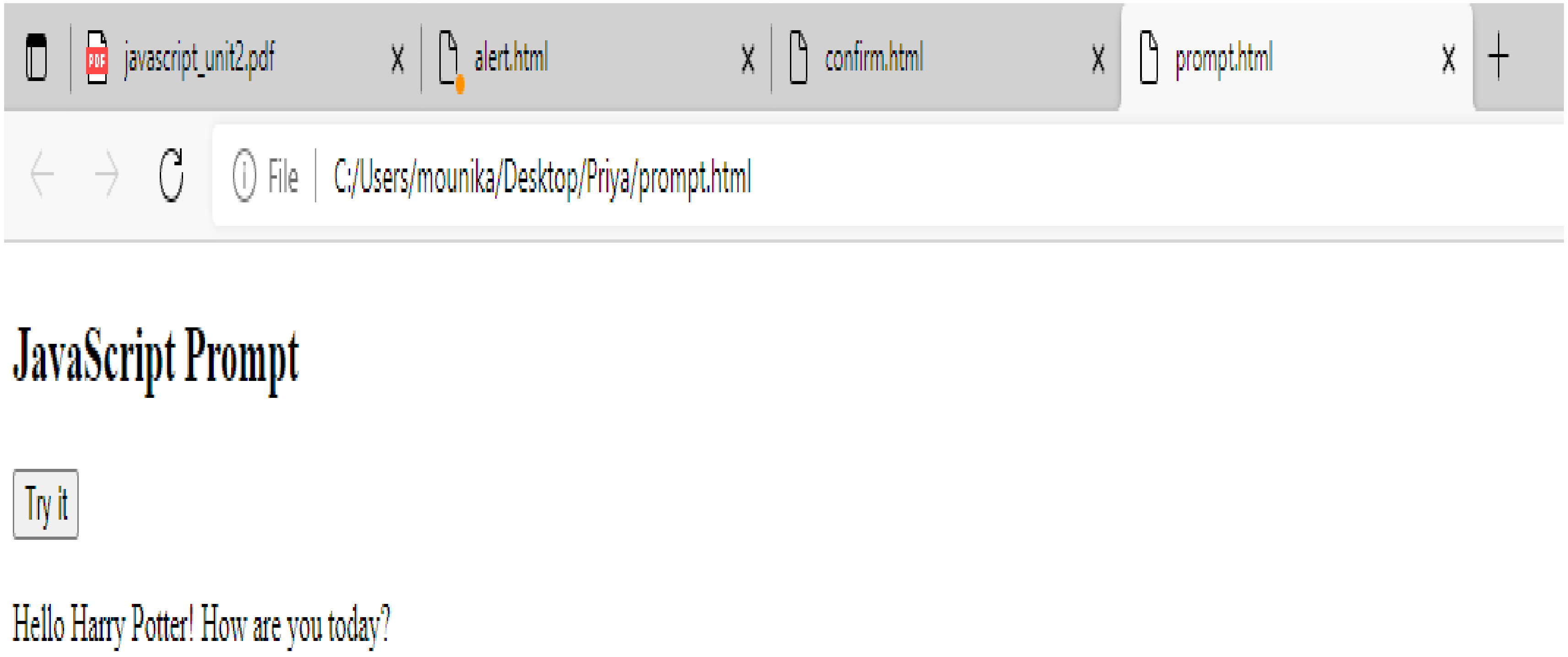
File | C:/Users/mounika/Desktop/Priya/prompt.html

JavaScript Prompt

This page says

Please enter your name:

Harry Potter


javascript_unit2.pdf
alert.html
confirm.html
prompt.html

File | C:/Users/mounika/Desktop/Priya/prompt.html

JavaScript Prompt

Try it

Hello Harry Potter! How are you today?

JavaScript Programming Elements

- Variables, datatypes , operators
- Statements
- Arrays
- Functions
- Objects in JavaScript
- Exception Handling
- Events
- Dynamic HTML with JavaScript

VARIABLES

- A variable is a name assigned to computer memory location to store data.
- As the name suggests, the value of the variable can vary, as the program runs.
- We can create a variable with the var statement

```
    var <variablename> = <some value>;
```

Example:

```
var sum = 0;
var str;
```

We can initialize a variable like this:

```
str = “hello”;
```

Rules for variable names:

# They must begin with a letter, digit or underscore character

# We can"t use spaces in names

# Variable names are case sensitive

# We can"t use reserved word as a variable name.

Weakly Typed Language:

Most high-level languages, including C and Java, are strongly typed. That is, a variable must be declared before it is used, and its type must be included in its declaration. Once a variable is declared, its type cannot be changed.

As the JavaScript is weakly typed language, data types are not explicitly declared.

Example:

```
var num;
 num = 3;
 num = "San Diego";
```

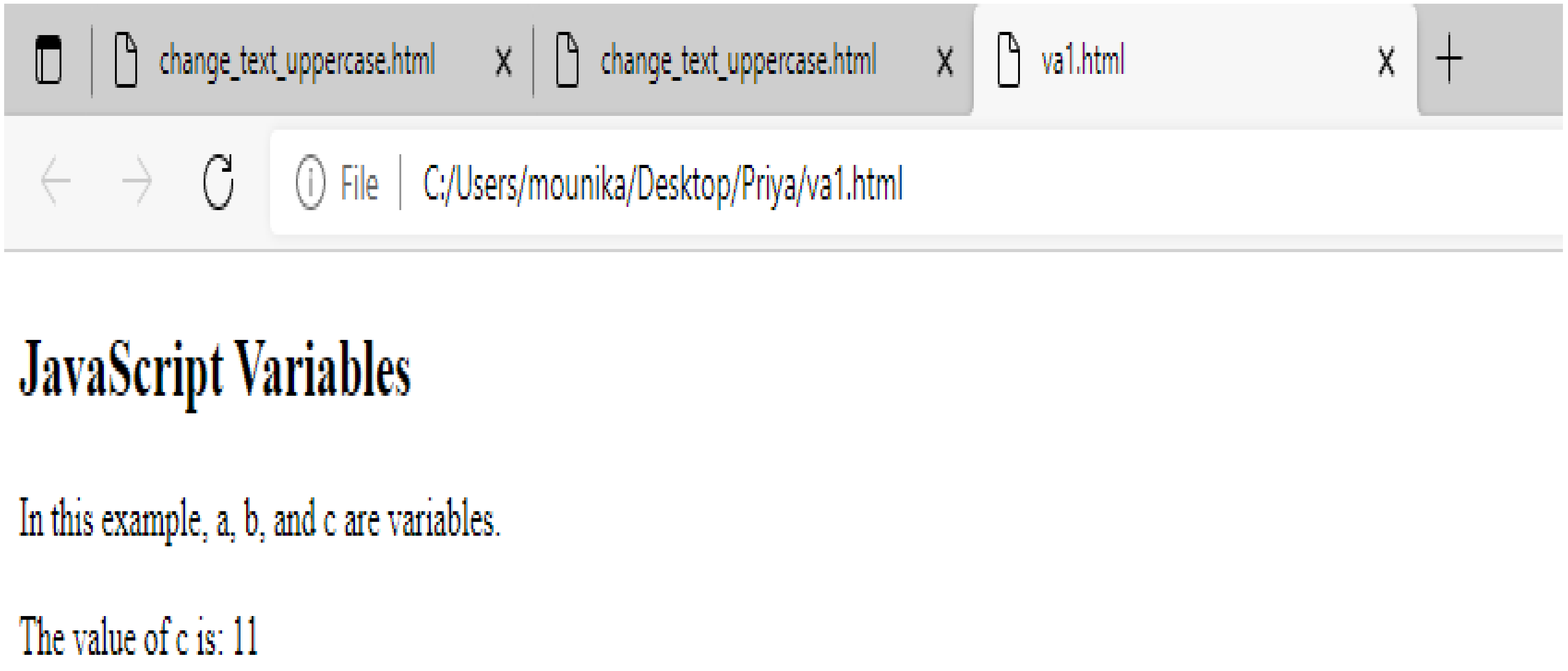
First, when the variable num is declared, it is empty. Its data type is actually the type undefined.

Then we assign it to the number 3, so its data type is numeric.

Next we reassign it to the string "San Diego", so the variable‟s type is now string.

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, a, b, and c are variables.</p>

<p id="demo"></p>

<script>
var a = 5;
var b = 6;
var c = a + b;
document.getElementById("demo").innerHTML =
"The value of c is: " + c;
</script>

</body>
</html>
```


change_text_uppercase.html
change_text_uppercase.html
va1.html
File
C:/Users/mounika/Desktop/Priya/va1.html

JavaScript Variables

In this example, a, b, and c are variables.

The value of c is: 11

DATATYPES

JavaScript supports five primitive data types:

- Number
- String
- Boolean
- Undefined
- Null

These types are referred to as ***primitive types*** because they are the basic building blocks from which more complex types can be built.

Of the five, only **number, string**, and **boolean** are real data types in the sense of actually storing data.

Undefined and null are types that arise under special circumstances.

- Primitive Data Types
  - Numbers
  - Strings
  - Boolean (True, False)
- Composite Data Types
  - Arrays
  - Objects
- **Numbers** - A number can be either an integer or a decimal
- **Strings -** A string is a sequence of letters or numbers enclosed in single or double quotes
- **Boolean** - True or False

- Although JavaScript does not have explicit data types, it does have implicit data types

- If you have an expression which combines two numbers, it will evaluate to a number

- If you have an expression which combines a string and a number, it will evaluate to a string

Example: Variables

```
var x = 1;

var y = 2;

var z = “abc”;

var q = “10”;
```

```
Ans = x + y;
 Ans => 03

Ans = z + x;
 Ans => abc1

Ans = x + q;
 Ans => 110
```

var x = 4;

var y = 5;

var z = “xyz”;

var q = “17”;

Ans = x + y + z;

Ans => 9xyz

Ans = q + x + y;

Ans => 179

- **Decimal:** The usual numbers which are having the base 10 are the decimal numbers

- **Octal:** Octal literals begin with a leading zero, and they consist of digits from zero through seven. The following are all valid octal literals:

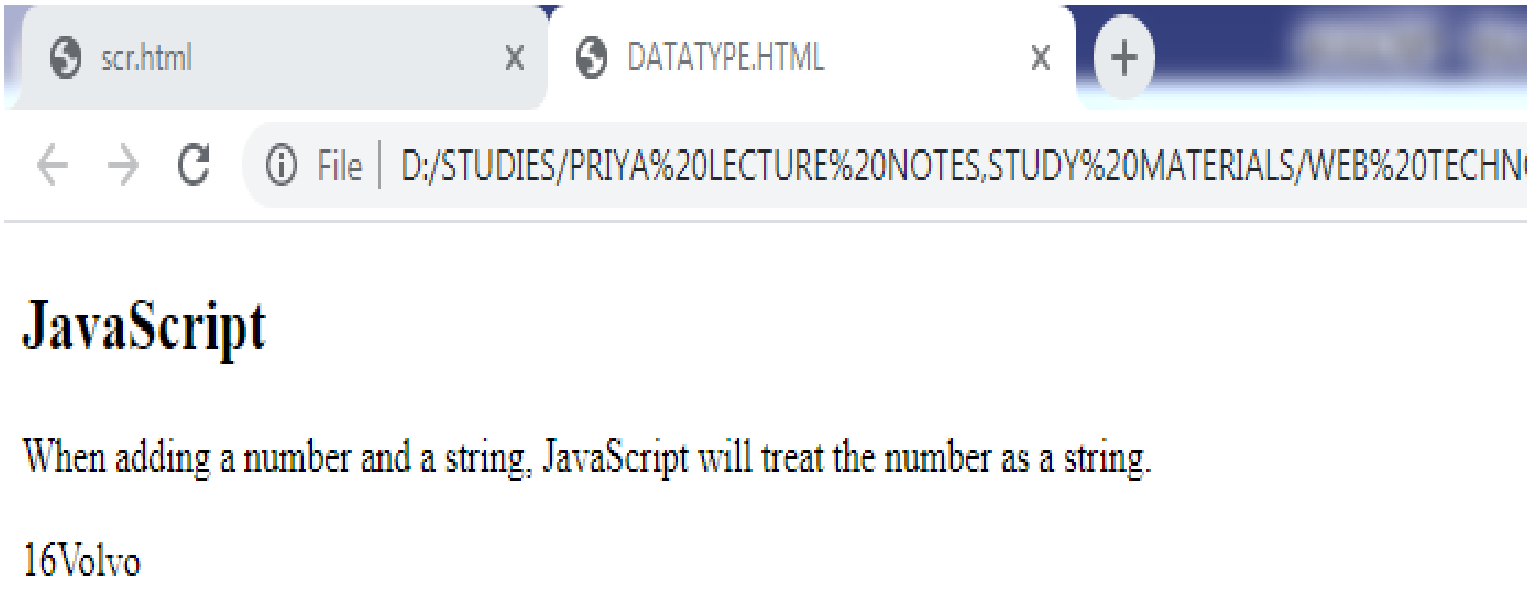
  00 0777 024513600

- **HexaDecimal:** Hexadecimal literals begin with a leading 0x, and they consist of digits from 0 through 9 and letters A through F. The following are all valid hexadecimal literals:

  0x0 0XF8f00 0x1a3C5e7

The Concept of Data Types

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript will treat the
   number as a string.</p>

<p id="demo"></p>

<script>
var x = 16 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

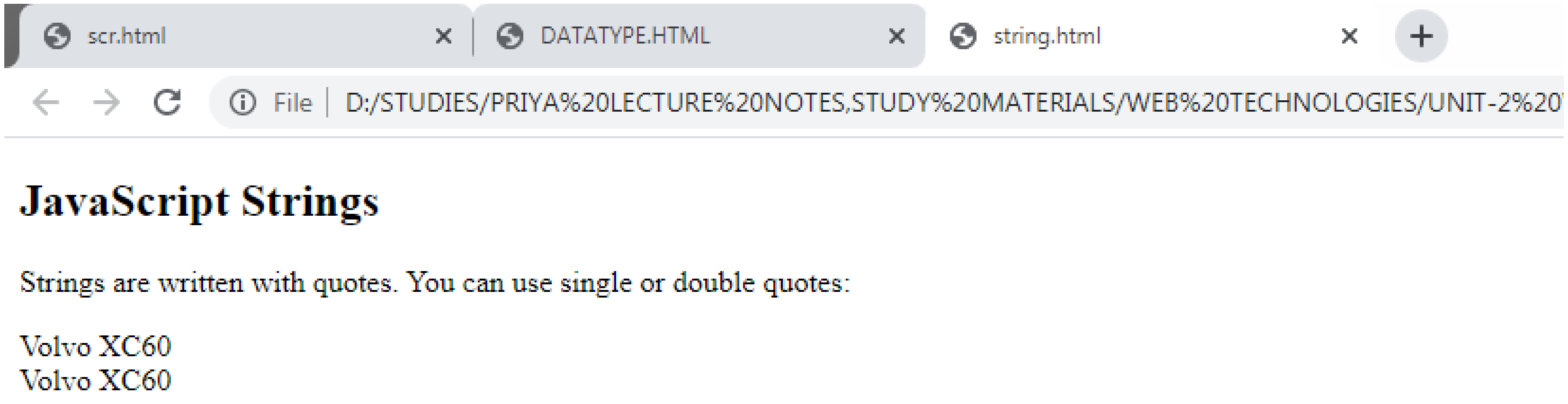

scr.html
DATATYPE.HTML
File
D:/STUDIES/PRIYA%20LECTURE%20NOTES,STUDY%20MATERIALS/WEB%20TECHN

```
JavaScript

When adding a number and a string, JavaScript will treat the number as a string.

16Volvo
```

JavaScript Strings

- A string (or a text string) is a series of characters like "John Doe".
- Strings are written with quotes. You can use single or double quotes:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>
<p>Strings are written with quotes. You can use single or double quotes:</p>
<p id="demo"></p>
<script>
var carName1 = "Volvo XC60";
var carName2 = 'Volvo XC60';
document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2;
</script>
</body>
</html>
```


scr.html
DATATYPE.HTML
string.html
File
D:/STUDIES/PRIYA%20LECTURE%20NOTES,STUDY%20MATERIALS/WEB%20TECHNOLOGIES/UNIT-2%20

JavaScript Strings

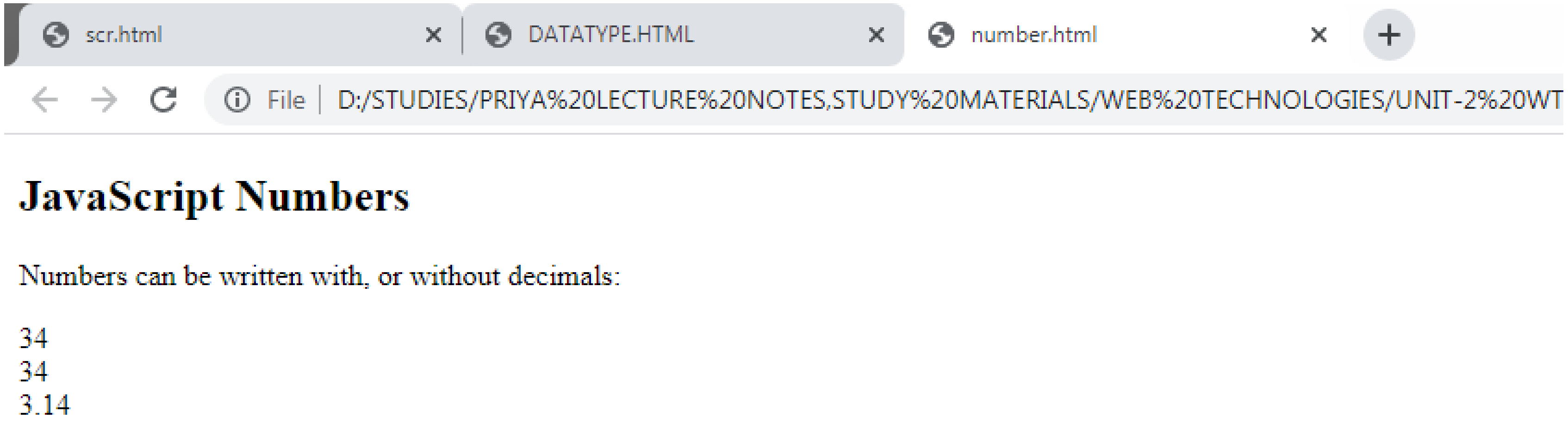
Strings are written with quotes. You can use single or double quotes:

```
Volvo XC60
Volvo XC60
```

JavaScript Numbers

- JavaScript has only one type of numbers.
- Numbers can be written with, or without decimals:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>

<p id="demo"></p>

<script>
var x1 = 34.00;
var x2 = 34;
var x3 = 3.14;

document.getElementById("demo").innerHTML =
x1 + "<br>" + x2 + "<br>" + x3;
</script>

</body>
</html>
```


scr.html
DATATYPE.HTML
number.html
File
D:/STUDIES/PRIYA%20LECTURE%20NOTES,STUDY%20MATERIALS/WEB%20TECHNOLOGIES/UNIT-2%20WT

JavaScript Numbers

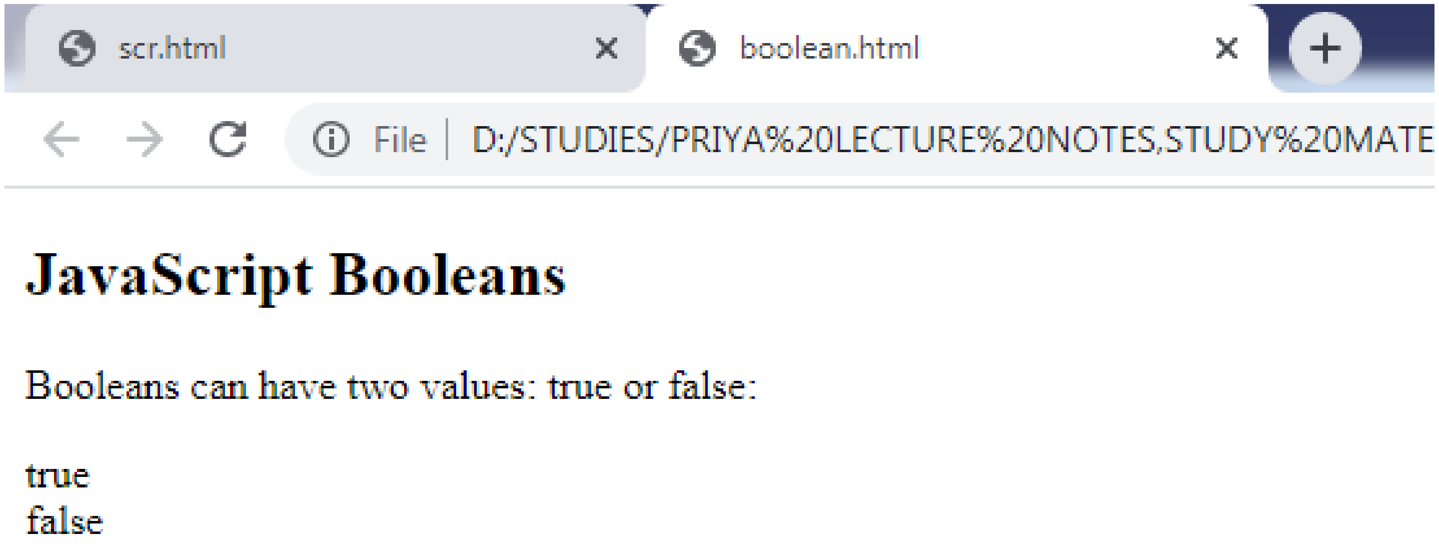
Numbers can be written with, or without decimals:

```
34
34
3.14
```

JavaScript Booleans

Booleans can only have two values: true or false.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Booleans</h2>

<p>Booleans can have two values: true or false:</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 5;
var z = 6;
document.getElementById("demo").innerHTML =
(x == y) + "<br>" + (x == z);
</script>

</body>
</html>
```



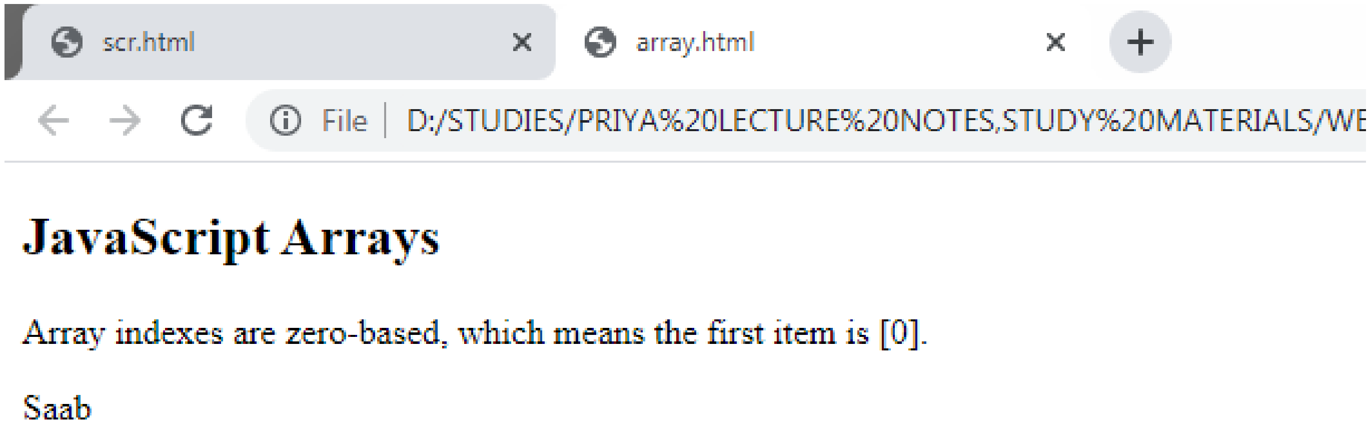
JavaScript Booleans

Booleans can have two values: true or false:

```
true
false
```

JavaScript Arrays

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array indexes are zero-based, which means the first item is [0].</p>

<p id="demo"></p>

<script>
var cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```



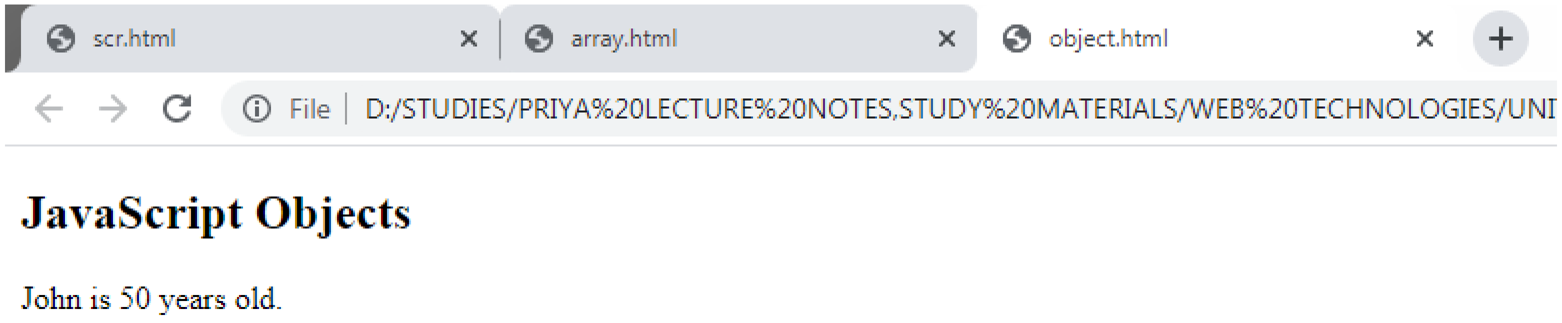
JavaScript Arrays

Array indexes are zero-based, which means the first item is [0].

Saab

JavaScript Objects

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
var person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

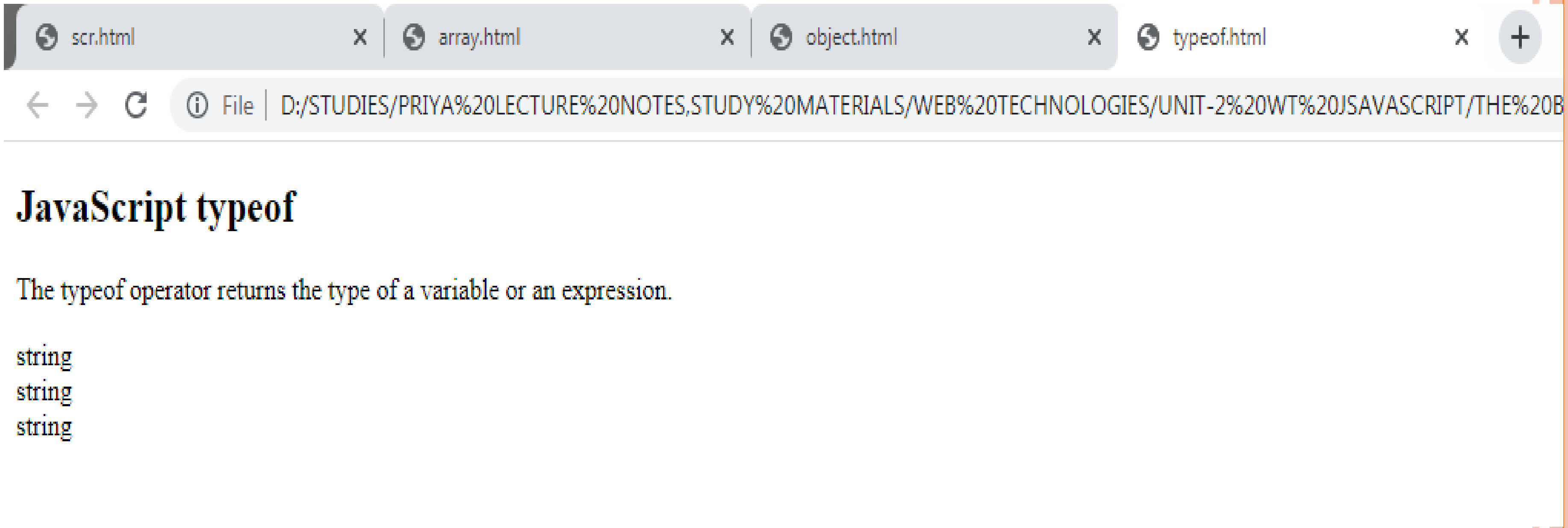

scr.html
array.html
object.html
File
D:/STUDIES/PRIYA%20LECTURE%20NOTES,STUDY%20MATERIALS/WEB%20TECHNOLOGIES/UNI

JavaScript Objects

John is 50 years old.

The typeof Operator

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof "" + "<br>" +
typeof "John" + "<br>" +
typeof "John Doe";
</script>
</body>
</html>
```



scr.html
array.html
object.html
typeof.html
File
D:/STUDIES/PRIYA%20LECTURE%20NOTES,STUDY%20MATERIALS/WEB%20TECHNOLOGIES/UNIT-2%20WT%20JSAVASCRIPT/THE%20B

JavaScript typeof

The typeof operator returns the type of a variable or an expression.

```
string
string
string
```

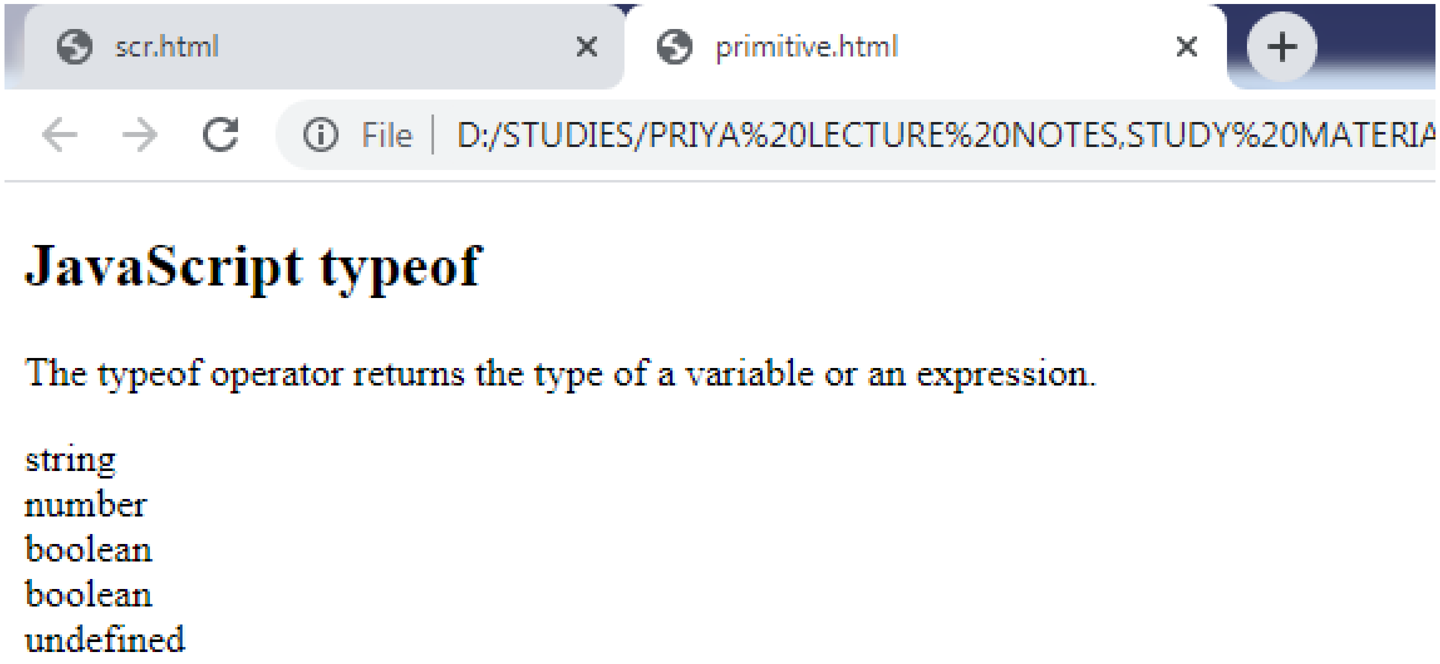
Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The typeof operator can return one of these primitive types:

- string
- number
- Boolean
- undefined

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript typeof</h2>
<p>The typeof operator returns the type of a variable or an expression.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
typeof "john" + "<br>" +
typeof 3.14 + "<br>" +
typeof true + "<br>" +
typeof false + "<br>" +
typeof x;
</script>
</body>
</html>
```



JavaScript typeof

The typeof operator returns the type of a variable or an expression.

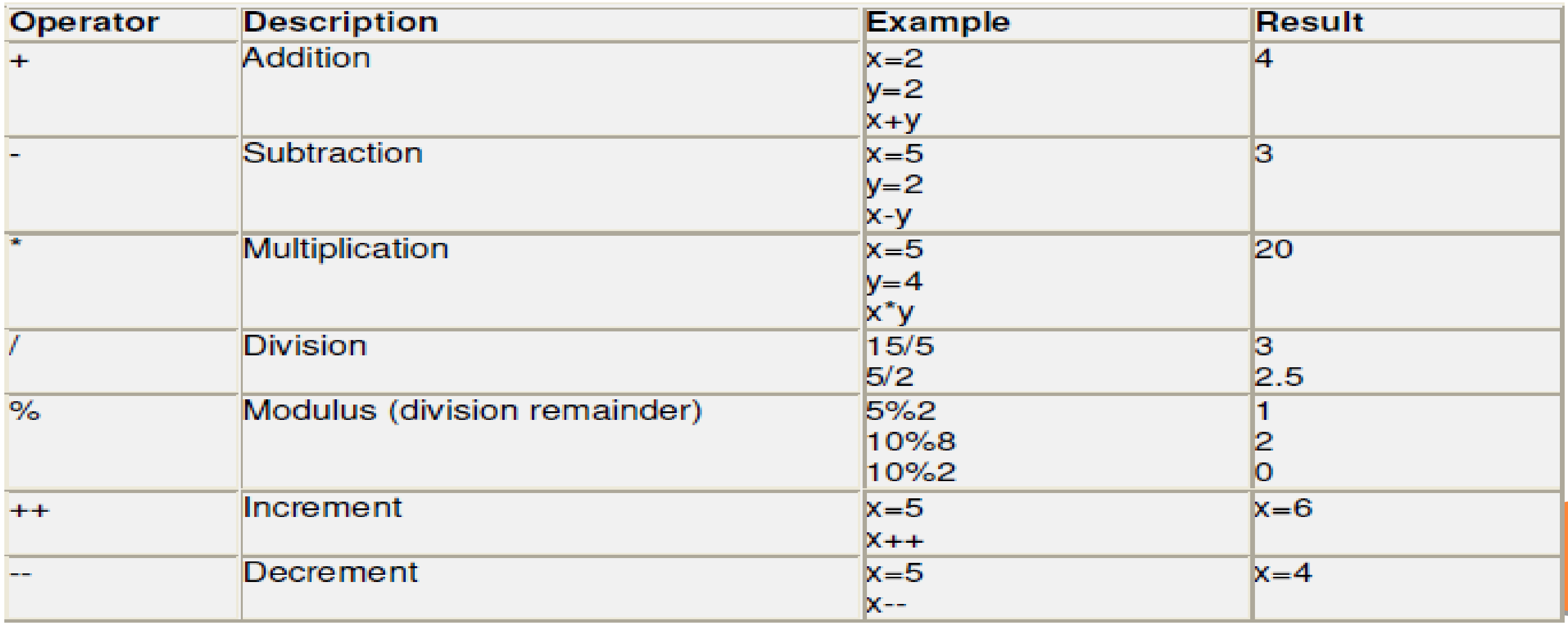
```
string
number
boolean
boolean
undefined
```

Operators in JavaScript

The operators in JavaScript can be classified as follows:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators

1. Arithmetic operators



Operator	Description	Example	Result

+	Addition	x=2
		y=2
		x+y
			4

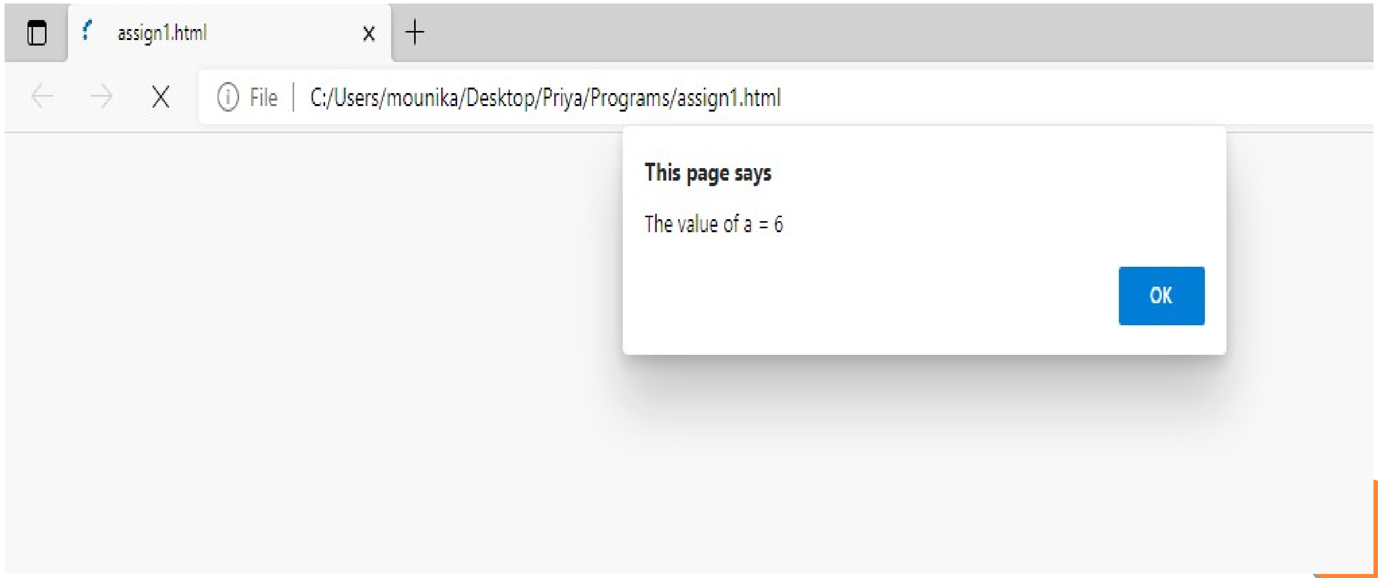
-	Subtraction	x=5
		y=2
		x-y
			3

*	Multiplication	x=5
		y=4
		x*y
			20

/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

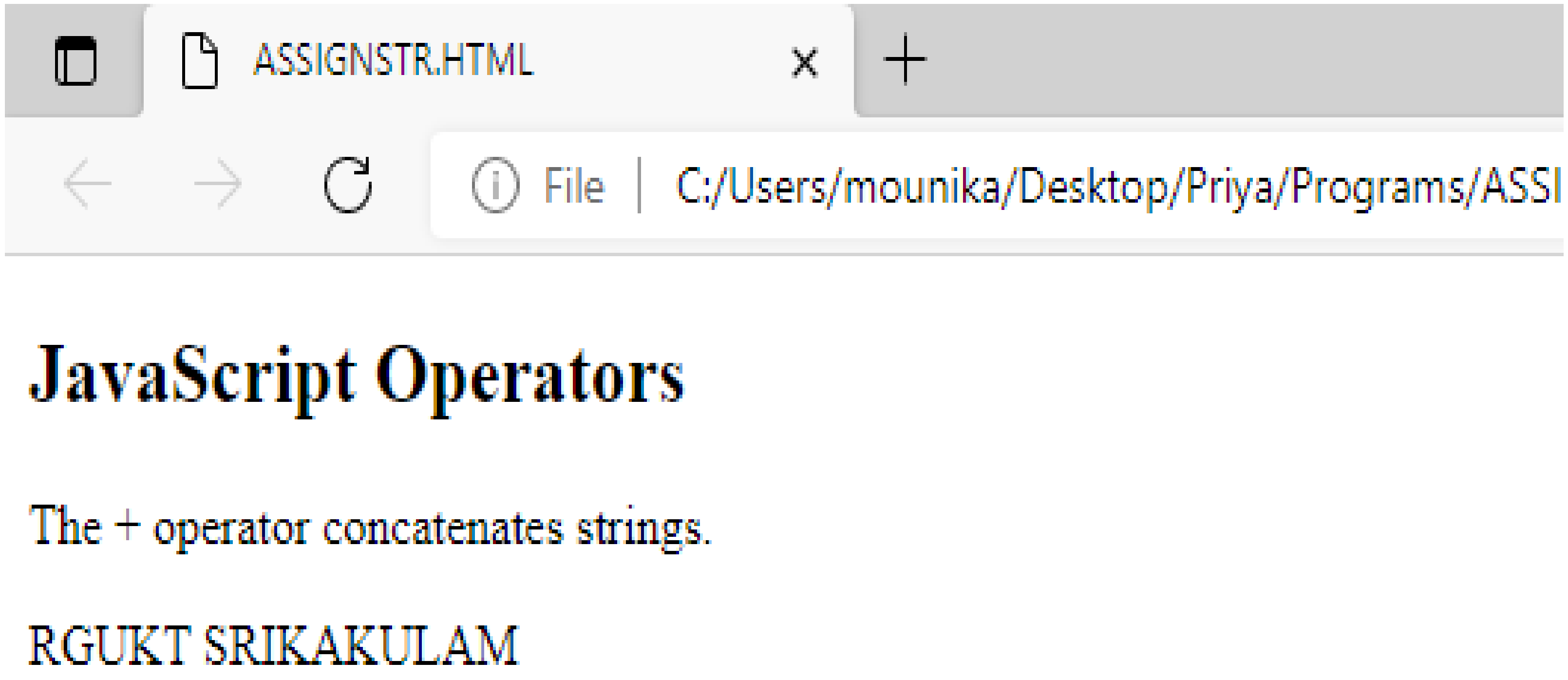
Example:

```
<html>
<body>
<script language="JavaScript">
<!--
var a = 5;
++a;
alert("The value of a = " + a );
-->
</script>
</body>
</html>
```



```
This page says
The value of a = 6
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Operators</h2>

<p>The + operator concatenates strings.</p>

<p id="demo"></p>

<script>
let text1 = "RGUKT";
let text2 = "SRIKAKULAM";
document.getElementById("demo").innerHTML = text1 + " " + text2;
</script>

</body>
</html>
```

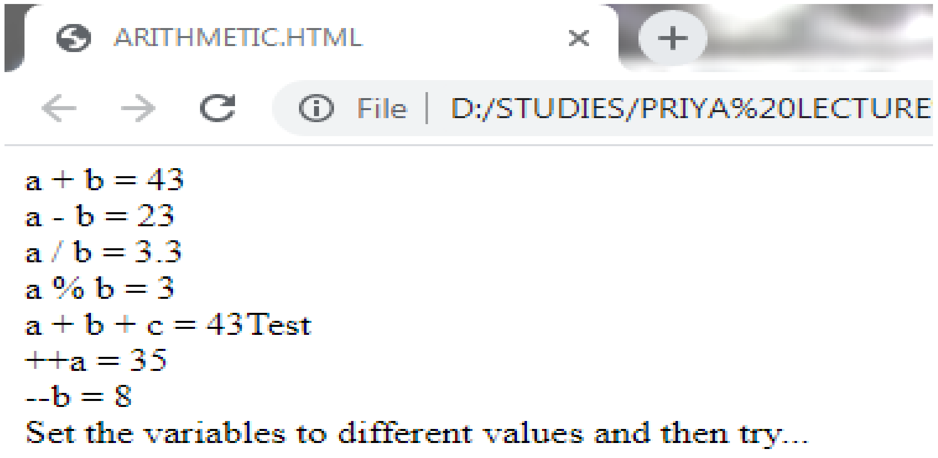


```
JavaScript Operators

The + operator concatenates strings.

RGUKT SRIKAKULAM
```

EXAMPLE:

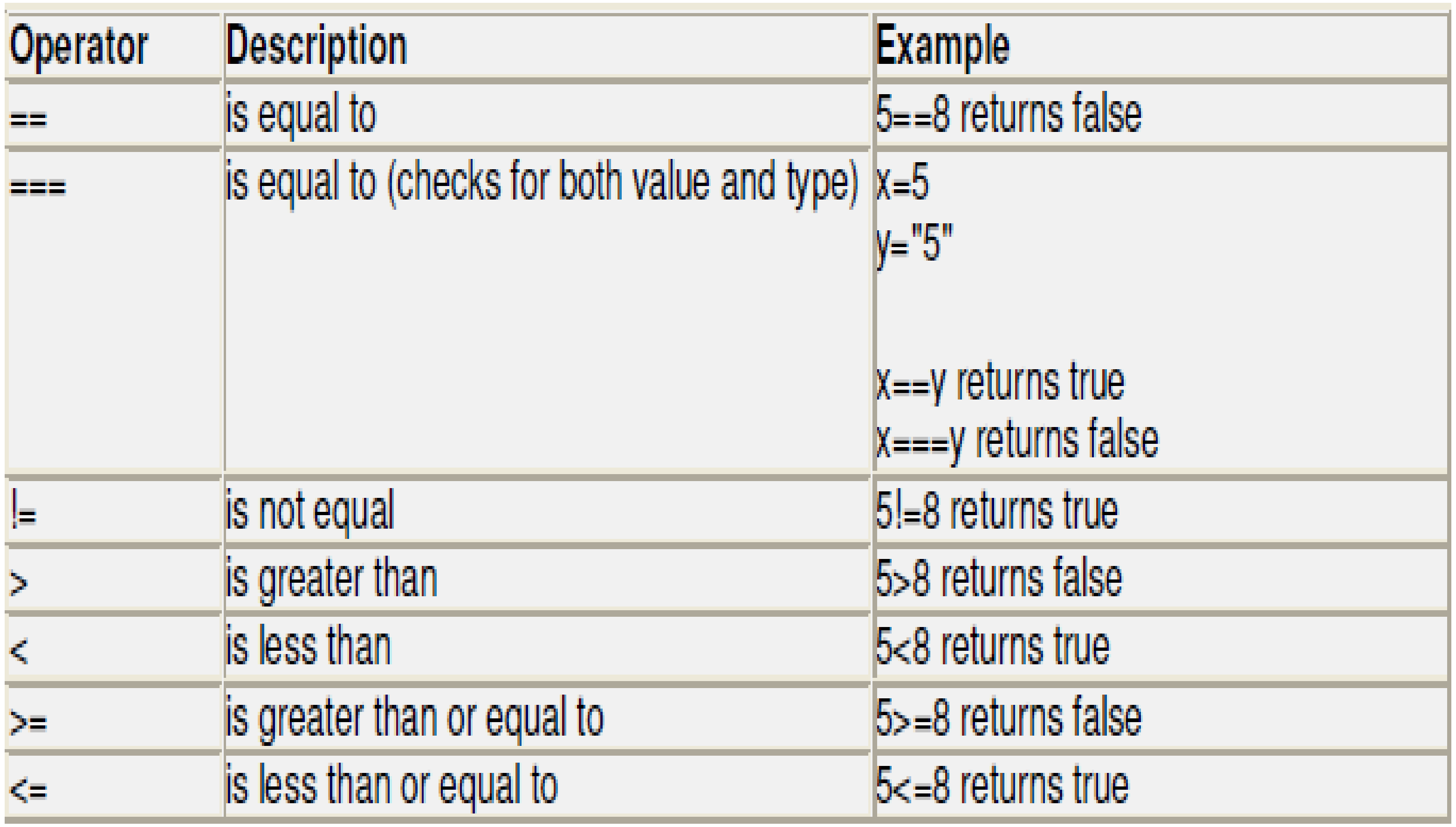
```
<html>
  <body>

    <script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);
document.write("a / b = ");
 result = a / b;
 document.write(result);
 document.write(linebreak);

 document.write("a % b = ");
 result = a % b;
 document.write(result);
 document.write(linebreak);

 document.write("a + b + c = ");
 result = a + b + c;
 document.write(result);
 document.write(linebreak);

 a = ++a;
 document.write("++a = ");
 result = ++a;
 document.write(result);
 document.write(linebreak);
```

```
          b = --b;
        document.write("--b = ");
        result = --b;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    Set the variables to different values and then try...
  </body>
</html>
```



```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Relational operators/Comparison operators:
Relational operators are used to compare quantities.



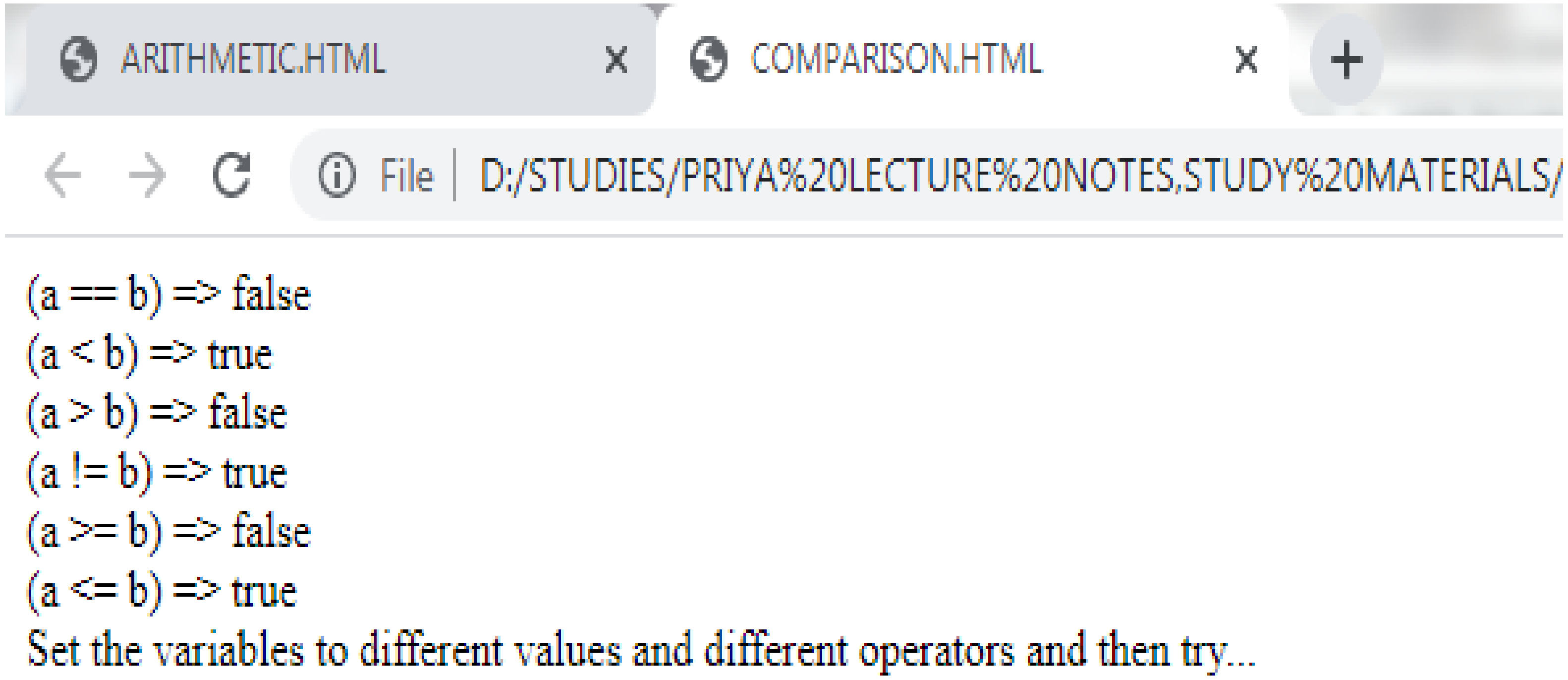
Operator	Description	Example
==	is equal to	5==8 returns false

===	is equal to (checks for both value and type)	x=5
		y="5"
		
		x==y returns true
		x===y returns false

!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

EXAMPLE:

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write("(a == b) => ");
        result = (a == b);
        document.write(result);
        document.write(linebreak);

        document.write("(a < b) => ");
        result = (a < b);
        document.write(result);
        document.write(linebreak);

        document.write("(a > b) => ");
        result = (a > b);
        document.write(result);
        document.write(linebreak);
        document.write("(a != b) => ");
       result = (a != b);
       document.write(result);
       document.write(linebreak);

       document.write("(a >= b) => ");
       result = (a >= b);
       document.write(result);
       document.write(linebreak);

       document.write("(a <= b) => ");
       result = (a <= b);
       document.write(result);
       document.write(linebreak);
     //-->
   </script>
   Set the variables to different values and different operators and then try...
 </body>
</html>
```

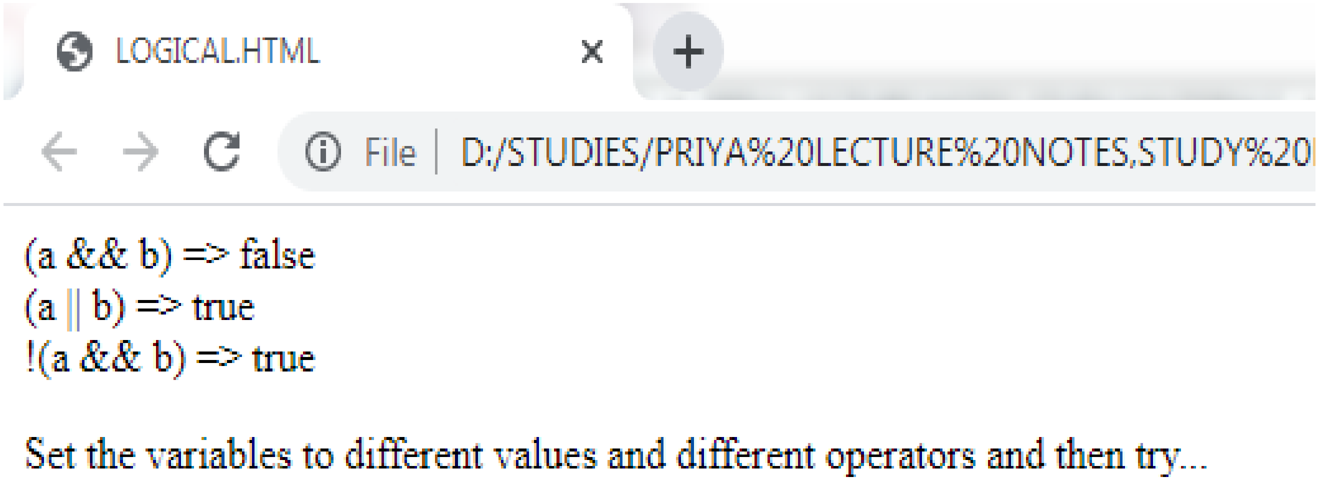


```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
Set the variables to different values and different operators and then try...
```

Logical operators:

Logical operator are used to combine two or more conditions.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = true;
            var b = false;
            var linebreak = "<br />";

            document.write("(a && b) => ");
            result = (a && b);
            document.write(result);
            document.write(linebreak);

            document.write("(a || b) => ");
            result = (a || b);
            document.write(result);
            document.write(linebreak);
document.write("!(a && b) => ");
        result = (!(a && b));
        document.write(result);
        document.write(linebreak);
      //-->
    </script>
    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

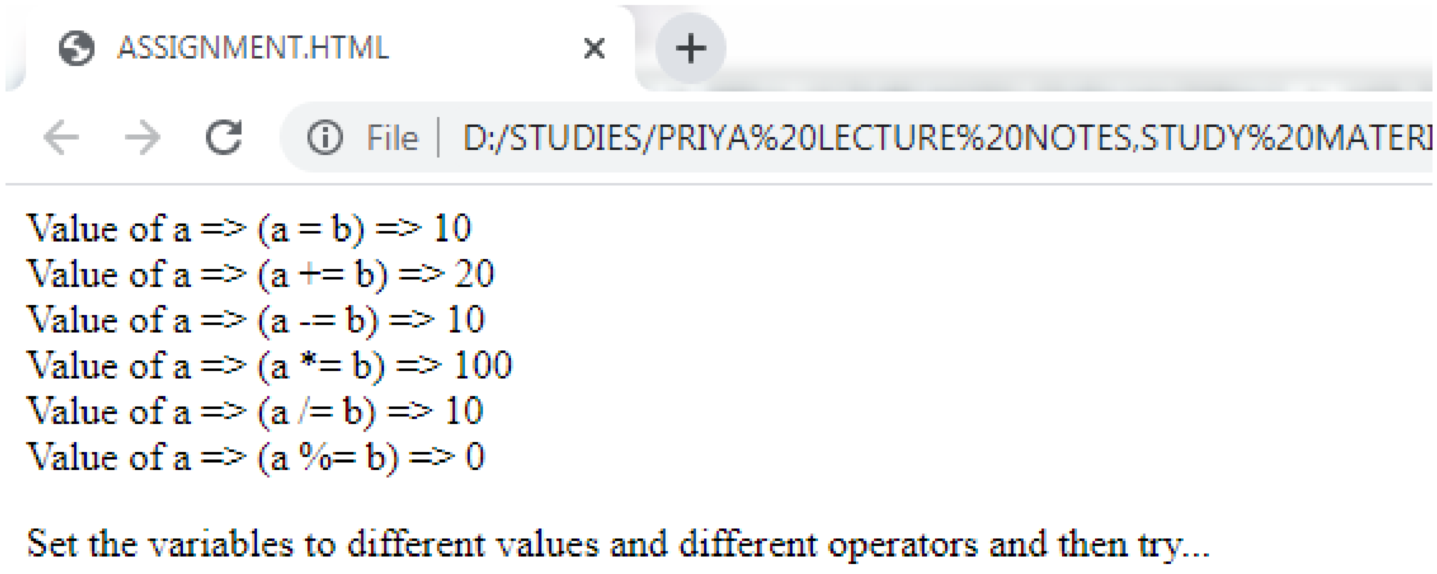


```
(a && b) => false
(a || b) => true
!(a && b) => true

Set the variables to different values and different operators and then try...
```

<u>Assignment Operators</u>: are used to assign the result of an expression to a variable

<u>Example:</u>

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";

            document.write("Value of a => (a = b) => ");
            result = (a = b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a += b) => ");
            result = (a += b);
            document.write(result);
            document.write(linebreak);
document.write("Value of a => (a -= b) => ");
          result = (a -= b);
          document.write(result);
          document.write(linebreak);

          document.write("Value of a => (a *= b) => ");
          result = (a *= b);
          document.write(result);
          document.write(linebreak);

          document.write("Value of a => (a /= b) => ");
          result = (a /= b);
          document.write(result);
          document.write(linebreak);

          document.write("Value of a => (a %= b) => ");
          result = (a %= b);
          document.write(result);
          document.write(linebreak);
       //-->
     </script>
     <p>Set the variables to different values and different operators and then try...</p>
   </body>
</html>
```



```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0

Set the variables to different values and different operators and then try...
```

ARRAYS

- An array is a compound data type that stores numbered pieces of data
- Each numbered datum is called an *element* of the array and the number assigned to it is called an *index*.
- The elements of an array may be of any type. A single array can even store elements of different type.

CREATING AN ARRAY:

- There are several different ways to create an array in JavaScript
- Using the `Array()` constructor:

  - var a = new Array(1, 2, 3, 4, 5);

  - var b = new Array(10);
- Using array literals:

  - var c = [1, 2, 3, 4, 5];

ACCESSING ARRAY ELEMENTS

- Array elements are accessed using the [ ] operator

Example:

- var colors = [“red”, “green”, “blue”];
- colors[0] => red
- colors[1] => green

Adding Elements

- To add a new element to an array, simply assign a value to it

Example:

```
var a = new Array(10);
a[7] = 17;
```

ARRAY LENGTH

- All arrays created in JavaScript have a special length property that specifies how many elements the array contains

Example:

- var colors = [“red”, “green”, “blue”];
- colors.length => 3

Changing an Array Element

This statement changes the value of the first element in campus:

campus[1] = "Ongole";

Example:

<!DOCTYPE html>

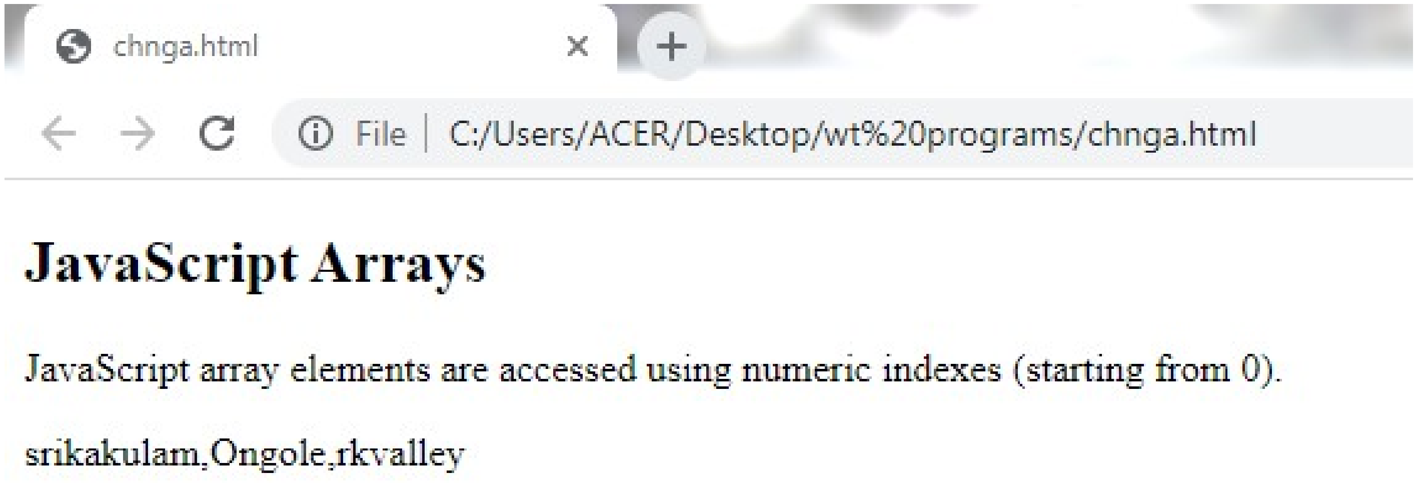
<html>

<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

```
<p id="demo"></p>
<script>
var campus = ["srikakulam", "nuzvid", "rkvalley"];
campus[1] = "Ongole";
document.getElementById("demo").innerHTML = campus;
</script>
</body>
</html>
```



JavaScript Arrays

JavaScript array elements are accessed using numeric indexes (starting from 0).

```
srikakulam,Ongole,rkvalley
```

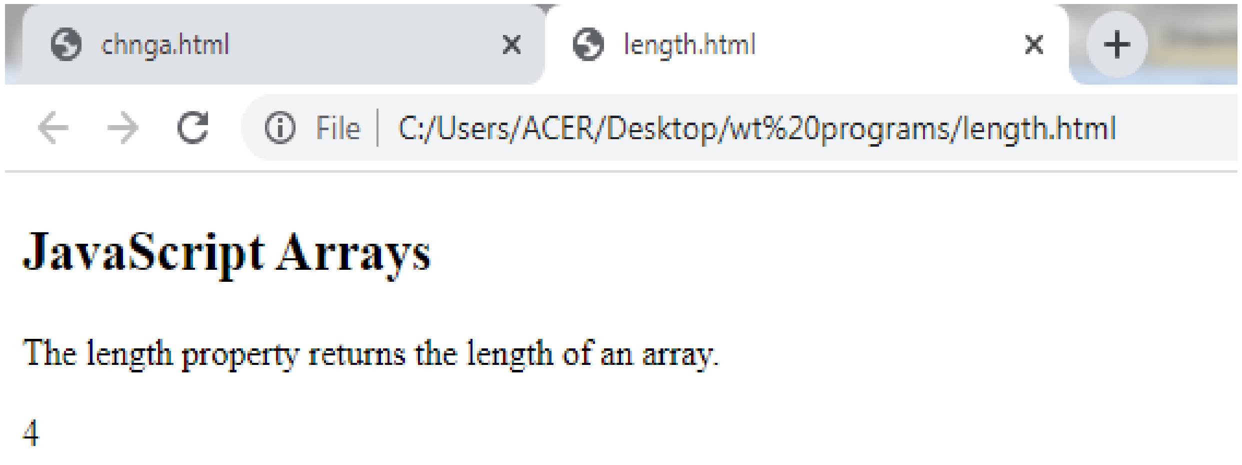
Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
var x = cars.length;   // The length property returns the number of elements
var y = cars.sort();   // The sort() method sorts arrays
```

<u>EXAMPLE:</u>

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The length property returns the length of an array.</p>
<p id="demo"></p>
<script>
var departments = ["cse", "ece", "mechanical", "eee"];
document.getElementById("demo").innerHTML = departments.length;
</script>
</body>
</html>
```

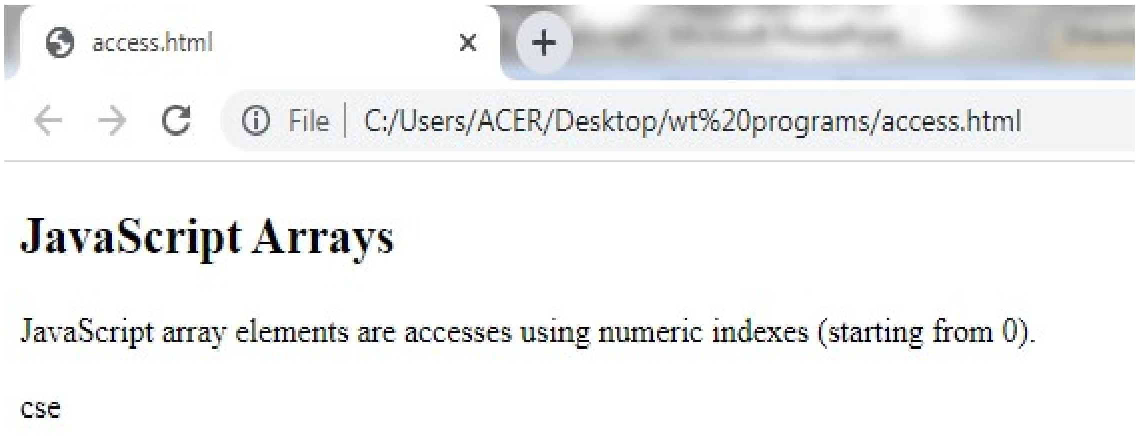

chnga.html
length.html
File | C:/Users/ACER/Desktop/wt%20programs/length.html

```
JavaScript Arrays

The length property returns the length of an array.

4
```

Accessing the Last Array Element

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>JavaScript array elements are accesses using numeric indexes (starting
   from 0).</p>
<p id="demo"></p>
```

```
<script>
var departments = ["cse", "eee", "ece", "Mechanical"];
var last = departments[departments.length-1];
document.getElementById("demo").innerHTML = last;
</script>
</body>
</html>
```



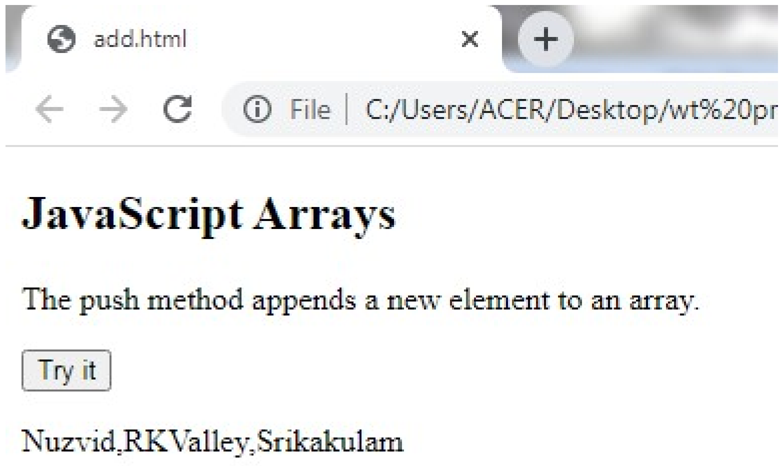
```
JavaScript Arrays

JavaScript array elements are accesses using numeric indexes (starting from 0).

cse
```

Adding Array Elements

The easiest way to add a new element to an array is using the push() method:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>

<p>The push method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var campus = ["Nuzvid", "RKValley", "Srikakulam"];
document.getElementById("demo").innerHTML = campus;
```

```
function myFunction() {
  campus.push("ongole");
  document.getElementById("demo").innerHTML = campus;
}
</script>
</body>
</html>
```



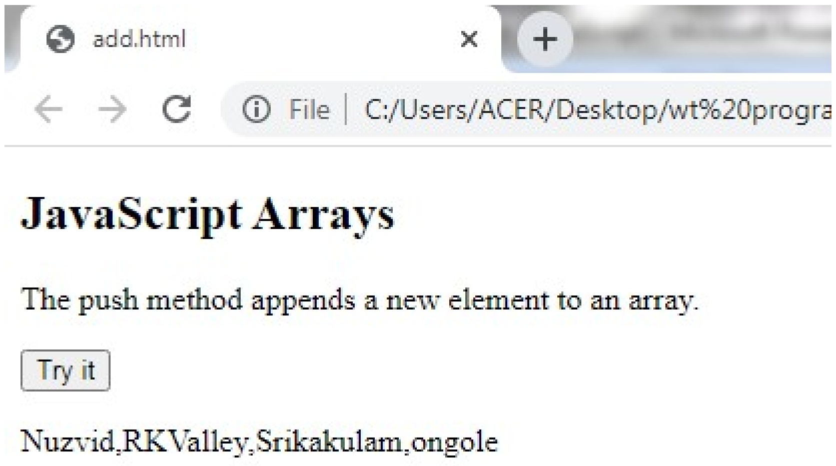
```
add.html
File | C:/Users/ACER/Desktop/wt%20pr
```

JavaScript Arrays

The push method appends a new element to an array.

Try it

Nuzvid,RKValley,Srikakulam



```
add.html
File | C:/Users/ACER/Desktop/wt%20progra
```

JavaScript Arrays

The push method appends a new element to an array.

Try it

Nuzvid,RKValley,Srikakulam,ongole

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

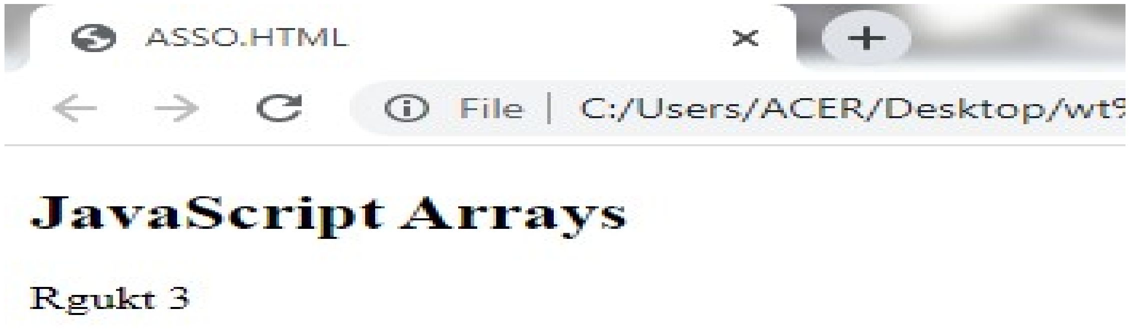
JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

EXAMPLE:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
```

```
var campus = [];
campus[0] = "Rgukt";
campus[1] = "srikakulam";
campus[2] = 1;
document.getElementById("demo").innerHTML =
campus[0] + " " + campus.length;
</script>
</body>
</html>
```



JavaScript Arrays

```
Rgukt 3
```

Functions

- Functions are a collection of JavaScript statements that performs a specified task
- Functions are used whenever it is necessary to repeat an operation
- Functions have inputs and outputs
- The inputs are passed into the function and are known as **arguments** or **parameters**
- Think of a function as a "black box" which performs an operation

A function is a JavaScript procedure--a set of statements that performs a specific task. A function definition has these basic parts:

- The **function** keyword.
- A function name.
- A comma-separated list of arguments to the function in parentheses.
- The statements in the function in curly braces.

- No type is specified for arguments!

Example: Function

```
function
   square(x)
{return x*x;}


z = 3;
sqr_z = square(z);
```

Name of Function:
square

Input/Argument: x

Output: x*x

JavaScript Function

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

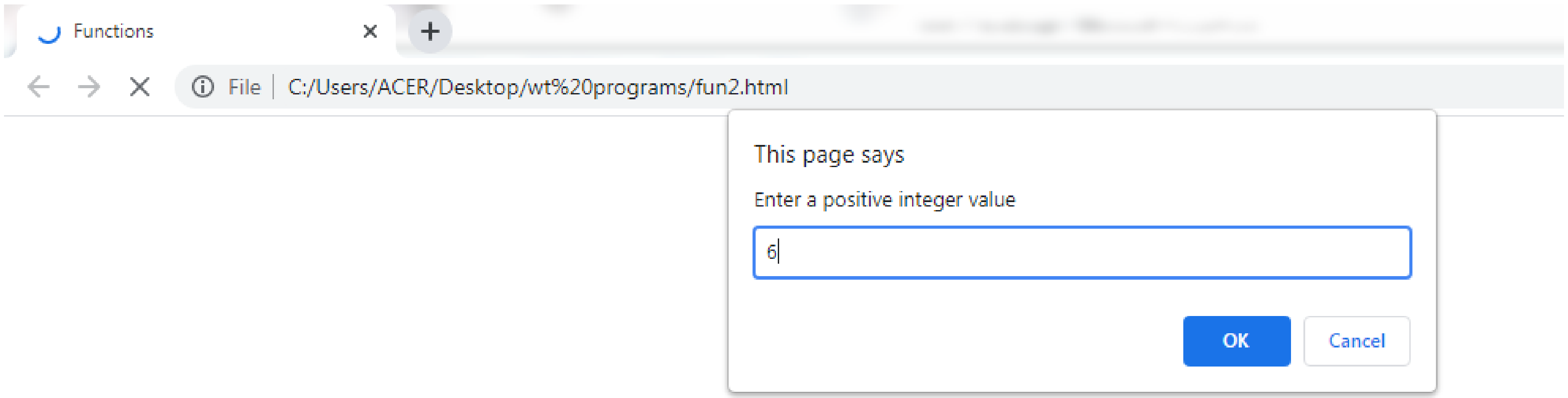
```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function Return

- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

EXAMPLE

```
<html>
    <head>
    <title>Functions</title>
    <script>
        function square(number)
        {
            return number * number;
        }
    </script>
    </head>
    <body>
        <script>
            var x = prompt("Enter a positive integer value",0);
            document.writeln("the value is ", square(x));
        </script>
    </body>
</html>
```

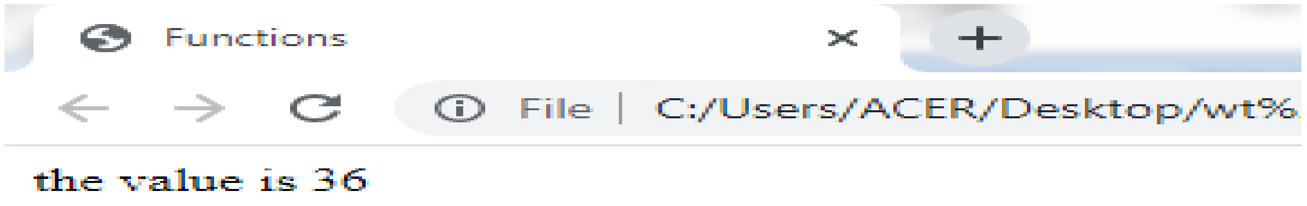


C:/Users/ACER/Desktop/wt%20programs/fun2.html

This page says

Enter a positive integer value

6



```
the value is 36
```

FUNCTION SCRIPT IN A FILE

function_square.html:

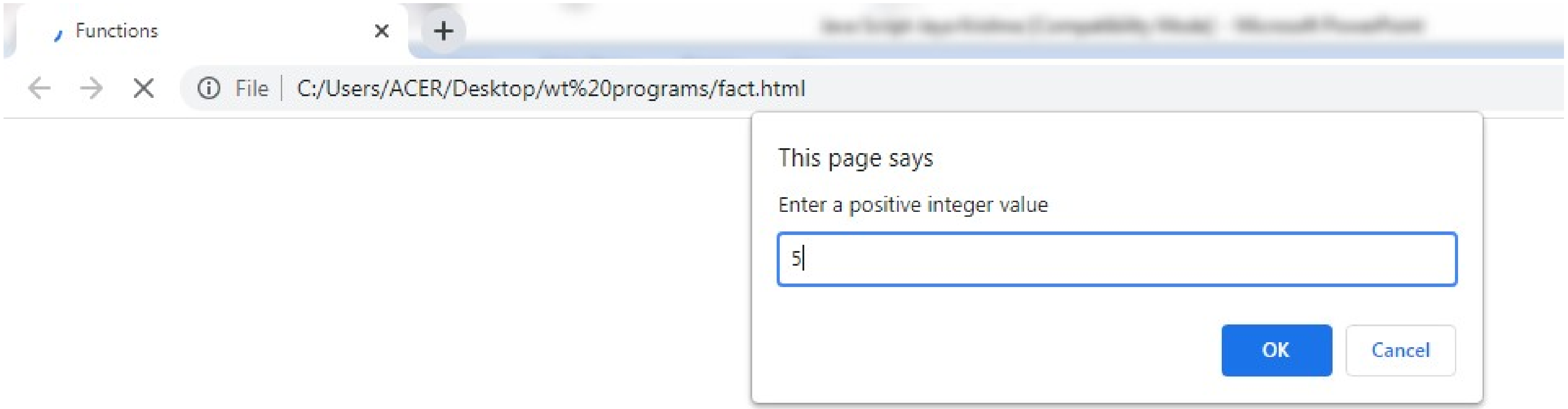
```
<html>
    <head>
    <title>Functions</title>
          <script src="square.js">
          </script>
    </head>
    <body>
          <script>
                var x = prompt("Enter a positive integer value",0);
                document.writeln("the value is ", square(x));
          </script>
    </body>
</html>
```

square.js:

```
function square(number)
{
      return number * number;
}
```

EXAMPLE:FACTORIAL

```
<head>
   <title>Functions</title>
   </head>
   <body>
   <script>
   function factorial(x) {
   if (x <= 0)
   return(1);
   else
   return( x * factorial(x-1));
   }
   var x = prompt("Enter a positive integer

value",0);
   document.write("The factorial of " + x + " is

" + factorial(x));

   </script>
   </body>
```

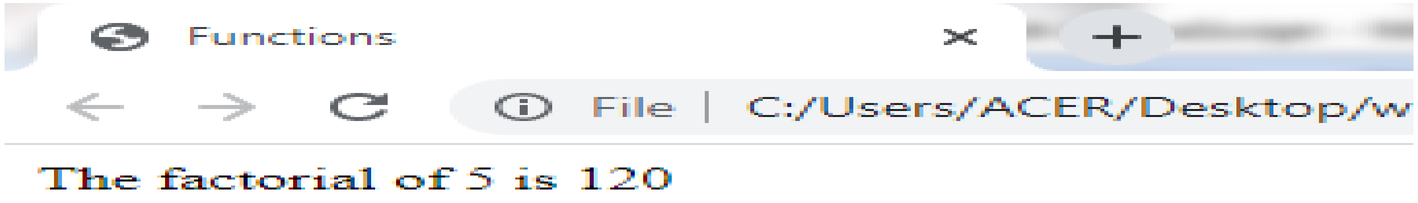

Functions

C:/Users/ACER/Desktop/wt%20programs/fact.html

This page says

Enter a positive integer value

5

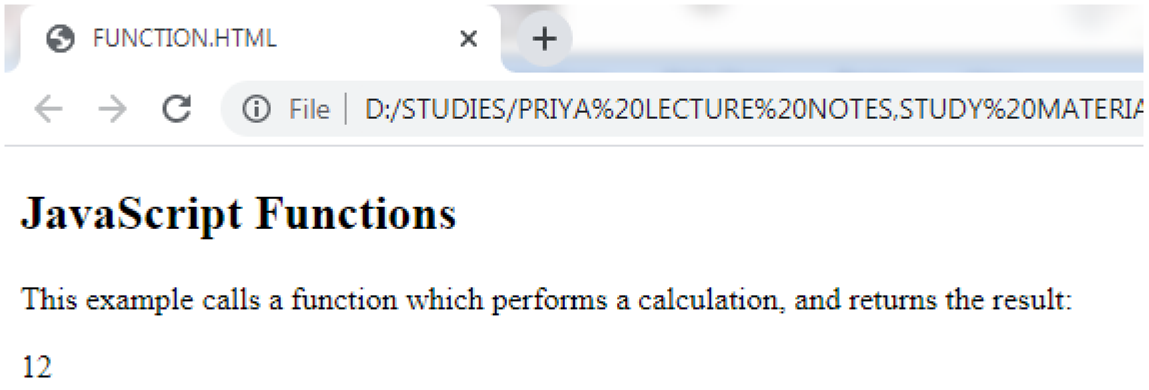

Functions

C:/Users/ACER/Desktop/w

The factorial of 5 is 120

EXAMPLE:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation and returns the
    result:</p>
<p id="demo"></p>
<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>
</body>
</html>
```



JavaScript Functions

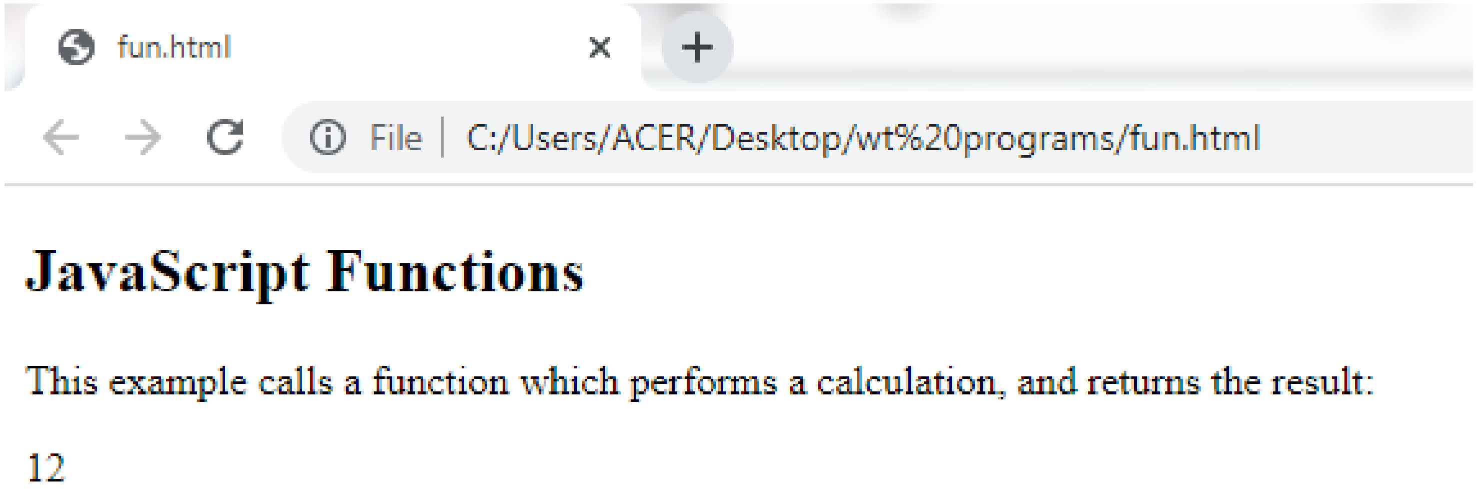
This example calls a function which performs a calculation, and returns the result:

12

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```



JavaScript Functions

This example calls a function which performs a calculation, and returns the result:

12

CONDITIONAL STATEMENTS

- There are three basic types of control structures in JavaScript: the **`if` statement, the `while` loop, and the `for` loop**
- Each control structure manipulates a block of JavaScript expressions beginning with { and ending with }
- In JavaScript we have the following conditional statements:
- **if** to specify a block of code to be executed, if the specified condition is true
- **else** to specify a block of code to be executed, if the same condition is false
- **else** if to specify a new condition to test, if the first condition is false
- **switch** to specify many alternative blocks of code to be executed

The If Statement

- The `if` statement allows JavaScript programmers to a make decision

```
If ( x  = =  10)
{  y  =  x*x;
}
else
{  x  =  0;
}
```

EXAMPLE

```
<html>
  <body>
    <script type = "text/javascript">

        var age = 20;

        if( age > 18 ) {
          document.write("<b>Eligible to Vote </b>");
        }

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

if...else statement

- The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.
- Syntax

```
if (expression){
Statement(s) to be executed if expression is true }
Else{
 Statement(s) to be executed if expression is false }
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

EXAMPLE

```
<html>
   <body>
      <script type = "text/javascript">

            var age = 25;

            if( age > 18 ) {
               document.write("<b>Eligible to Vote</b>");
            } else {
               document.write("<b>Not Eligible to Vote</b>");
            }

      </script>

   </body>
</html>
```

if...else if... statement

- The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows –

```
if (expression 1) {
Statement(s) to be executed if expression 1 is true
 } else if (expression 2) {
Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
Statement(s) to be executed if expression 3 is true
} else {
Statement(s) to be executed if no expression is true
 }
```

if is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

EXAMPLE

```
<html>
  <body>
    <script type = "text/javascript">

        var dept = "CSE";
        if( dept == "ece" ) {
          document.write("<b>ece</b>");
        } else if( dept == "CSE" ) {
          document.write("<b>CSE</b>");
        } else if( dept == "CIVIL" ) {
          document.write("<b>CIVIL</b>");
        } else {
          document.write("<b>Unknown</b>");
        }

    </script>

  </body>
</html>
```

SWITCH CASE

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

SYNTAX:

```
switch (expression) {
case condition 1: statement(s)
break;

case condition 2: statement(s)
break;
 ...
case condition n: statement(s)
break;

default: statement(s)
 }
```

Repeat Loops

- A repeat loop is a group of statements that is repeated until a specified condition is met
- Repeat loops are very powerful programming tools; They allow for more efficient program design and are ideally suited for working with arrays.

The While Loop

- The while loop is used to execute a block of code while a certain **condition** is true

```
count = 0;
while (count <= 10) {
    document.write(count);
    count++;
}
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

THE FOR LOOP

The '**for**' loop is the most compact form of looping. It includes the following three important parts –

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Syntax

The syntax of **for** loop is JavaScript is as follows –

```
for (initialization; test condition; iteration statement) {
Statement(s) to be executed if test condition is true
}
```

Example: For Loop

```
// Print the numbers 1
 through 10

for (i=1; i<= 10; i++)
 document.write(i);
```

i=1 initializes the counter

i<=10 is the target value

i++ updates the counter at the end of the loop

```
<SCRIPT>
document.write("1");
document.write("2");
document.write("3");
document.write("4");
document.write("5");
</SCRIPT>
```

```
<SCRIPT>

for (i=1; i<=5; i++)
 document.write(i);
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
const dept = ["CSE", "CIVIL", "ECE", "EEE", "EIE", "MECHANICAL"];

let text = "";
for (let i = 0; i < dept.length; i++) {
  text += dept[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

OBJECTS

- Objects have attributes and methods.
- Many pre-defined objects and object types.
- Using objects follows the syntax of C++/Java:

  objectname.attributename

  objectname.methodname()

The **document** Object

- Many attributes of the current document are available via the **document** object:

Title	**URL**	**Images**
Forms	**Links**	**Colors**

document.links	Returns a collection of all <a> and <area> elements in the document that have a href attribute
document.title	Sets or returns the title of the document
document.URL	Returns the full URL of the HTML document
document.write()	Writes HTML expressions or JavaScript code to a document
document.writeln()	Same as write(), but adds a newline character after each statement

THE DOCUMENT METHODS

- **document.write()** like a print statement – the output goes into the HTML document.
- **document.writeln()** adds a newline after printing.

document.write("My title is" + document.title);

THE **MATH** OBJECT

- Access to mathematical functions and constants.
- Constants: **`Math.PI`**
- Methods:

```
Math.abs(), Math.sin(), Math.log(), Math.max(),
  Math.pow(), Math.random(), Math.sqrt(), …
```

-

Creating **Array** Objects

- With the **`new`** operator and a size:

```
var x = new Array(10);
```

- With the new operator and an initial set of element values:

```
var y = new Array(18,”hi”,22);
```

- Assignment of an *array literal*

```
var x = [1,0,2];
```

JAVASCRIPT OBJECTS

- JavaScript is an Object Oriented Programming (OOP) language.
- Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object.

The syntax for adding a property to an object is:

- objectName.objectProperty = propertyValue;
- document.write("This is test");--------object method

User-defined Objects

- All user-defined objects and built-in objects are descendants of an object called **Object**.
- The **new** operator is used to create an instance of an object.
- To create an object, the **new** operator is followed by the constructor method.

More Objects

The constructor methods are Object(), Array(), and Date() and String(). These constructors are built-in JavaScript functions.

- var employee = new Object();
- var books = new Array("C++", "Perl", "Java");
- var day = new Date("August 15, 1947");

- Objects are variables too. But objects can contain many values.
- The values are written as **name : value** pairs (name and value separated by a colon).

Object Definition

You define (and create) a JavaScript object with an object literal:

Example

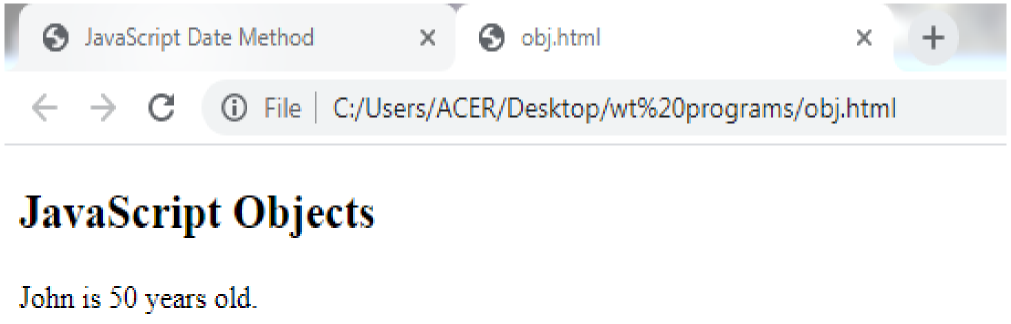
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Spaces and line breaks are not important. An object definition can span multiple lines:

```
var person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    eyeColor: "blue"
  };
```

EXAMPLE:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```



```
JavaScript Objects

John is 50 years old.
```

Object Properties

- The **name:values** pairs in JavaScript objects are called **properties**:

Property name	Property value
First name	John
Last Name	Doe
Age	50
Eye Color	white

Accessing Object Properties

You can access object properties in two ways:
(i) objectName.propertyName

Eg: person.lastName;

(ii) objectName["propertyName"]
Eg: person["lastName"];

Object Methods

Objects can also have **methods**.
Methods are **actions** that can be performed on objects.
Methods are stored in properties as **function definitions**

Example: fullNamefunction() -- {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

```
var person = {
      firstName: "John",
      lastName : "Doe",
      id       : 5566,
      fullName : function() {
        return this.firstName + " " + this.lastName;
      }
    };
```

<u>The this Keyword</u>

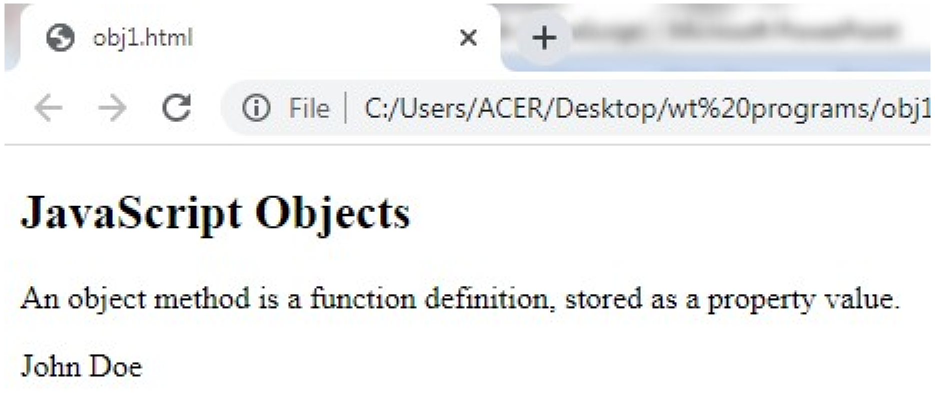
- In a function definition, this refers to the "owner" of the function.
- In the example above, this is the **person object** that "owns" the fullName function.
- In other words, this.firstName means the firstName property of **this object**.

Accessing Object Methods:You access an object method with the following syntax:

objectName.methodName()

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p>An object method is a function definition, stored as a property

value.</p>
<p id="demo"></p>
<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```



JavaScript Objects

An object method is a function definition, stored as a property value.

John Doe

JAVASCRIPT - DOCUMENT OBJECT MODEL OR DOM

- Every web page resides inside a browser window which can be considered as an object.
- A Document object represents the HTML document that is displayed in that window.
- The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**.
- The Objects are organized in a hierarchy.

Window object − Top of the hierarchy. It is the outmost element

Document object − Each HTML document that gets loaded into a window becomes a document object.

Form object − Everything enclosed in the <form>...</form> tags

Form control elements − All the elements such as text fields, buttons, radio buttons, and checkboxes.

Common Methods and Properties

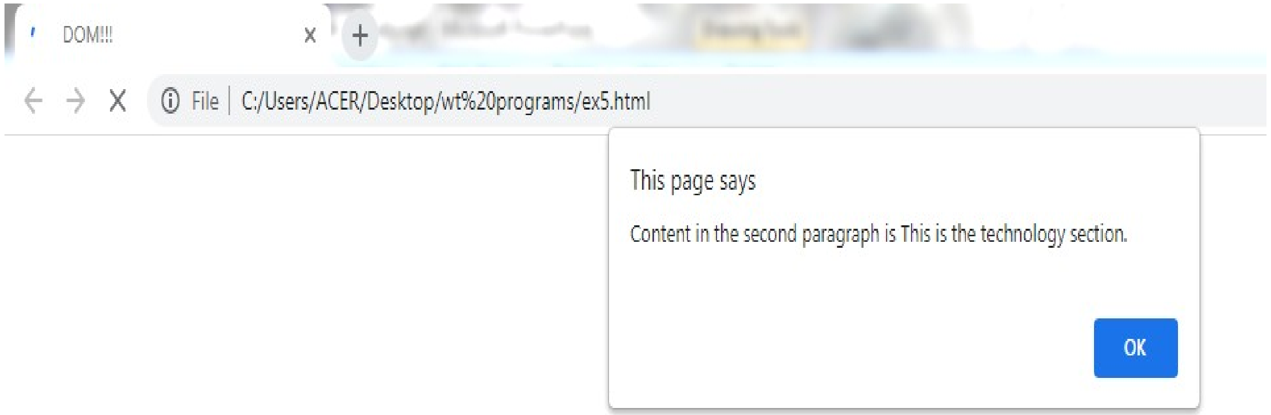
- Finding HTML Elements
- getElementById(): To access elements and attributes with id
- innerHTML: To access the content of an element.
- getElementsByTagName()
- document.getElementsByClassName(*name, value)*
- createElement: To create new element
- removeChild: Remove an element
- an **event handler** to a particular element like this:

document.getElementById(id).addEventListener("click", functionname)

Example:

```
<html>
<head>    <title>DOM!!!</title></head>
<body>
  <p>This is the welcome message.</p>
```

```
<p id="second">This is the technology section.</p>
 <input type="button" id="btnClick" value="Click Me!!" />

 <script type="text/javascript">
   var paragraphs = document.getElementsByTagName("p");
   alert("Content in the second paragraph is " + paragraphs[1].innerHTML);
   document.getElementById("second").innerHTML = "The orginal message is
   changed.";

  document.getElementById("btnClick").addEventListener("click", clicked);
   function clicked()   { alert("You clicked me!!!");    }
 </script>
</body>
</html>
```



This page says

Content in the second paragraph is This is the technology section.

CHANGING HTML ELEMENTS

- **Method Description**
- *element*.innerHTML = *new html content*
- *element.attribute = new value*
- *element*.setAttribute*(attribute, value)*
- *element*.style.*property = new style*
- **Adding and Deleting Elements**
- document.createElement(*element*)
- document.removeChild(*element*)
- document.appendChild(*element*)
- document.replaceChild(*element*)
- document.write(*text*)

JavaScript HTML DOM Events

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded

- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When an reset button is clicked.
- When focused on an element.

EVENTLISTENERS

- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The addEventListener() method makes it easier to control how the event reacts.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.

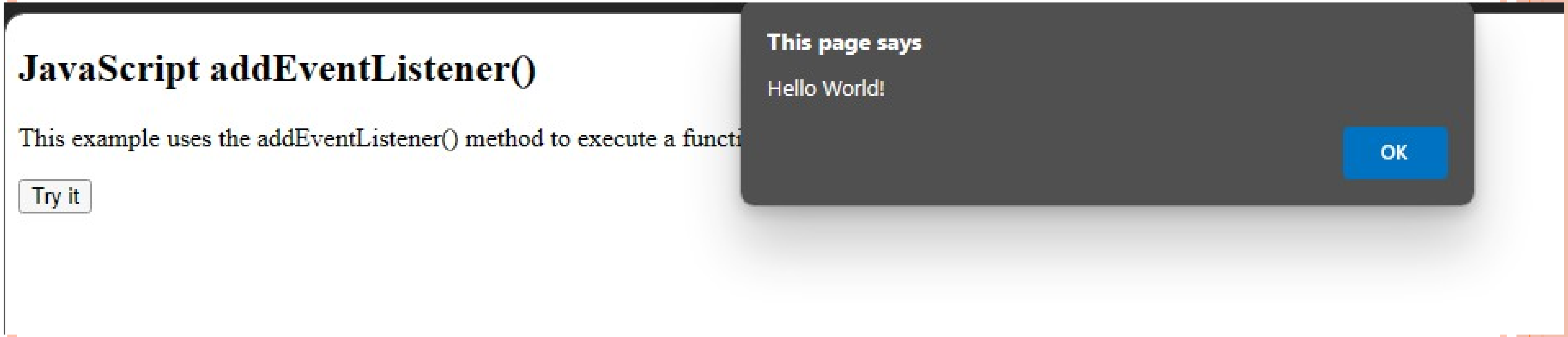
Syntax

***element*.addEventListener(*event*, *function*, *useCapture*);**

- The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

ADDING EVENTLISTENER

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to execute a
   function when a user clicks on a button.</p>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click",
   myFunction);
function myFunction() {
  alert ("Hello World!");
}
</script>
</body>
</html>
```



JavaScript addEventListener()

This example uses the addEventListener() method to execute a functi

Try it

This page says

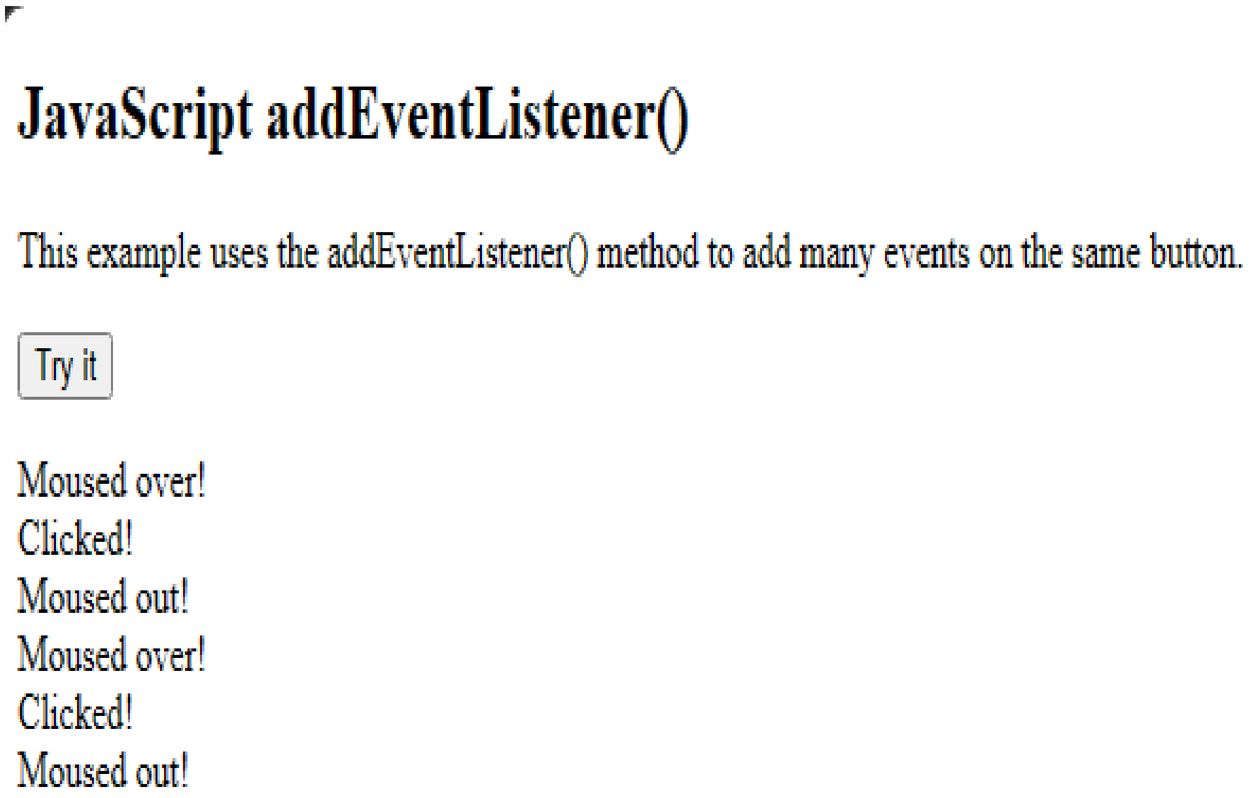
Hello World!

OK

Add many Event Handlers to the same element

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to add many events
   on the same button.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);
```

```
function myFunction() {
  document.getElementById("demo").innerHTML += "Moused
   over!<br>";
}
function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!
   <br>";
}
function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused
   out!<br>";
}
</script>
</body>
</html>
```



JavaScript addEventListener()

This example uses the addEventListener() method to add many events on the same button.

```
Moused over!
Clicked!
Moused out!
Moused over!
Clicked!
Moused out!
```