Chapter 1

Basics of JavaScript

CO(COURSE OUTCOME)

CO1 Build interactive web pages using program flow control structure.

CO2 Implement Arrays , functions and create event based web forms using Java Script.

CO3 Use JavaScript for browser data persistence.

CO4 Create menus, navigation in interactive webpages using regular expressions for

JavaScript

- JavaScript is an object-based scripting language which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language.
- The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

Features of Javascript

Scripting
Javascript executes the client-side script in the browser.

Interpreter
The browser interprets JavaScript code.

Event Handling
Events are actions. Javascript provides event-handling options.

Light Weight
As Javascript is not a compiled language, source code never changes to byte code before running time. Low-end devices can also run Javascript because of its lightweight feature.

Case Sensitive
In Javascript, names, variables, keywords, and functions are case-sensitive.

Control Statements
Javascript has control statements like if-else-if, switch case, and loop. Users can write complex code using these control statements.

Objects as first-class Citizens
Javascript arrays, functions, and symbols are objects which can inherit the Object prototype properties. Objects being first-class citizens means Objects can do all tasks.

Supports Functional Programming
Javascript functions can be an argument to another function, can call by reference, and can assign to a variable.

JavaScript Advantages

Following are the advantages of JavaScript –

•**Simple** − JavaScript is simple to comprehend and pick up. Both users and developers will find the structure to be straightforward. Additionally, it is very doable to implement, saving web developers a tonne of money when creating dynamic content.

•**Speed** − JavaScript is a "interpreted" language, it cuts down on the time needed for compilation in other programming languages like Java. Another client-side script is JavaScript, which accelerates programme execution by eliminating the wait time for server connections.
•No matter where JavaScript is hosted, it is always run in a client environment to reduce bandwidth usage and speed up execution.

•**Interoperability** − Because JavaScript seamlessly integrates with other programming languages, many developers favour using it to create a variety of applications. Any webpage or the script of another programming language can contain it.

•**Server Load** − Data validation can be done within the browser itself rather than being forwarded to the server because JavaScript is client-side. The entire website does not need to be reloaded in the event of any discrepancy. Only the chosen area of the page is updated by the browser.

JavaScript Disadvantages:

Following are the disadvantages of JavaScript –

•**Cannot Debug** – Although some HTML editors allow for debugging, they are not as effective as editors for C or C++. Additionally, the developer has a difficult time figuring out the issue because the browser doesn't display any errors.

•**Unexpected stop of rendering** – The website's entire JavaScript code can stop rendering due to a single error in the code. It appears to the user as though JavaScript is absent. The browsers, however, are very forgiving of these mistakes.

•**Client-side Security** – The user can see the JavaScript code; it could be misused by others. These actions might involve using the source code anonymously. Additionally, it is very simple to insert code into the website that impair the security of data transmitted via the website.

•**Inheritance** – JavaScript does not support multiple inheritance; only one inheritance is supported. This property of object-oriented languages might be necessary for some programmes.

•**Browser Support** – Depending on the browser, JavaScript is interpreted differently. Therefore, before publication, the code needs to run on various platforms. We also need to check the older browsers because some new functions are not supported by them.

JavaScript Objects

Real Life Objects, Properties, and Methods

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

In real life, a car is an **object**.

- A car has **properties** like weight and color, and **methods** like start and stop
- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Variables

Variables are containers for storing data (storing data values)

4 Ways to Declare a JavaScript Variable:

- Using var
- **Using let**
- Using const
- Using nothing.

Examples:

```
In this example, x, y, and z, are variables, declared with the var keyword:
var x = 5;
var y = 6;
var z = x + y;

In this example, x, y, and z, are variables, declared with the let keyword:
let x = 5;
let y = 6;
let z = x + y;
```

In this example, x, y, and z, are undeclared variables:

```
x = 5;
y = 6;
z = x + y;
```

When to Use JavaScript var?

- Always declare JavaScript variables with var, let, or const.
- The var keyword is used in all JavaScript code from 1995 to 2015.
- **The let and const keywords were added to JavaScript in 2015.**
- If you want your code to run in older browsers, you must use var.

When to Use JavaScript const?

- If you want a general rule: always declare variables with const.
- If you think the value of the variable can change, use let.

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

- The two variables price1 and price2 are declared with the const keyword.
- These are constant values and cannot be changed.
- The variable total is declared with the let keyword.
- This is a value that can be changed.

JavaScript Identifiers:

- All JavaScript **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- **Names can contain letters, digits, underscores, and dollar signs.**
- Names must begin with a letter.
- Names can also begin with $ and _.
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note: JavaScript identifiers are case-sensitive.

The Assignment Operator

- In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.
- This is different from algebra. The following does not make sense in algebra:

x = x + 5:

In JavaScript, however, it makes perfect sense: **it assigns the value of x + 5 to x.**

Note: The "equal to" operator is written like == in JavaScript.

JavaScript Data Types

- **Strings are written inside double or single quotes. Numbers are written without quotes.**
- **If you put a number in quotes, it will be treated as a text string.**

Example:

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable:
You declare a JavaScript variable with the var or the let keyword:

```
var carName;     or     let carName;
```

After the declaration, the variable has no value (technically it is undefined).

To assign a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

Note:
It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables
You can declare many variables in one statement.

Start the statement with let and separate the variables by comma:

```
let person = "John Doe", carName = "Volvo", price = 200;
```

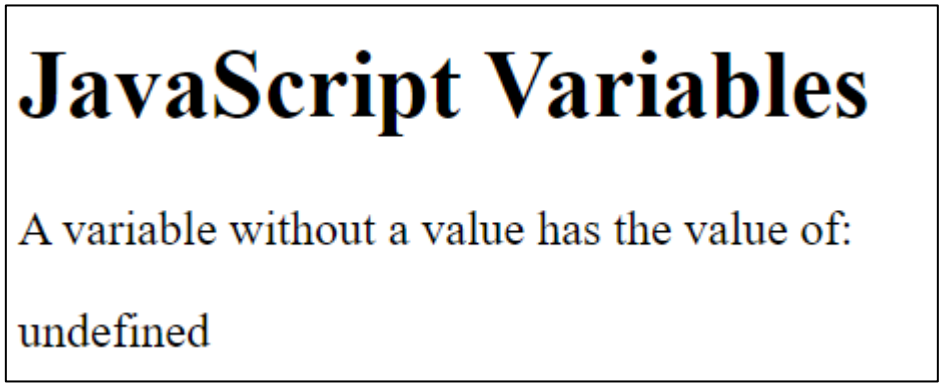
Value = undefine

- **A variable declared without a value will have the value undefined.**

- The variable carName will have the value undefined after the execution of this statement:

```
let carName;
```

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>A variable without a value has the value of:</p>
<p id="demo"></p>

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

</body>
</html>
```



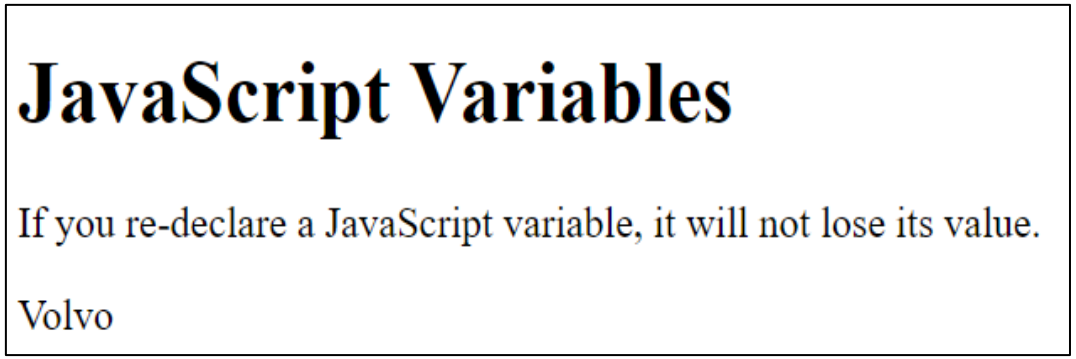
Note: The innerHTML property sets or returns the HTML content (inner HTML) of an element.

Re-Declaring JavaScript Variables:

If you re-declare a JavaScript variable declared with var, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
<html>
<body>

<h1>JavaScript Variables</h1>

<p>If you re-declare a JavaScript variable, it will not lose its value.</p>

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

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

</body>
</html>
```



Difference between var and let

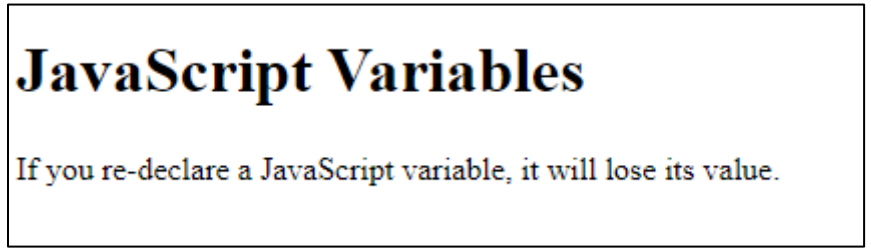
Note

You cannot re-declare a variable declared with let or const.

This will not work:

```
let carName = "Volvo";
let carName;
```

```
<h1>JavaScript Variables</h1>

<p>If you re-declare a JavaScript variable, it will lose its value.</p>

<script>
let carName = "Volvo";
let carName;
document.getElementById("demo").innerHTML = carName;
</script>
```



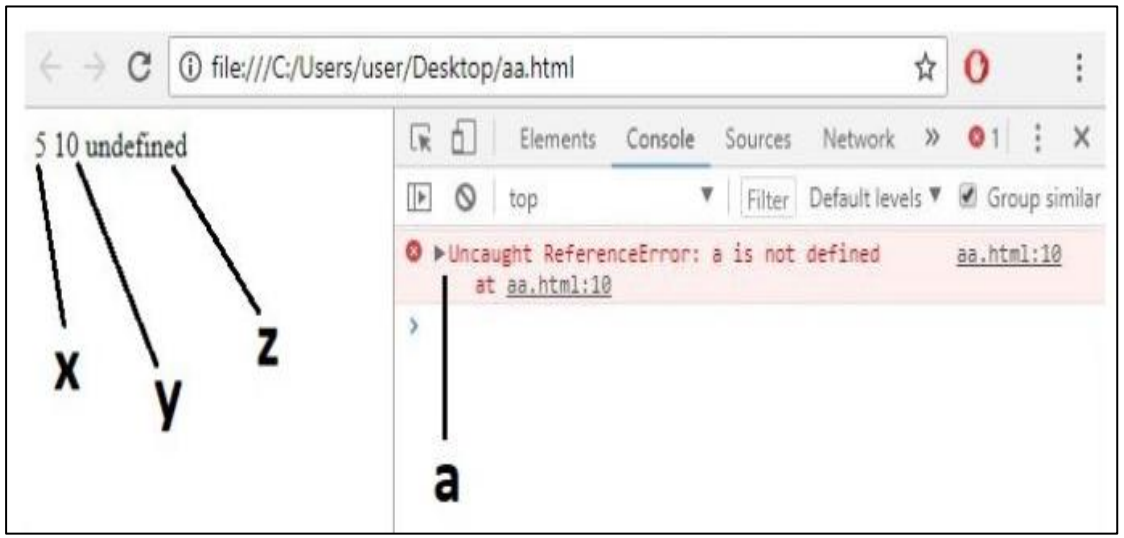
Difference between var and let

```
<script>
// calling x after definition
   var x = 5;
   document.write(x, "\n");

   // calling y after definition
   let y = 10;
   document.write(y, "\n");

   // calling var z before definition will return undefined
   document.write(z, "\n");
   var z = 2;

   // calling let a before definition will give error
   document.write(a);
   let a = 3;
</script>
```

JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

5 10 undefined



JavaScript Arithmetic:

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +

```
let x = 5 + 2 + 3;.........................10
```

You can also add strings, but strings will be concatenated:

```
let x = "John" + " " + "Doe"; ............. John Doe
```

```
let x = "5" + 2 + 3; ..................... 523
```

Note
If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators:

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

OUTPUT:

450

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The Addition Assignment Operator (+=) adds a value to a variable.

```
let x = 10;
x += 5;
```

OUTPUT

15

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Adding JavaScript Strings

The + operator can also be used to add (concatenate) strings.

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

OUTPUT:

```
John Doe
```

The += assignment operator can also be used to add (concatenate) strings:

```
let text1 = "What a very ";
text1 += "nice day";
```

OUTPUT:

```
What a very nice day
```

Note: When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

If you add a number and a string, the result will be a string!

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Example 1:

```
<p>Assign 5 to x, and display the value of the comparison (x === 5):</p>

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

<script>
let x = 5;
document.getElementById("demo").innerHTML = (x === 5);
</script>
```

OUTPUT

Assign 5 to x, and display the value of the comparison (x === 5):

true

Example 2:

```
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x === "5");
</script>
```

OUTPUT

false

Example 3:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== 5):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== 5);
</script>
</body>
</html>
```

OUTPUT

JavaScript Comparison
The !== Operator
Assign 5 to x, and display the value of the comparison (x !== 5):
false

Example 4:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== "5"):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== "5");
</script>
</body>
</html>
```

OUTPUT

JavaScript Comparison
The !== Operator
Assign 5 to x, and display the value of the comparison (x !== "5"):
true

Example 5:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== 8):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== 8);
</script>
</body>
</html>
```

OUTPUT

JavaScript Comparison

The !== Operator

Assign 5 to x, and display the value of the comparison (x !== 8):

true

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

The && Operator (Logical AND)

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The && Operator (Logical AND)</h2>
<p>The && operator returns true if both expressions are true, otherwise it returns false.</p><p id="demo"></p>
<p id="demo"></p>
<script>
let x = 6;
let y = 3;
document.getElementById("demo").innerHTML =
(x < 10 && y > 1) + "<br>" +
(x < 10 && y < 1);
</script>
</body>
</html>
```

OUTPUT

JavaScript Comparison

The && Operator (Logical AND)

The && operator returns true if both expressions are true, otherwise it returns false.

true

false

The || Operator (Logical OR)

```
<h1>JavaScript Comparison</h1>
<h2>The || Operator (Logical OR)</h2>
<p>The || returns true if one or both expressions are true, otherwise it returns false.</p>
<p id="demo"></p>
<script>
let x = 6;
let y = 3;
document.getElementById("demo").innerHTML =
(x == 5 || y == 5) + "<br>" +
(x == 6 || y == 0) + "<br>" +
(x == 0 || y == 3) + "<br>" +
(x == 6 || y == 3);
</script>
```

OUTPUT

JavaScript Comparison

The || Operator (Logical OR)

The || returns true if one or both expressions are true, otherwise it returns false.

false
true
true
true

The NOT operator (!)

```
<h2>JavaScript Comparison</h2>

<p>The NOT operator (!) returns true for false statements and false for true statements.</p>

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

<script>
let x = 6;
let y = 3;

document.getElementById("demo").innerHTML =
!(x === y) + "<br>" +
!(x > y);
</script>
```

OUTPUT:

JavaScript Comparison

The NOT operator (!) returns true for false statements and false for true statements.

true

false

Ternary operator?

A ternary operator evaluates a condition and executes a block of code based on the condition.

Syntax:

condition ? expression1 : expression2

The ternary operator evaluates the test condition.

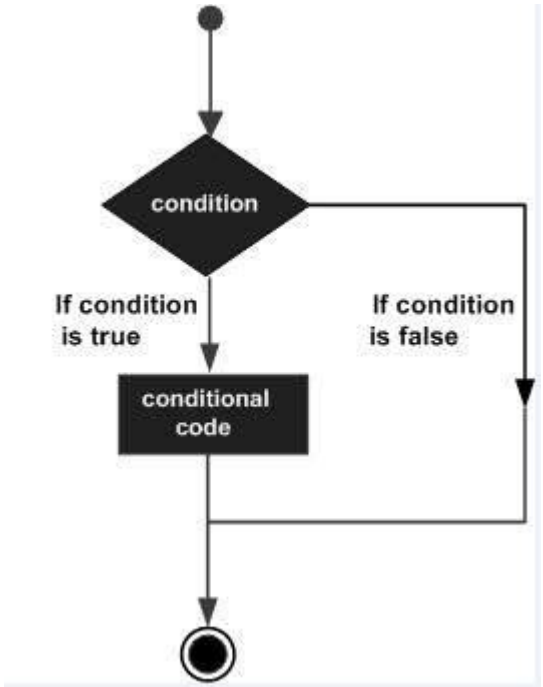
If the condition is true, expression1 is executed.
If the condition is false, expression2 is executed.

The ternary operator takes three operands, hence, the name ternary operator. It is also known as a conditional operator.

Control structures and looping:

While writing a program, there may be a situation when you need to adopt one out of a given set of paths.
In such cases, you need to use conditional statements that allow your program to make correct decisions
and perform right actions.
JavaScript supports conditional statements which are used to perform different actions based on different conditions.

1) **if..else** statement.



Example

Try the following example to understand how the **if** statement works.

```
<html>
  <body>

    <script type="text/javascript">
          var age = 20;

        if( age > 18 )
          {
          document.write("<b>Qualifies for driving</b>");
          }



          </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
          var age = 20;

       if( age > 18 ){
         document.write("<b>Qualifies for driving</b>");
       }

       else{
         document.write("<b>Does not qualify for driving</b>");
       }
          </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

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

        var book = "maths";
      if( book == "history" ){
        document.write("<b>History Book</b>");
      }

      else if( book == "maths" ){
        document.write("<b>Maths Book</b>");
      }

      else if( book == "economics" ){
        document.write("<b>Economics Book</b>");
      }

      else{
        document.write("<b>Unknown Book</b>");
      }
</script>
```

The JavaScript Switch Statement

- The switch statement is used to perform different actions based on different conditions.
- Use switch to specify many alternative blocks of code to be executed

Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

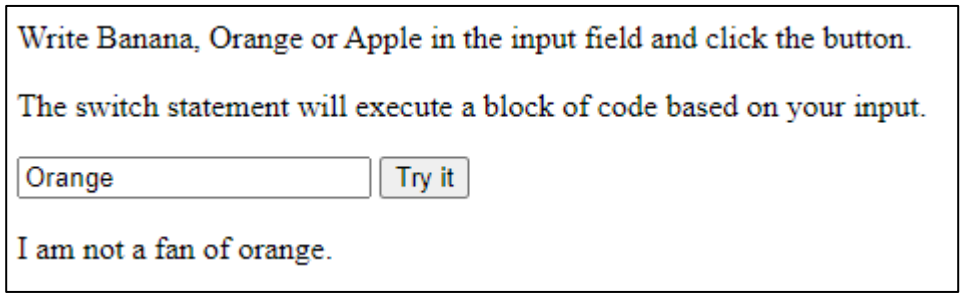
This is how it works:

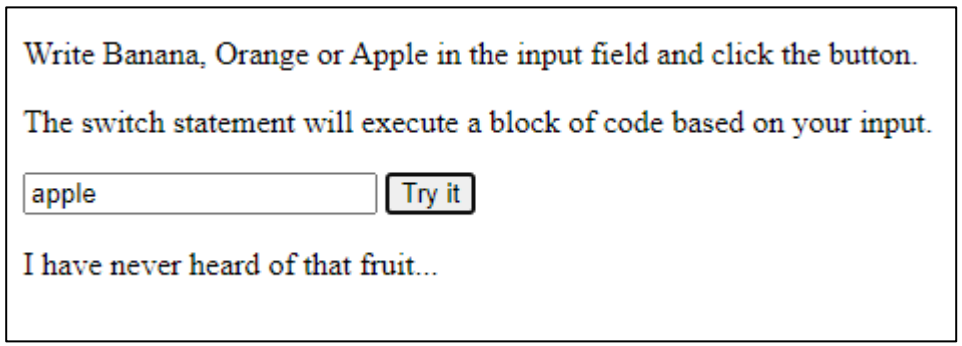
- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

SWITCH CASE:

```
<html>
<body>
<input id="myInput" type="text">
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var text;
  var fruits = document.getElementById("myInput").value;
  switch(fruits) {
    case "Banana":
      text = "Banana is good!";
    break;
    case "Orange":
    text = "I am not a fan of orange.";
    break;
    case "Apple":
    text = "How you like them apples?";
    break;
    default:
    text = "I have never heard of that fruit...";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

OUTPUT





- The switch statement executes a block of code depending on different cases.

- The switch statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions. Use switch to select one of many blocks of code to be executed. This is the perfect solution for long, nested if/else statements.

- The switch statement evaluates an expression. The value of the expression is then compared with the values of each case in the structure. If there is a match, the associated block of code is executed.

- The switch statement is often used together with a break or a default keyword (or both). These are both optional:

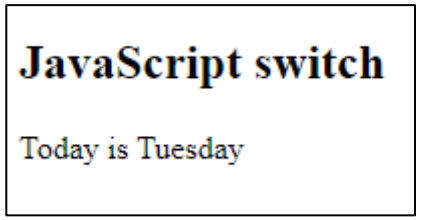
- The **break keyword** breaks out of the switch block. This will stop the execution of more execution of code and/or case testing inside the block. If break is omitted, the next code block in the switch statement is executed.

- The **default keyword** specifies some code to run if there is no case match. There can only be one default keyword in a switch. Although this is optional, it is recommended that you use it, as it takes care of unexpected cases.

Example

- The getDay() method returns the weekday as a number between 0 and 6.
- (Sunday=0, Monday=1, Tuesday=2 ..)
- This example uses the weekday number to calculate the weekday name:

```
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
</html>
```



The break Keyword

- When JavaScript reaches a break keyword, it breaks out of the switch block.
- This will stop the execution inside the switch block.
- It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword

- The default keyword specifies the code to run if there is no case match:

- The getDay() method returns the weekday as a number between 0 and 6.

- If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Note:If default is not the last case in the switch block, remember to end the default case with a break.

```
<html>
<body>

<h2>JavaScript switch</h2>

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

<script>
let text;
switch (new Date().getDay()) {
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

OUTPUT:

JavaScript switch
Looking forward to the Weekend

Common Code Blocks:

- Sometimes you will want different switch cases to use the same code.
- In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

```
<script>
let text;
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>
```

OUTPUT:

JavaScript switch
Looking forward to the Weekend

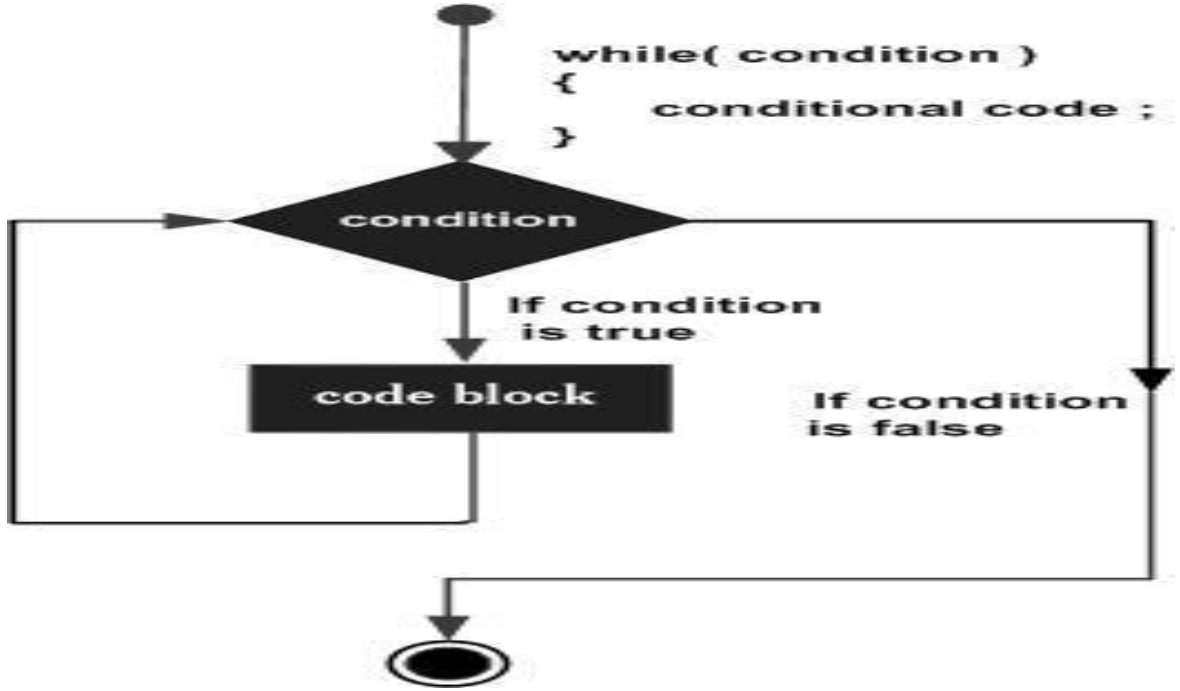
Switching Details

- If multiple cases matches a case value, the first case is selected.
- If no matching cases are found, the program continues to the default label.
- If no default label is found, the program continues to the statement(s) after the switch.

The while Loop:

The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

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



The while Loop

Example:

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

var count = 0;
document.write("Starting Loop ");
while (count < 10)
{
document.write("Current Count : " + count + "<br />");
count++;
}
document.write("Loop stopped!");

</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

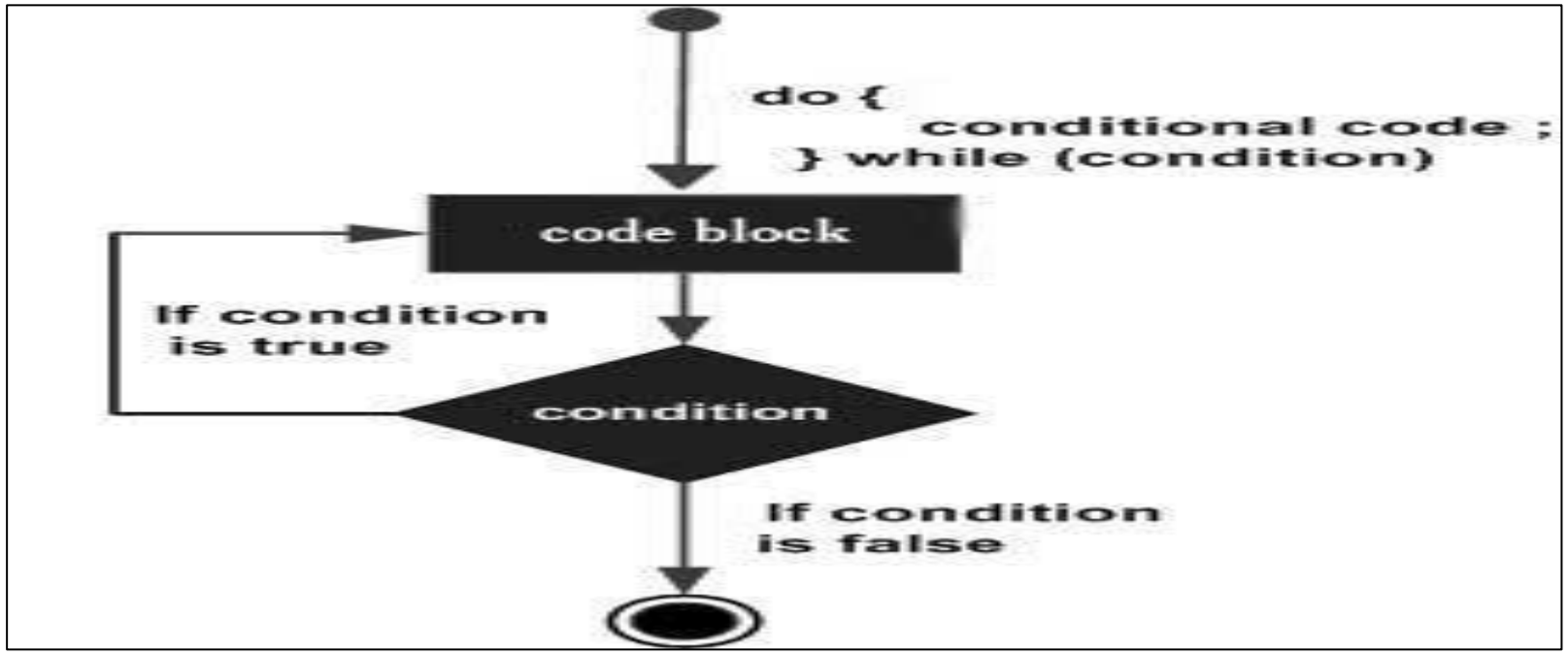
OUTPUT:

Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!Set the variable to different value and then try...

The do...while Loop:

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

```
do{
   Statement(s) to be executed;
}
while (expression);
```



The do...while Loop:

```
<html>
  <body>

    <script type="text/javascript">

          var count = 0;

        document.write("Starting Loop" + "<br />");
        do{
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        while (count < 5);
        document.write ("Loop stopped!");
   </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then
try...
```

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.

FOR LOOP

```
<html>
<body>
 <script type="text/javascript">
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++)
{
document.write("Current Count : " + count );
document.write("<br />");
}
 document.write("Loop stopped!");
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

OUTPUT:

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...

For Loop with -- operator

```
<html>
<body>
 <script type="text/javascript">
var count;
document.write("Starting Loop" + "<br />");
for(count = 10; count > 0; count--)
{
document.write("Current Count : " + count );
document.write("<br />");
}
 document.write("Loop stopped!");
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Starting Loop
Current Count : 10
Current Count : 9
Current Count : 8
Current Count : 7
Current Count : 6
Current Count : 5
Current Count : 4
Current Count : 3
Current Count : 2
Current Count : 1
Loop stopped!

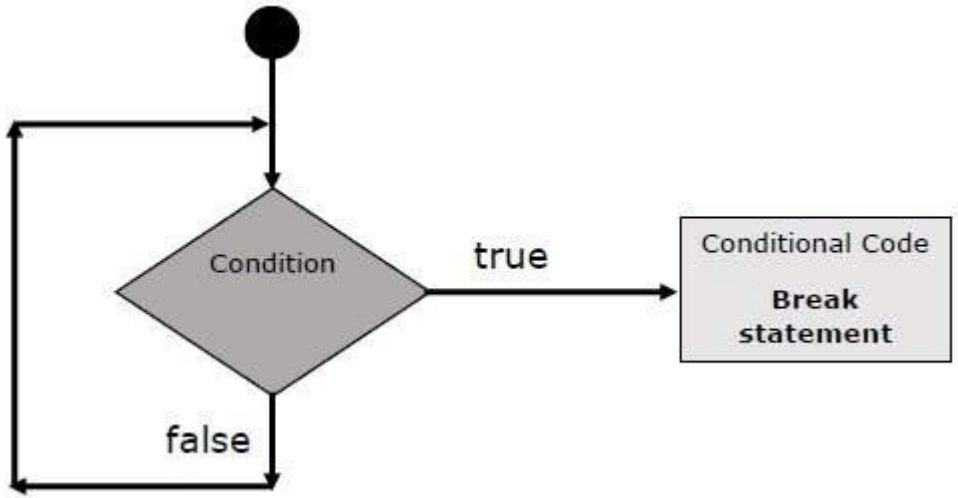
Set the variable to different value and then try...

The break Statement:

JavaScript provides full control to handle loops. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

To handle all such situations, JavaScript provides **break** and **continue**statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement



Example:

```
<html>
  <body>
        <script type="text/javascript">
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 20)
      {
        if (x == 5){
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
        document.write("Exiting the loop!<br /> ");
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

OUTPUT:

Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...

The continue Statement

The continue statement breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop.
The difference between continue and the break statement, is instead of "jumping out" of a loop, the continue statement "jumps over" one iteration in the loop.

```
<html>
  <body>

    <script type="text/javascript">
        var x = 1;
        document.write("Entering the loop<br /> ");

        while (x < 10)
        {
          x = x + 1;

          if (x == 5){
            continue; // skip rest of the loop body
          }
          document.write( x + "<br />");
        }

        document.write("Exiting the loop!<br /> ");
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

OUTPUT:

Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try...

Write javascript to print following pattern

*****
*****
*****
*****
*****

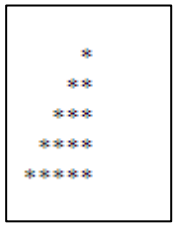
```
<html>
<body>
<script>
let n = 5; // row or column count
// defining an empty string
let string = "";

for(let i = 0; i < n; i++) { // external loop
  for(let j = 0; j < n; j++) { // internal loop
    string += "*";
  }
  // newline after each row
  string += "<br>";
}
// printing the string
document.write(string);
</script>
</body>
</html>
```

Write javascript to print following pattern

```
*
**
***
****
*****
```

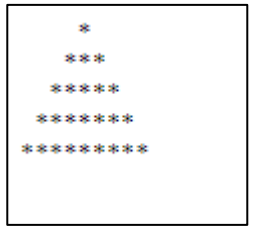
```
<html>
<head>
   <title>right triangle star pattern - javascript</title>
</head>
<body>
  <h2>Right triangle star pattern in javascript</h2>
  <script>
   let n = 5; // you can take input from prompt or
change the value
   let string = "";
   for (let i = 0; i < n; i++) {
     for (let j = 0; j <= i; j++) {
       string += "*";
     }
     string += "<br>";
   }
   document.write(string);
  </script>
</body>
</html>
```

Write javascript to print following pattern



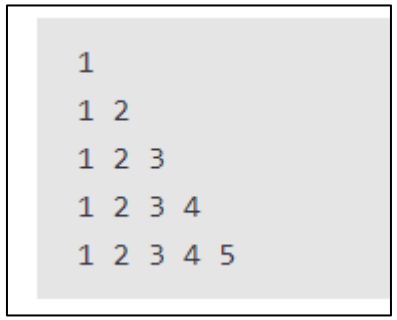

```
<html>
<body>
 <h2>Left triangle star pattern in javascript</h2>
 <script>
  let n = 5; // you can take input from prompt or change the value
  let string = "";
  for (let i = 1; i <= n; i++) {
    // printing spaces
    for (let j = 0; j < n - i; j++) {
      string += " ";
    }
    // printing star
    for (let k = 0; k < i; k++) {
      string += "*";
    }
    string += "<br>";
  }
  document.write(`<pre>${string}</pre>`);
  //document.write(string);
 </script>
</body>
</html>
```

Write javascript to print following pattern



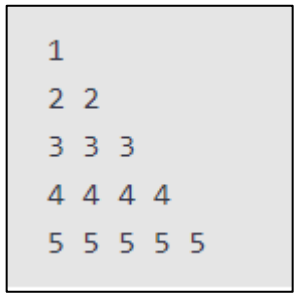

```
<html>
<body>
<script>
   let n = 5; // you can take input from prompt or change the value
   let string = "";
   // External loop
   for (let i = 1; i <= n; i++) {
     // printing spaces
     for (let j = n; j > i; j--) {
       string += " ";
     }
     // printing star
     for (let k = 0; k < i * 2 - 1; k++) {
       string += "*";
     }
     string += "<br>";
   }
   document.write(`<pre>${string}</pre>`);
   //document.write(string);
  </script>
</body>
</html>
```

Write javascript to print following pattern




```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 5; // height of pattern
  let string = "";
  // External loop
  for (let i = 1; i <= n; i++) {
    // Internal loop
    for (let j = 1; j <= i; j++) {
      string += j;
    }
    string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

Write javascript to print following pattern



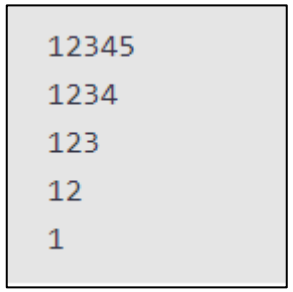

```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 5; // height of pattern
  let string = "";
  // External loop
  for (let i = 1; i <= n; i++) {
    // Internal loop
    for (let j = 1; j <= i; j++) {
      string += i;
    }
    string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

Write javascript to print following pattern

```
1
2 3
4 5 6
7 8 9 10
```

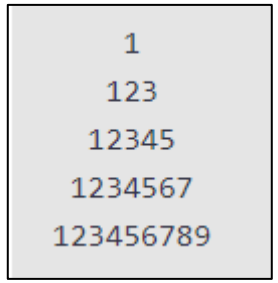
```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 4; // height of pattern
  let string = "";
  let count = 1;
  // External loop
  for (let i = 1; i <= n; i++) {
   // Internal loop
   for (let j = 1; j <= i; j++) {
    string += count;
    count++;
   }
   string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

Write javascript to print following pattern




```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 5; // height of pattern
  let string = "";
  // External loop
  for (let i = 1; i <= n; i++) {
    for (let j = 1; j <= n - i + 1; j++) {
      string += j;
    }
    string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

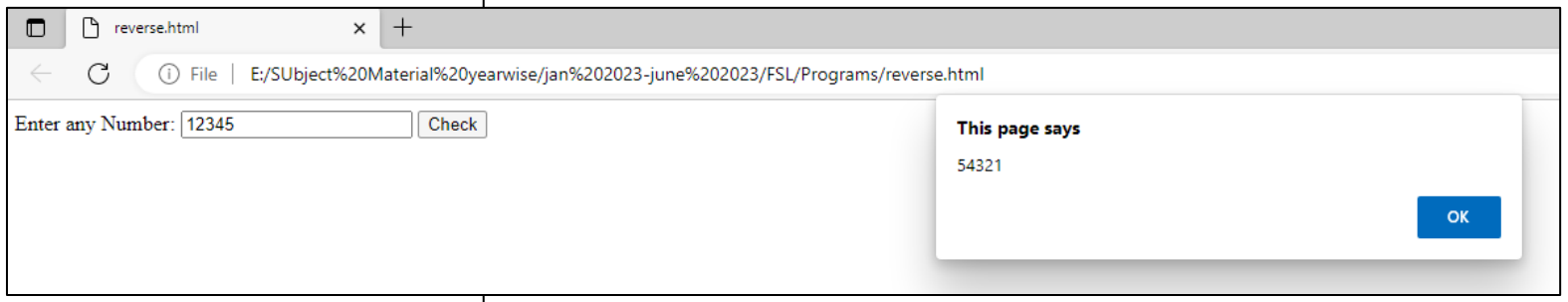
Write javascript to print following pattern




```
<html>
<body>
   <script>
   let n = 5;
   let string = "";
   // External loop
   for (let i = 1; i <= n; i++) {
    // creating spaces
    for (let j = 1; j <= n - i; j++) {
      string += " ";
    }
    // creating alphabets
    for (let k = 1; k <= 2 * i - 1; k++) {
      string += k;
    }
    string += "<br>";
   }
   document.write(string);
  </script>
</body>
</html>
```

Write a javascript to reverse a given nos.

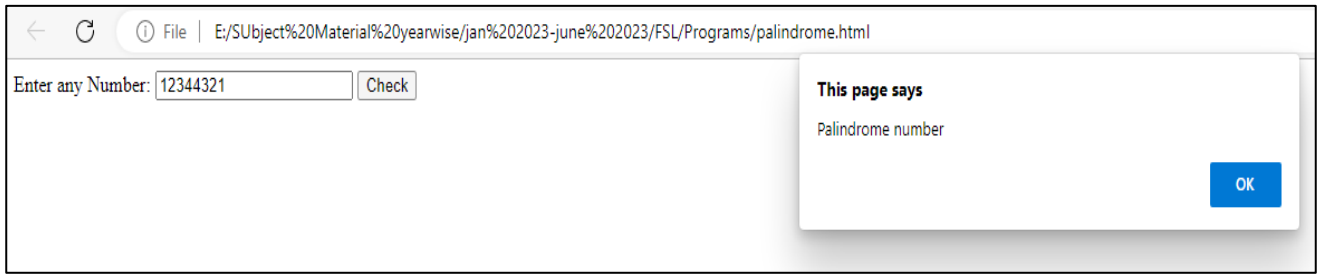
```
<html>
<head>
<script>
function palin()
{
var a,no,b,temp=0;
no=parseInt (document.getElementById("no_input").value);
b=no;
while(no>0)
{
a=no%10;
no=parseInt(no/10);
temp=temp*10+a;
}
alert(temp);
}
</script>
</head>
<body>
Enter any Number: <input id="no_input">
<button onclick="palin()">Check</button></br></br>
</body>
</html>
```



Write a javascript to check given no is palindrome or not

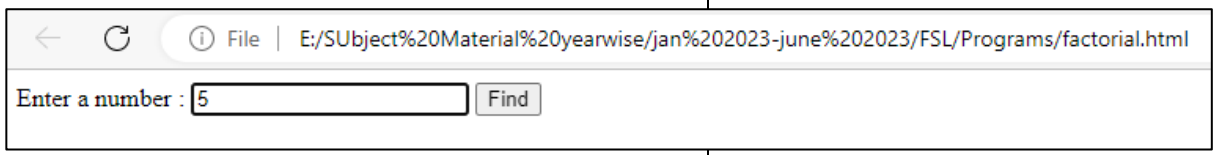
```
<html>
<head>
<script>
function palin()
{
var a,no,b,temp=0;
no=parseInt(document.getElementById("no_input").value);
b=no;
while(no>0)
{
a=no%10;
no=parseInt(no/10);
temp=temp*10+a;
}
if(temp==b)
{
alert("Palindrome number");
}
else
{
alert("Not Palindrome number");
}
}
</script>
</head>
```

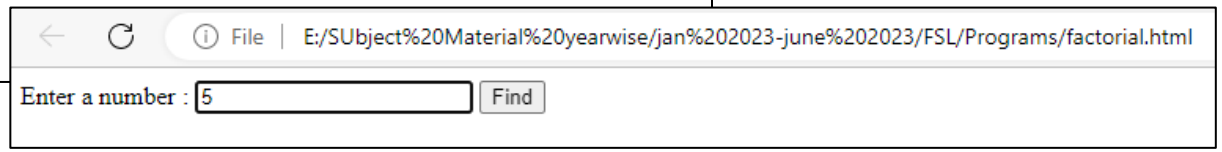
```
<body>
Enter any Number: <input id="no_input">
<button onclick="palin()">Check</button></br></br>
</body>
</html>
```

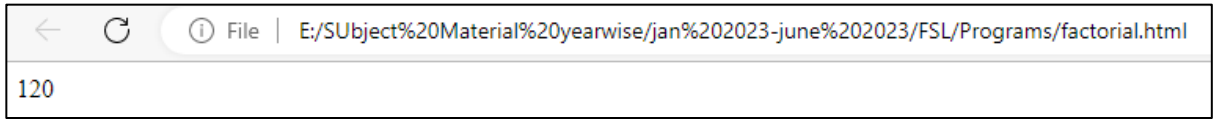


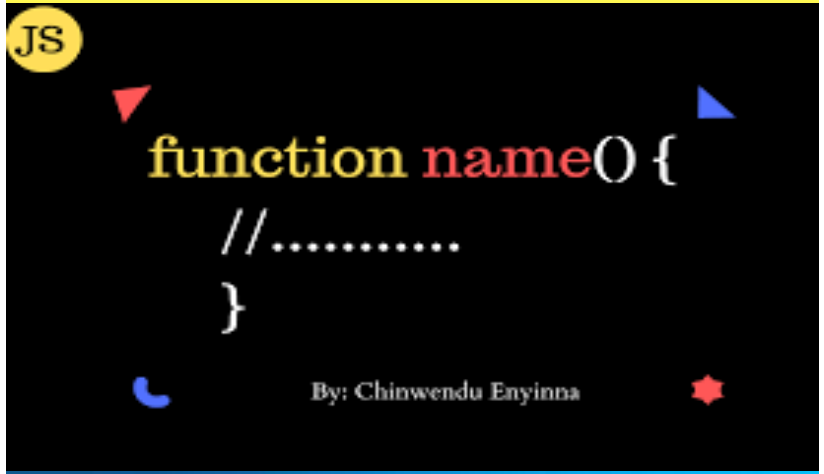
Factorial of a given no

```
<html>
<body>
<script>
function factor()
{
var f = 1;
var n = parseInt(document.getElementById("num").value);
for(i = 1;i<=n;i++)
{
   f = f*i ;
}
document.write (f);
}
</script>
<form>
Enter a number : <input type = "text" id = "num">
<input type = "button" value = "Find" onclick = "factor()" >
</form>
</body>
</html>
```









Chapter 2

Array, Function and String

Arrays in JavaScript:

- JavaScript array is a single variable that is used to store different elements. It is often used when we want to store a list of elements and access them by a single variable.

- In JavaScript, an array is a single variable that stores multiple elements.

- An array is a special variable, which can hold more than one value:

const cars = ["Saab", "Volvo", "BMW"];

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Declaration of an Array: There are basically two ways to declare an array.

```
const arrayName = [value1, value2, ...]; // Method 1
```

```
const arrayName = new Array(value1, value2, ...);// Method 2
```

Example:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

You can also create an array, and then provide the elements:

Example:

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Example:

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Using the JavaScript Keyword new:

The following example also creates an Array, and assigns values to it:

const cars = new Array("Saab", "Volvo", "BMW");

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays
Saab,Volvo,BMW

For simplicity, readability and execution speed, use the array literal method.

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
```

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

OUTPUT:

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

Saab

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

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

```
cars[0] = "Opel";
```

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

OUTPUT

JavaScript Arrays

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

Opel,Volvo,BMW

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
<html>
<body>

<h2>JavaScript Arrays</h2>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

OUTPUT

JavaScript Arrays

Saab,Volvo,BMW

Arrays are Objects

- Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.
- But, JavaScript arrays are best described as arrays.
- Arrays use numbers to access its "elements". In this example, person[0] returns John:

```
<script>
const person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
</script>
```

John

Objects use names to access its "members". In this example, person.firstName returns John:

```
<script>
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML =
person.firstName;
</script>
```

John

Array Elements Can Be Objects

- Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
<html>
<body>

<h2>JavaScript Arrays</h2>
<p>The length property returns the length of an array.</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits.length;
</script>

</body>
</html>
```

OUTPUT

JavaScript Arrays
The length property returns the length of an array.

4

Note:The length property is always one more than the highest array index.

Accessing the First Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

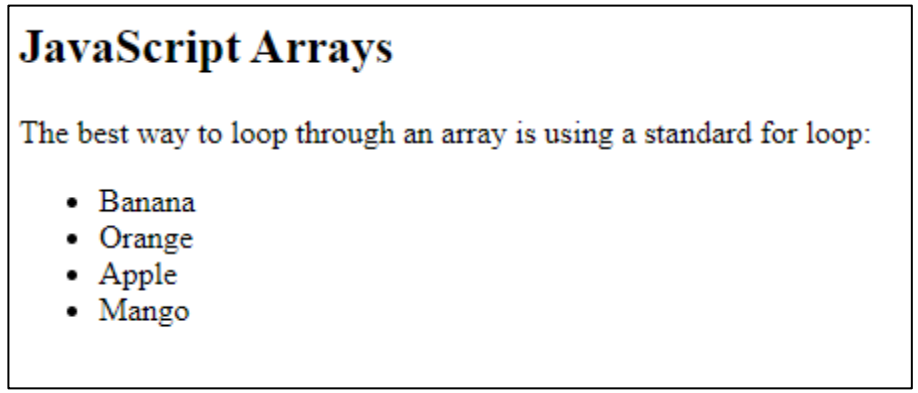
Accessing the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Looping Array Elements

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for loop:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;
let text = "<ul>";
for (let i = 0; i < fLen; i++)
{
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



JavaScript Array forEach()

Definition and Usage

- The forEach() method calls a function for each element in an array.
- The forEach() method is not executed for empty elements.

Syntax

```
array.forEach(function(currentValue, index, arr), thisValue)
```

Parameters

function()	Required. A function to run for each array element.
currentValue	Required. The value of the current element.
index	Optional. The index of the current element.
arr	Optional. The array of the current element.
thisValue	Optional. Default `undefined`. A value passed to the function as its `this` value.

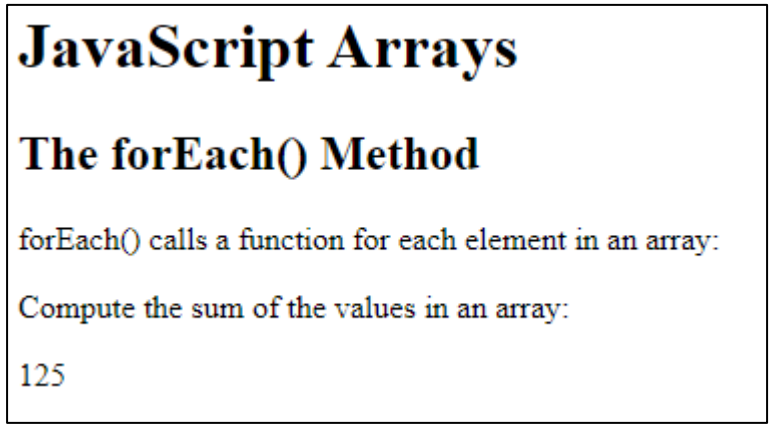
JavaScript Array forEach()

Calls a function for each element in fruits:

```
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p id="demo"></p>
<script>
let text = "";
const fruits = ["apple", "orange", "cherry"];
fruits.forEach(myFunction);
document.getElementById("demo").innerHTML = text;
function myFunction(item, index)
{
  text += index + ": " + item + "<br>";
}
</script>
</body>
</html>
```

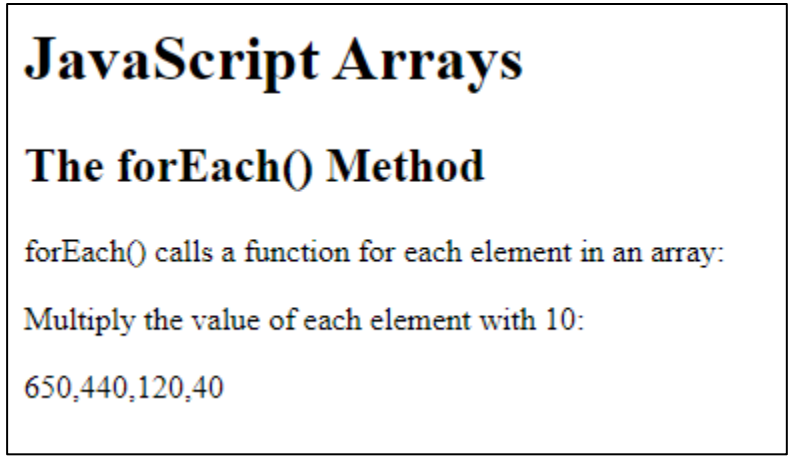
Compute the sum: using FOR-EACH

```
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p>Compute the sum of the values in an array:</p>
<p id="demo"></p>
<script>
let sum = 0;
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = sum;
function myFunction(item)
{
  sum += item;
}
</script>
</body>
</html>
```



Multiply each element:

```
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>forEach() calls a function for each element in an array:</p>
<p>Multiply the value of each element with 10:</p>
<p id="demo"></p>
<script>
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction)
document.getElementById("demo").innerHTML = numbers;
function myFunction(item, index, arr) {
  arr[index] = item * 10;
}
</script>
</body>
</html>
```



JavaScript for...in Loop

Definition and Usage

- The for...in statements combo iterates (loops) over the properties of an object.
- The code block inside the loop is executed once for each property.

Syntax

```
for (x in object)
{
  code block to be executed
}
```

Parameters

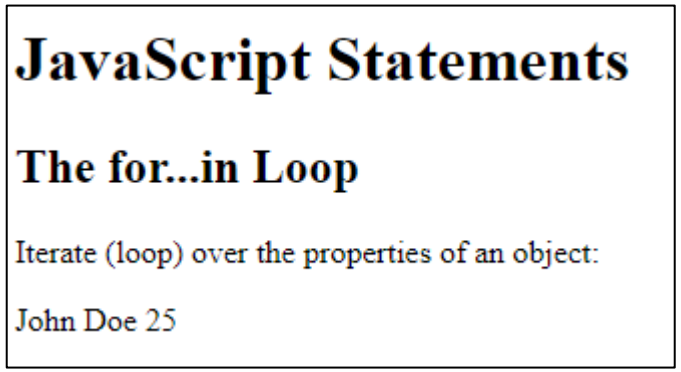
Parameter	Description
x	Required. A variable to iterate over the properties.
object	Required. The object to be iterated

Examples

Iterate (loop) over the properties of an object:

```
<html>
<body>
<h1>JavaScript Statements</h1>
<h2>The for...in Loop</h2>
<p>Iterate (loop) over the properties of an object:</p>
<p id="demo"></p>
<script>
const person = {fname:"John", lname:"Doe", age:25};
let text = "";
for (let x in person) {
  text += person[x] + " ";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

OUTPUT



Example Explained

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

For In Over Arrays

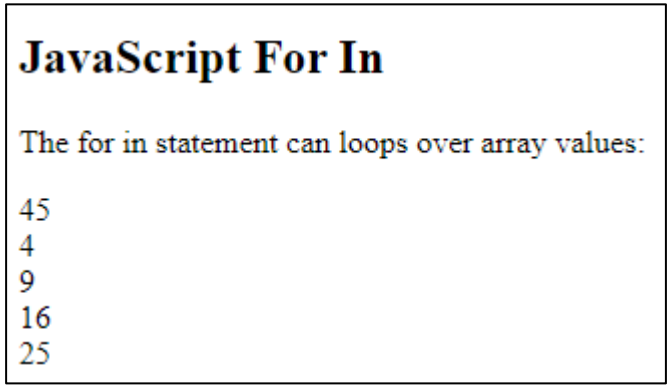
The JavaScript for in statement can also loop over the properties of an Array:

Syntax

```
for (variable in array)
{
  code
}
```

Example

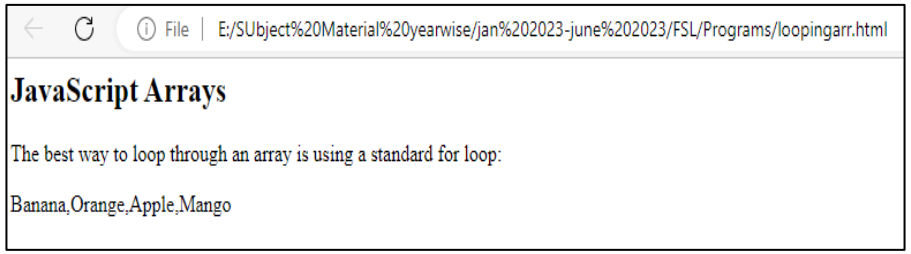
```
<h2>JavaScript For In</h2>
<p>The for in statement can loops over array values:</p>
<p id="demo"></p>
<script>
const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
  txt += numbers[x] + "<br>";
}
document.getElementById("demo").innerHTML = txt;
</script>
```



Looping Array Elements

Example

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The best way to loop through an array is using a standard for loop:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

for (let i = 0; i < fLen; i++)
{
  fruits[i];
}

document.getElementById("demo").innerHTML = fruits;
</script>
</body>
</html>
```

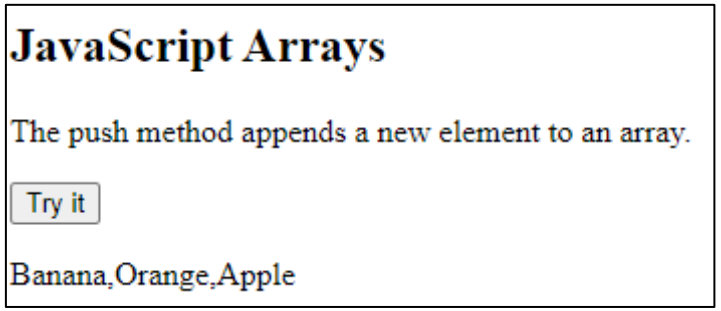


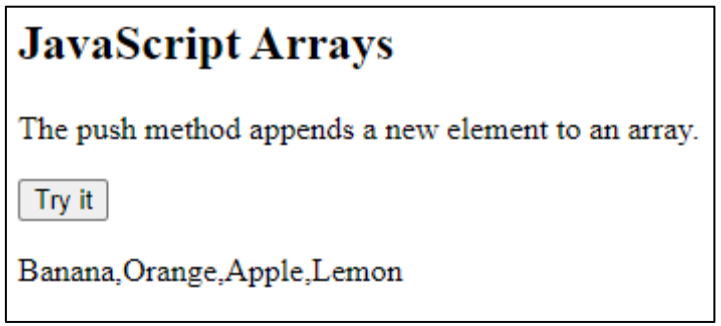
Adding Array Elements

To add a new element to an array is using the push() method:

Example

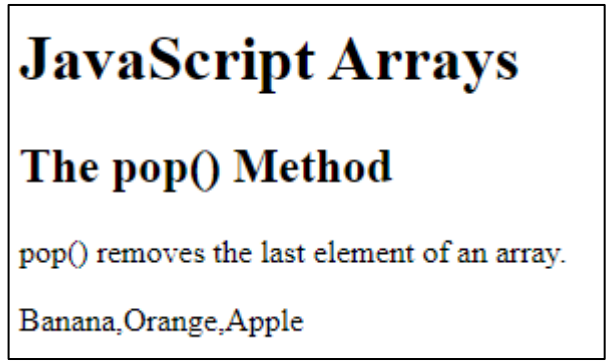
```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The push method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>
</body>
</html>
```





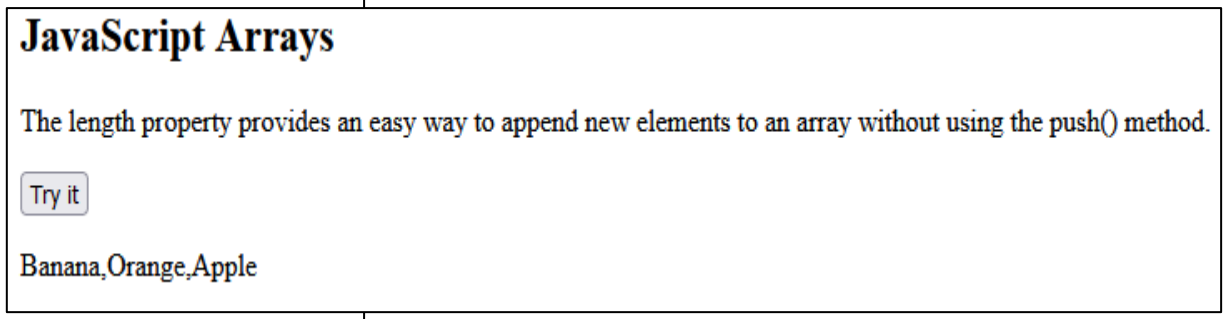
Removing an Array Elements

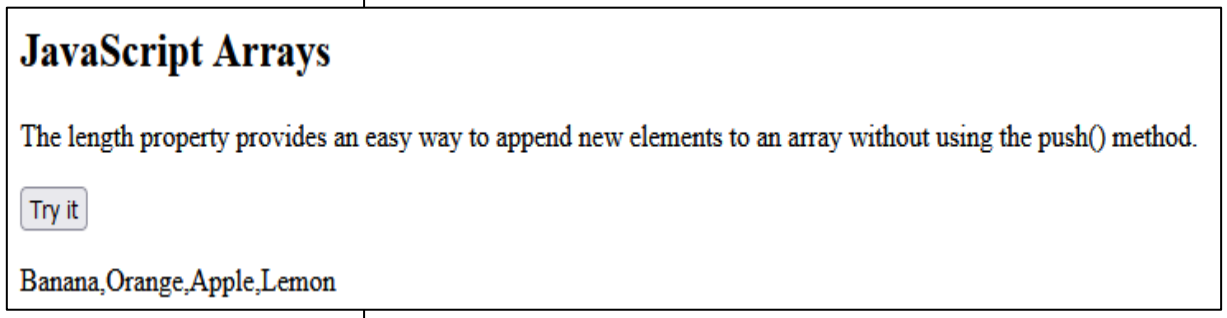
```
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The pop() Method</h2>

<p>pop() removes the last element of an array.</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
document.getElementById("demo").innerHTML = fruits;
</script>

</body>
</html>
```



New element can also be added to an array using the length property:

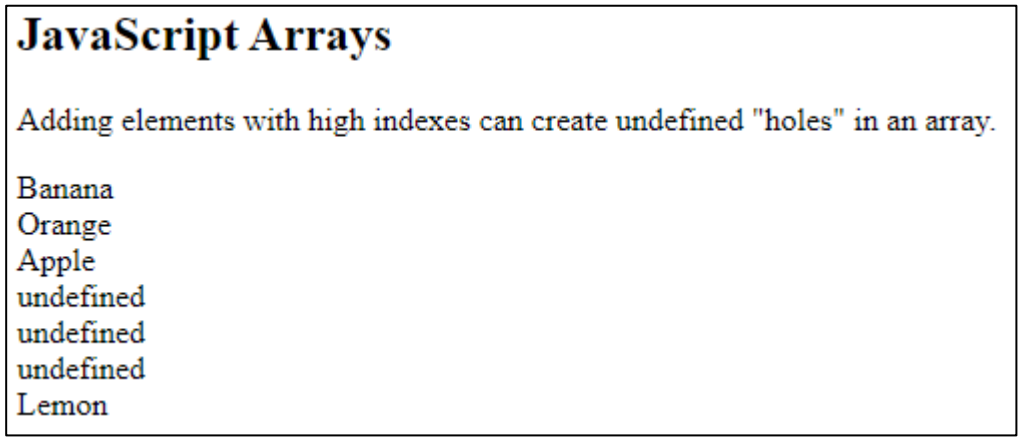
```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>The length property provides an easy way to append new elements to an
array without using the push() method.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits[fruits.length] = "Lemon";
  document.getElementById("demo").innerHTML = fruits;
}
</script>
</body>
</html>
```





Note:Adding elements with high indexes can create undefined "holes" in an array:

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>Adding elements with high indexes can create undefined "holes" in an array.</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";
let fLen = fruits.length;
let text = "";
for (i = 0; i < fLen; i++) {
  text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

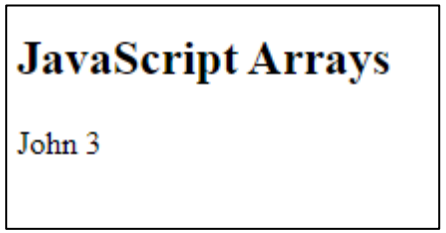


Associative Arrays

- 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

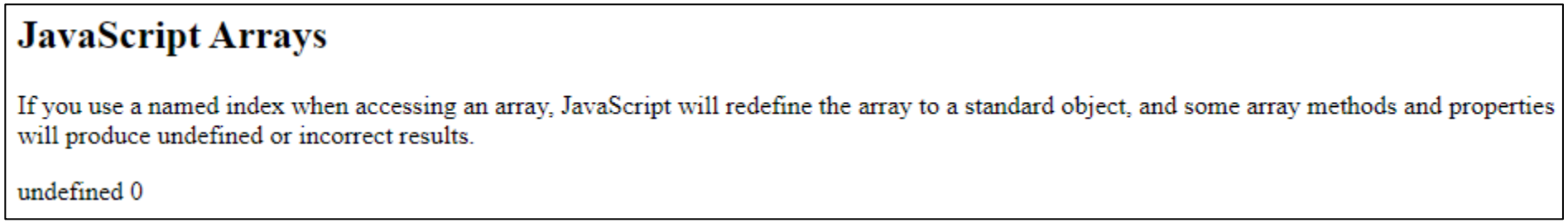
```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
</body>
</html>
```



WARNING !!

- **If you use named indexes, JavaScript will redefine the array to an object.**
- **After that, some array methods and properties will produce incorrect results.**

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some array methods and properties will produce undefined or incorrect results.</p>
<p id="demo"></p>
<script>
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
</body>
</html>
```

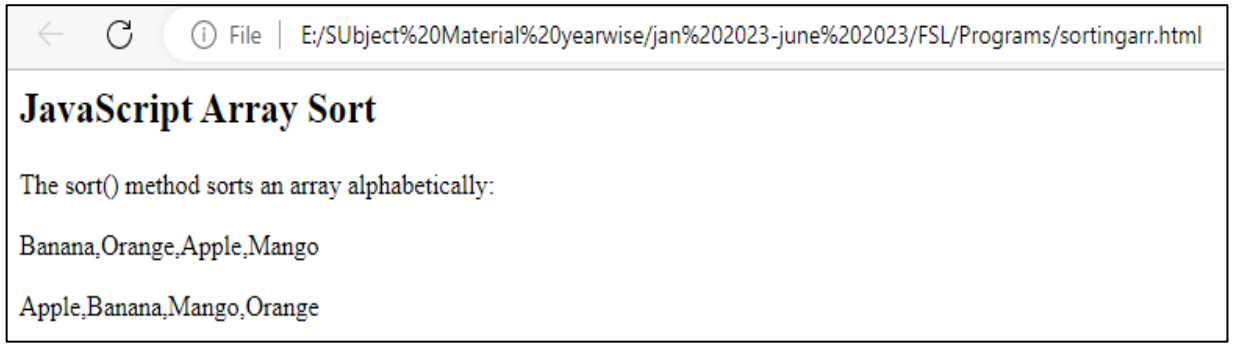


JavaScript Sorting Arrays

The sort() method sorts an array alphabetically:

Example

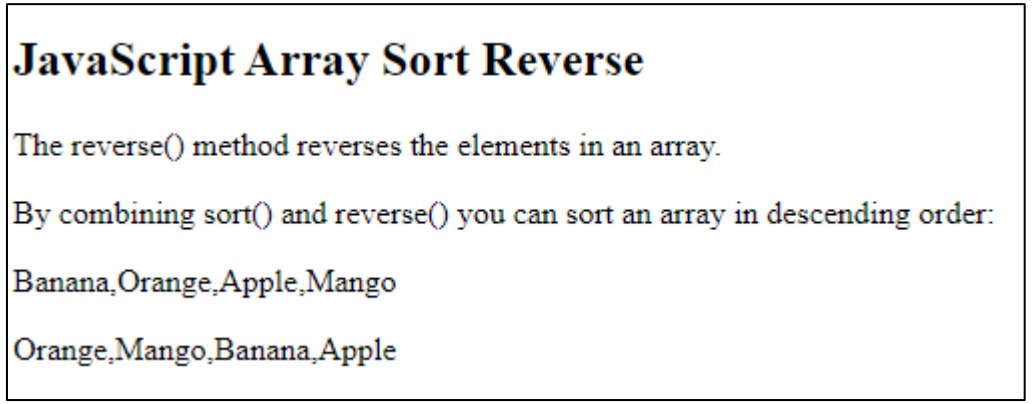
```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>The sort() method sorts an array alphabetically:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.sort();
document.getElementById("demo2").innerHTML = fruits;
</script>
</body>
</html>
```



Reversing an Array

- The reverse() method reverses the elements in an array.
- You can use it to sort an array in descending order:

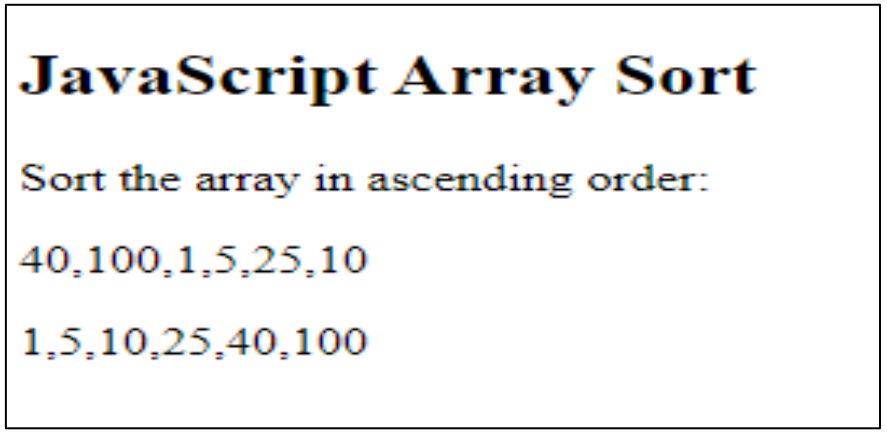
```
<html>
<body>
<h2>JavaScript Array Sort Reverse</h2>
<p>The reverse() method reverses the elements in an array.</p>
<p>By combining sort() and reverse() you can sort an array in descending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
// Create and display an array:
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
// Then reverse it:
fruits.reverse();
document.getElementById("demo2").innerHTML = fruits;
</script>
</body>
</html>
```



Numeric Sort

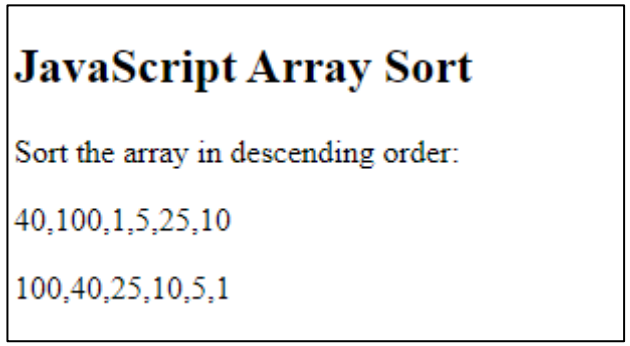
- By default, the sort() function sorts values as strings.
- This works well for strings ("Apple" comes before "Banana").
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".
- Because of this, the sort() method will produce incorrect result when sorting numbers.
- You can fix this by providing a compare function:

```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>Sort the array in ascending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = points;
points.sort(function(a, b){return a - b});
document.getElementById("demo2").innerHTML = points;
</script>
</body>
</html>
```



Sort an array descending

```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>Sort the array in descending order:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = points;
points.sort(function(a, b){return b - a});
document.getElementById("demo2").innerHTML = points;
</script>
</body>
</html>
```



The Compare Function

- The purpose of the compare function is to define an alternative sort order.
- The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a - b}
```

- When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

- If the result is negative, a is sorted before b.

- If the result is positive, b is sorted before a.

- If the result is 0, no changes are done with the sort order of the two values.

Example:

- The compare function compares all the values in the array, two values at a time (a, b).

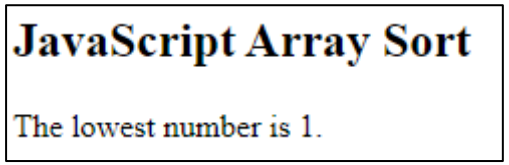
- When comparing 40 and 100, the sort() method calls the compare function(40, 100).

- The function calculates 40 - 100 (a - b), and since the result is negative (-60), the sort function will sort 40 as a value lower than 100.

Find the Highest (or Lowest) Array Value

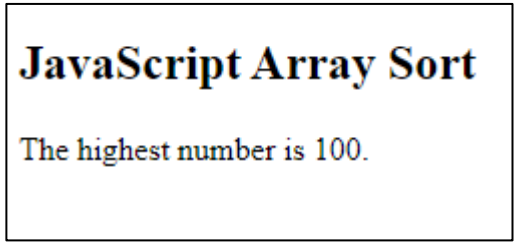
Lowest

```
<html>
<body>
<h2>JavaScript Array Sort</h2>
<p>The lowest number is <span id="demo"></span>.</p>
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
document.getElementById("demo").innerHTML = points[0];
</script>
</body>
</html>
```



Highest

```
<script>
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b-a});
document.getElementById("demo").innerHTML = points[0];
</script>
```



Combining an Array element into a String

JavaScript Array join()

- The join() method returns an array as a string.
- The join() method does not change the original array.
- Any separator can be specified. The default is comma (,).

Syntax

```
array.join(separator)
```

Parameters

Parameter	Description
separator	Optional. The separator to be used. Default is a comma.

Return Value

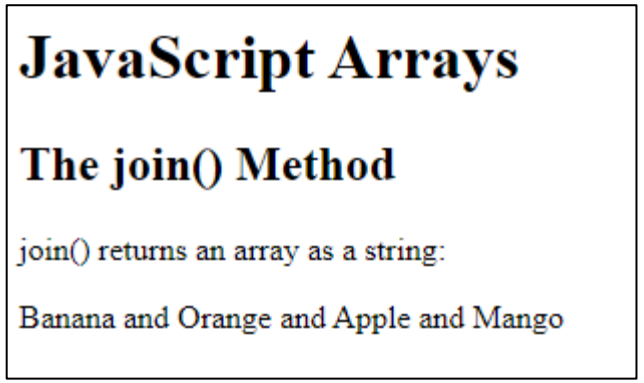
Type	Description
A string.	The array values, separated by the specified separator.

Example:----join

```
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The join() Method</h2>

<p>join() returns an array as a string:</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let text = fruits.join();

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

</body>
</html>
```

Example:----join

```
<body>
<h1>JavaScript Arrays</h1>
<h2>The join() Method</h2>
<p>join() returns an array as a string:</p>
<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let text = fruits.join(" and ");
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```



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).

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 parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

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:

Calculate the product of two numbers, and return the result:

```
<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>
```

Why Functions?

- **Code reusability**: We can call a function several times so it save coding.
- **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function to convert from Fahrenheit to
Celsius:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML =
toCelsius(77);
</script>
</body>
</html>
```

JavaScript Functions
This example calls a function to convert from Fahrenheit to Celsius:

25

The () Operator Invokes the Function

Accessing a function without () will return the function object instead of the function result.

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>Accessing a function without () will return the function
definition instead of the function result:</p>
<p id="demo"></p>
<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius;
</script>
</body>
</html>
```

JavaScript Functions
Accessing a function without () will return the function definition instead of the function result:

function toCelsius(f) { return (5/9) * (f-32); }

Local Variables

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Function Example

```
<html>
<body>
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
</body>
</html>
```

JavaScript Function Arguments

```
<script>
function getcube(number){
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
```

Function with Return Value

- The return statement is used to return a particular value from the function to the function caller. The function will stop executing when the return statement is called. The return statement should be the last statement in a function because the code after the return statement will be unreachable.

- We can return primitive values (such as Boolean, number, string, etc.) and Object types (such as functions, objects, arrays, etc.) by using the return statement.

- We can also return multiple values using the **return** statement. It cannot be done directly. We have to use an **Array** or **Object** to return multiple values from a function.

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
```

Example1
This is a simple example of using the return statement. Here, we are returning the result of the product of two numbers and returned back the value to the function caller.

The variable res is the function caller; it is calling the function fun() and passing two integers as the arguments of the function. The result will be stored in the res variable. In the output, the value 360 is the product of arguments 12 and 30.

```
<html>
  <head>
</head>
  <body>
<h3> Example of the JavaScript's return statement </h3>
    <script>
var res = fun(12, 30);
function fun(x, y)
{
return x * y;
}
document.write(res);
    </script>
</body>
 </html>
```

Example of the JavaScript's return statement
360

Example 2

```
<html>
<body>
<p>This example calls a function which interrupts the
loop using return statement</p>
<p id="demo"></p>
<script>
function func(){
for(var i = 0; i <= 10; i++){
if(i === 5)
return i;
}
}
document.getElementById("demo").innerHTML = func();
</script>
</body>
</html>
```

This example calls a function which interrupts the loop using return statement

5

Note:

In the above program, we can see we have a function defined with the name “func” where we have created a for loop to print the value where the return statement will return the value, but in the function, it will stop at 5, and therefore the return statement will stop the for a loop at 5 though it can traverse up to 10.

Example 3

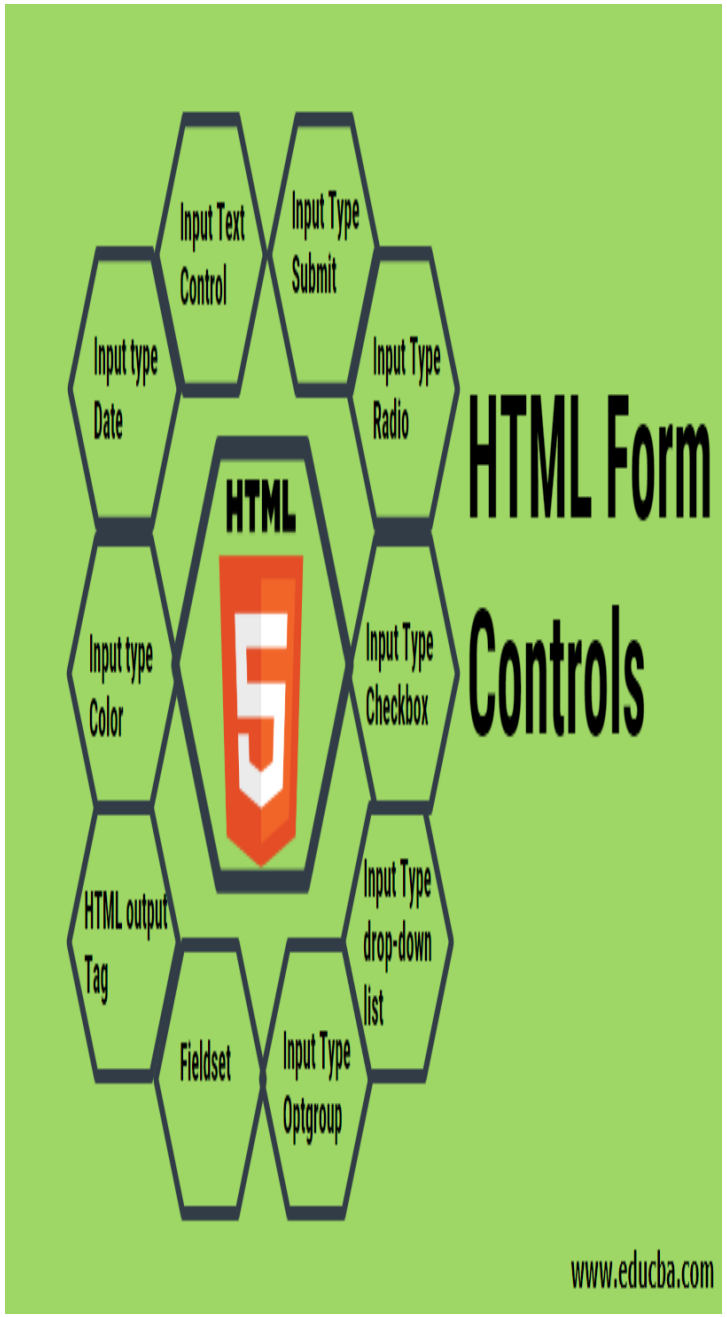
- Here, we are interrupting a function using the **return** statement. The function stops executing immediately when the **return** statement is called.
- There is an infinite **while** loop and variable **i,** which is initialized to 1. The loop continues until the value of **i** reached to **4**. When the variable's value will be 4, the loop stops its execution because of the **return** statement. The statement after the loop will never get executed.
- Here, the **return** statement is without using the ***expression***, so it returns **undefined.**

```
<script>
   var x = fun();
function fun() {
var i = 1;
 while(i) {
   document.write(i + '<br>');
     if (i == 4) {
       return;
     }
     document.write(i + '<br>');
     i++;
   }
 document.write('Hello world');
}
</script>
```

OUTPUT:

1
1
2
2
3
3
4

THANK YOU



Experiment 6-HTML Form

COURSE OUTCOMES:

CO1 : Build interactive web pages using program flow control structure.

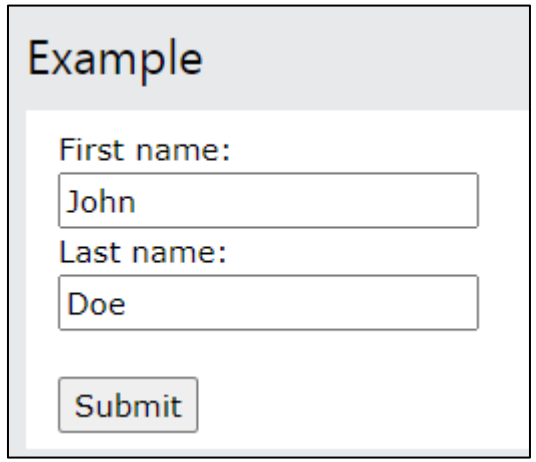
CO2 : Implement Arrays , functions and create event based web forms using Java Script.

CO3 : Use JavaScript for browser data persistence.

CO4 : Create menus, navigation in interactive webpages using regular expressions for validations.

HTML Forms:

An HTML form is used to collect user input. The user input is most often sent to a server for processing.



The <form> Element

The HTML <form> element is used to create an HTML form for user input:

<form>
.
form elements
.
</form>

The <form> element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.

The <input> Element:

The HTML <input> element is the most used form element.

An <input> element can be displayed in many ways, depending on the type attribute.

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

Label tag:

The <label> tag defines a label for several elements:

Attributes

Attribute	Value	Description
for	*element_id*	Specifies the id of the form element the label should be bound to
form	*form_id*	Specifies which form the label belongs to

Text Fields

The <input type="text"> defines a single-line input field for text input.

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

Note: The default width of an input field is 20 characters.

HTML Name attribute:

- The name attribute specifies the name of an <input> element.
- The name attribute is used to reference elements in a JavaScript, or to reference form data after a form is submitted.

HTML id Attribute:

- The id attribute specifies a unique id for an HTML element (the value must be unique within the HTML document).
- The id attribute is most used to point to a style in a style sheet, and by JavaScript (via the HTML DOM) to manipulate the element with the specific id.

The id Attribute

- The id attribute is a unique identifier of the HTML element. Each id attribute must be unique. Also, this attribute must begin with a letter and is case sensitive.
- In CSS, the id attribute is referenced with the # character. In Javascript, it is referenced with getElementById().

The name Attribute

- The name attribute defines a name of the element. It is used in the HTTP request that is sent to the server as a variable name by the browser.
- This attribute is associated with the data within the element.
- Like the id attribute, the name attribute must begin with a letter and is case sensitive, but unlike the id attribute, it can be not unique.
- The name attribute cannot be referenced in CSS. In Javascript, it is referenced with getElementsByName().

The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

HTML <input> value Attribute

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname" value="John"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Definition and Usage

The value attribute specifies the value of an <input> element.

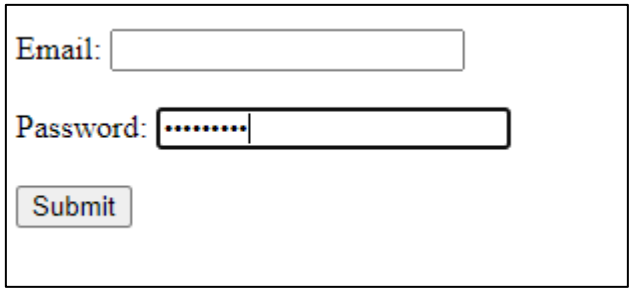
The value attribute is used differently for different input types:

- For "button", "reset", and "submit" - it defines the text on the button
- For "text", "password", and "hidden" - it defines the initial (default) value of the input field
- For "checkbox", "radio", "image" - it defines the value associated with the input (this is also the value that is sent on submit)

HTML <input type="password">

The <input type="password"> defines a password field (characters are masked).

```
<label for="pwd">Password:</label>
<input type="password" id="pwd" name="pwd">
```



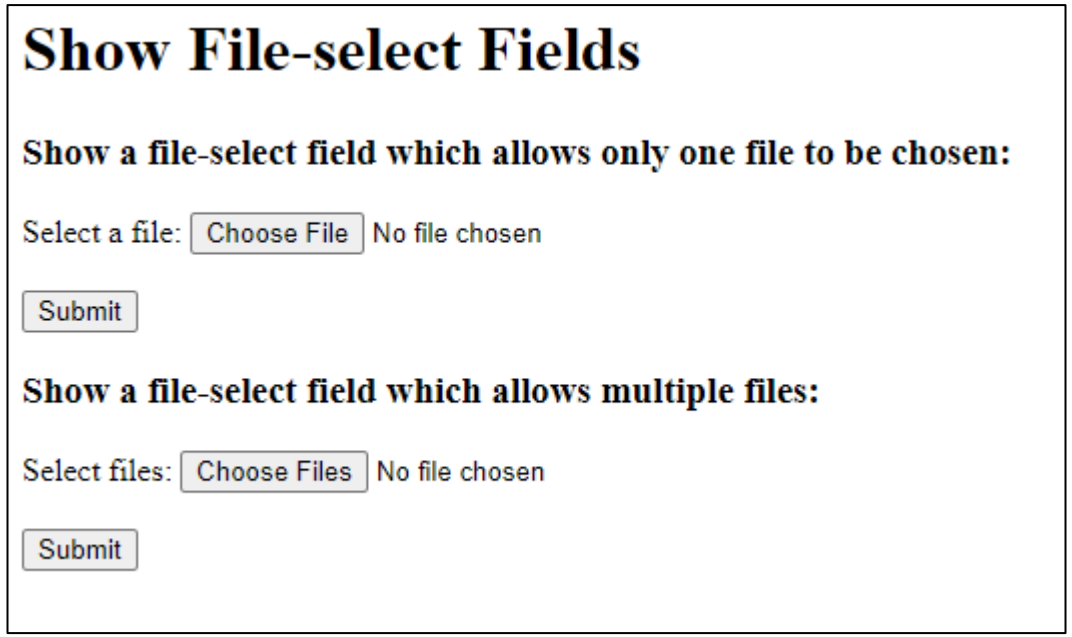
HTML <input type="file">

- The <input type="file"> defines a file-select field and a "Browse" button for file uploads.
- To define a file-select field that allows multiple files to be selected, add the multiple attribute.

```
<label for="myfile">Select a file:</label>
<input type="file" id="myfile" name="myfile">
```

Multiple file

```
<h1>Show File-select Fields</h1>

<h3>Show a file-select field which allows only one file to be
chosen:</h3>
<form action="/action_page.php">
  <label for="myfile">Select a file:</label>
  <input type="file" id="myfile" name="myfile"><br><br>
  <input type="submit">
</form>

<h3>Show a file-select field which allows multiple files:</h3>
<form action="/action_page.php">
  <label for="myfile">Select files:</label>
  <input type="file" id="myfile" name="myfile" multiple><br><br>
  <input type="submit">
</form>
```



HTML <input type="date">

- The <input type="date"> defines a date picker.
- The resulting value includes the year, month, and day.

```
<label for="birthday">Birthday:</label>
<input type="date" id="birthday" name="birthday">
```

Radio Buttons:

The <input type="radio"> defines a radio button.

Radio buttons let a user select ONE of a limited number of choices.

Example

```
<p>Choose your favorite Web language:</p>
<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

OUTPUT

Choose your favorite Web language:

- ○ HTML
- ○ CSS
- ○ JavaScript

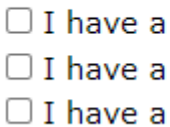
Checkboxes:

The <input type="checkbox"> defines a checkbox.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

This is how the HTML code above will be displayed in a browser:



The Submit Button:

- The <input type="submit"> defines a button for submitting the form data to a form-handler.
- The form-handler is typically a file on the server with a script for processing input data.
- The form-handler is specified in the form's action attribute.

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

This is how the HTML code above will be displayed in a browser:

First name:
John
Last name:
Doe

Submit

The <select> Element:

The <select> element defines a drop-down list:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

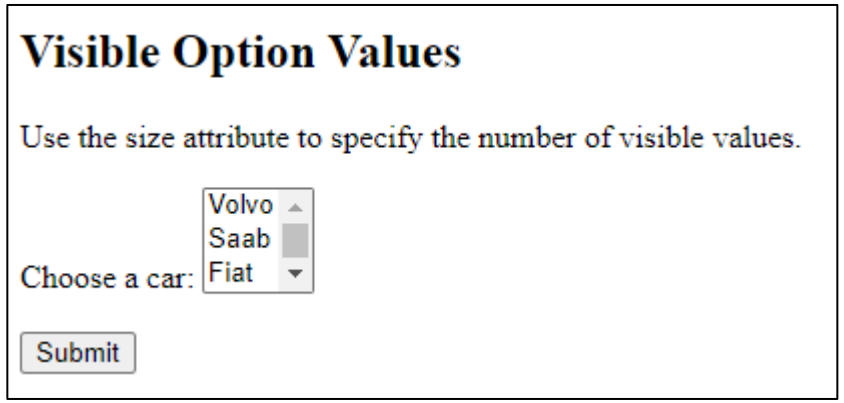
- The <option> elements defines an option that can be selected.
- By default, the first item in the drop-down list is selected.
- To define a pre-selected option, add the selected attribute to the option:

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the size attribute to specify the number of visible values:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```



Allow Multiple Selections:

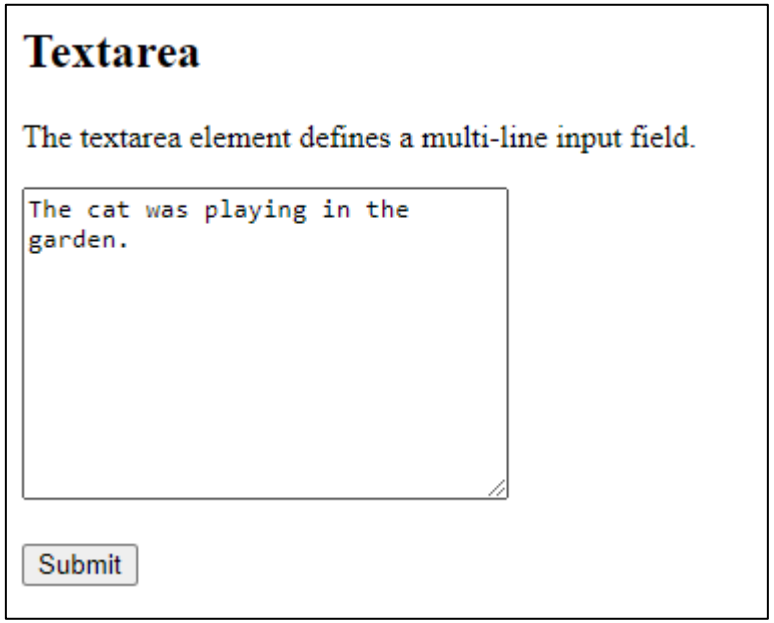
Use the multiple attribute to allow the user to select more than one value:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <textarea> Element

The <textarea> element defines a multi-line input field (a text area):

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```



- The rows attribute specifies the visible number of lines in a text area.
- The cols attribute specifies the visible width of a text area.

The <button> Element

The <button> element defines a clickable button:

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

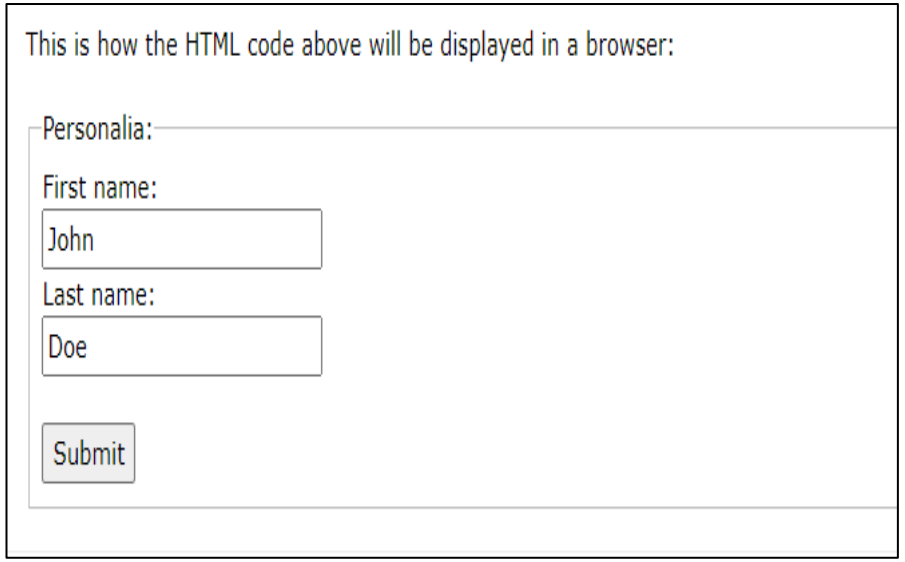
This is how the HTML code above will be displayed in a browser:

The <fieldset> and <legend> Elements

- The <fieldset> element is used to group related data in a form.
- The <legend> element defines a caption for the <fieldset> element.

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

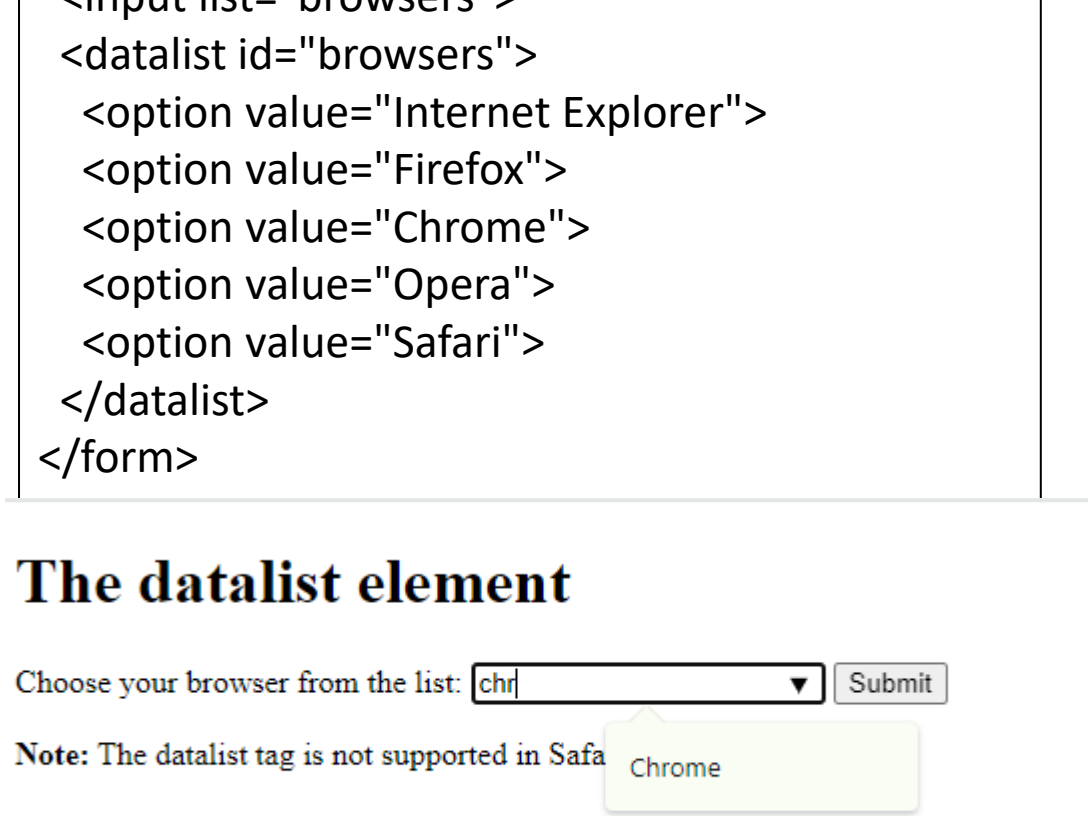
This is how the HTML code above will be displayed in a browser:

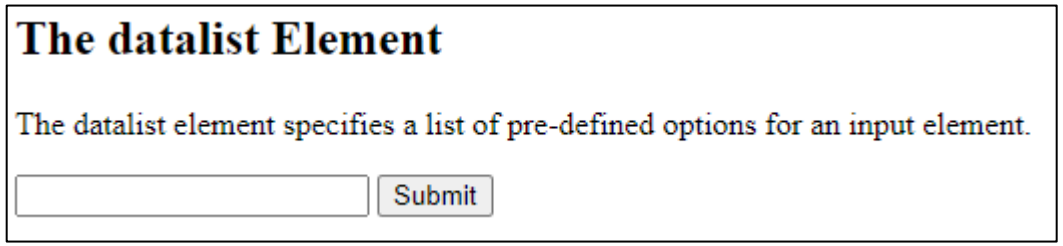


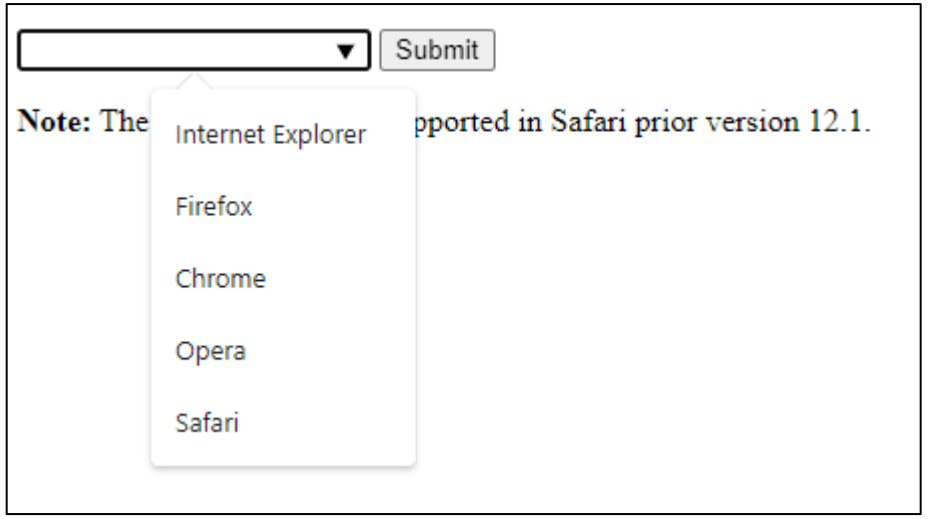
The <datalist> Element

- The <datalist> element specifies a list of pre-defined options for an <input> element.
- Users will see a drop-down list of the pre-defined options as they input data.
- The list attribute of the <input> element, must refer to the id attribute of the <datalist> element.

```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```





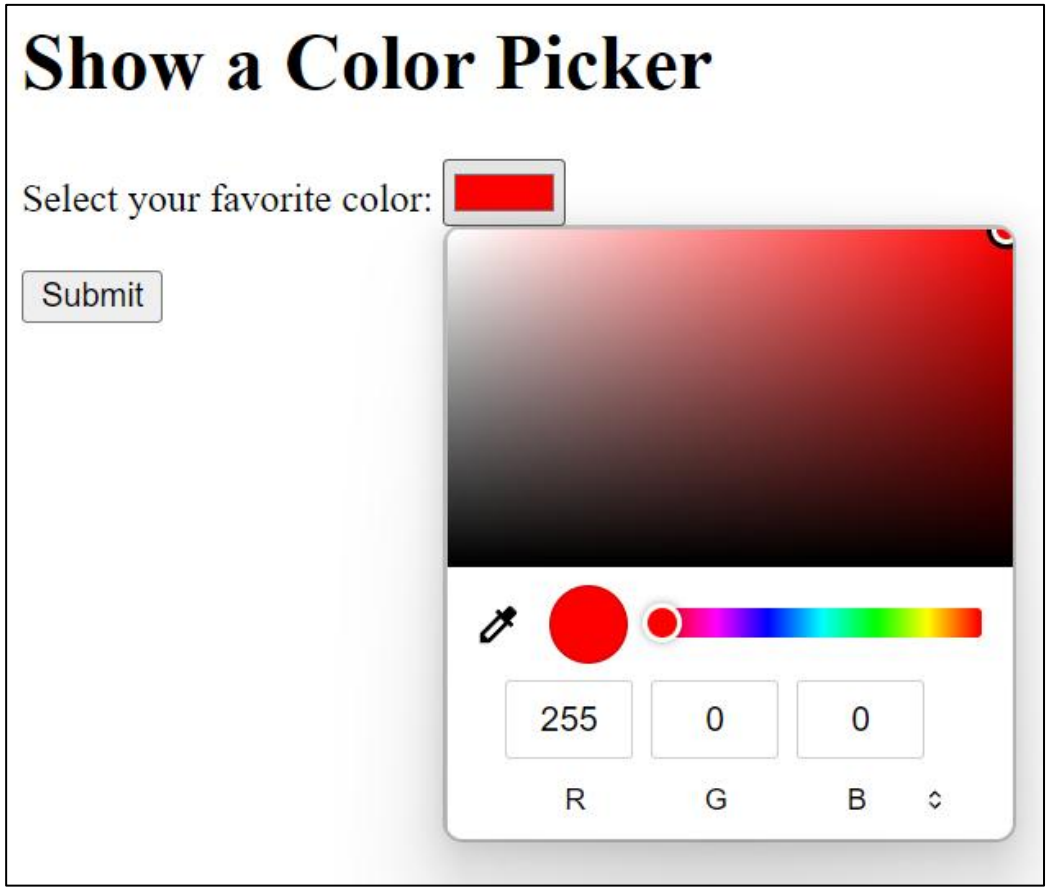


<input type="color">

The <input type="color"> is used for input fields that should contain a color.

```
<form>
  <label for="favcolor">Select your favorite color:</label>
  <input type="color" id="favcolor" name="favcolor">
</form>
```

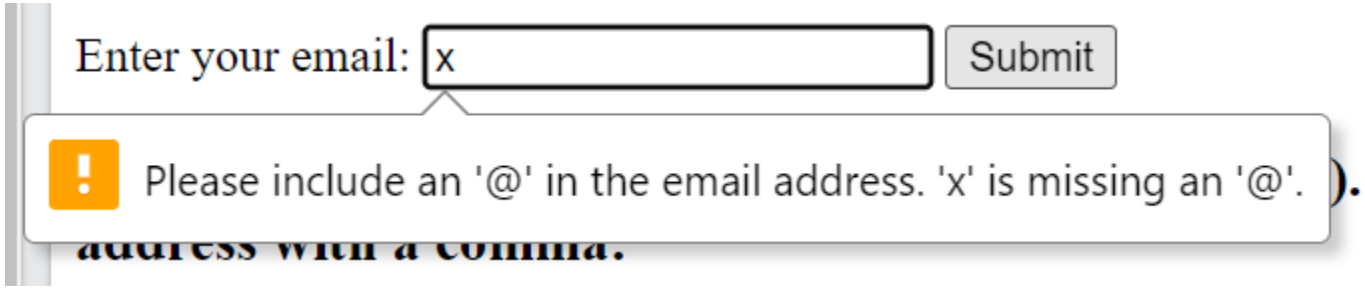
The default value is #000000 (black).
The value must be in seven-character hexadecimal notation.



<input type="email">

Define a field for an e-mail address (validates automatically when submitted):

```
<label for="email">Enter your email:</label>
<input type="email" id="email" name="email">
```



To define an e-mail field that allows multiple e-mail addresses, add the "multiple" attribute.

<input type="hidden">

The <input type="hidden"> defines a hidden input field.

A hidden field lets web developers include data that cannot be seen or modified by users when a form is submitted.

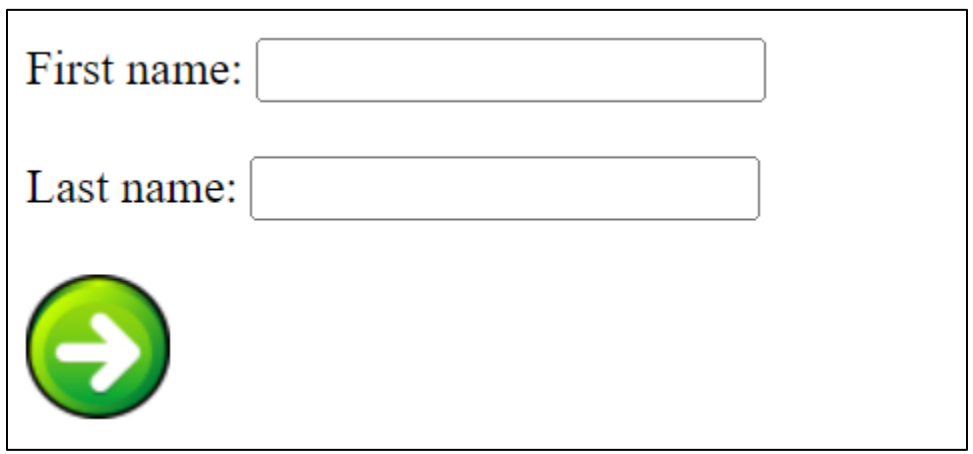
```
<form action="/action_page.php">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="hidden" id="custId" name="custId" value="3487">
  <input type="submit" value="Submit">
</form>
```

<input type="image">

- The <input type="image"> defines an image as a submit button.

- The path to the image is specified in the src attribute.

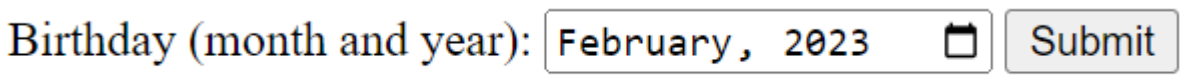
```
<form action="/action_page.php">
  <label for="fname">First name: </label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name: </label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="image" src="img_submit.gif" alt="Submit" width="48" height="48">
</form>
```



<input type="month">

- The <input type="month"> defines a month and year control.
- The format is "YYYY-MM".

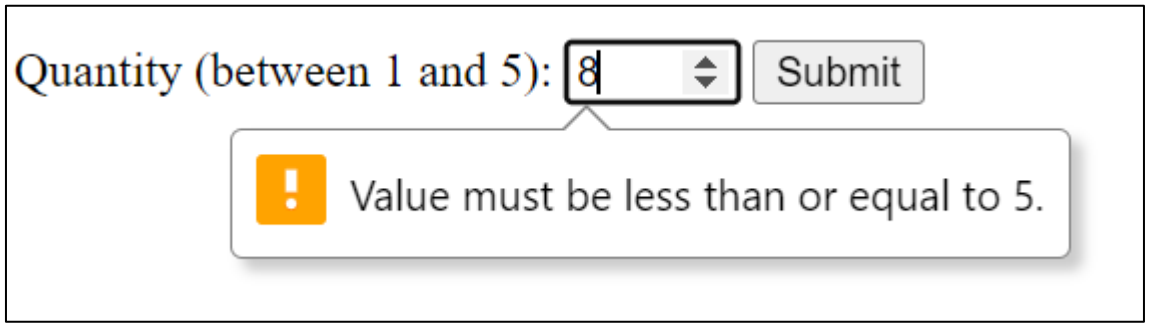
```
<form action="/action_page.php">
  <label for="bdaymonth">Birthday (month and year):</label>
  <input type="month" id="bdaymonth" name="bdaymonth">
  <input type="submit">
</form>
```



<input type="number">

The <input type="number"> defines a field for entering a number.

```
<label for="quantity">Quantity (between 1 and 5):</label>
<input type="number" id="quantity" name="quantity" min="1" max="5">
```



Use the following attributes to specify restrictions:

- max - specifies the maximum value allowed
- min - specifies the minimum value allowed
- step - specifies the legal number intervals

```
<form action="/action_page.php">
  <label for="points">Points:</label>
  <input type="number" id="points" name="points" step="3">
  <input type="submit">
</form>
```

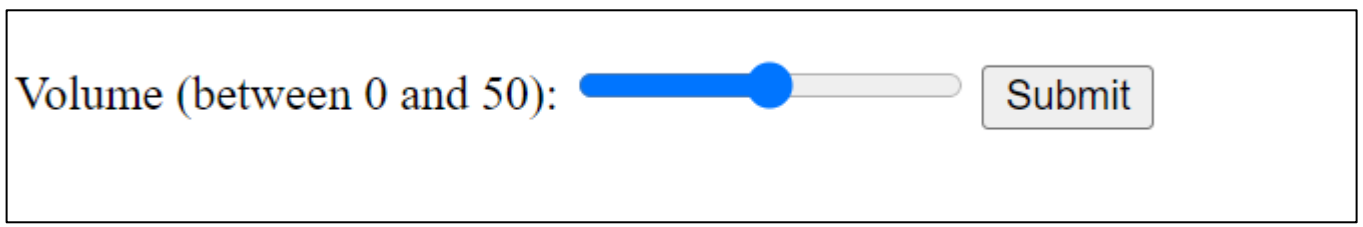
The step attribute specifies the interval between legal numbers in an <input> element.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

<input type="range">

The <input type="range"> defines a control for entering a number whose exact value is not important (like a slider control).

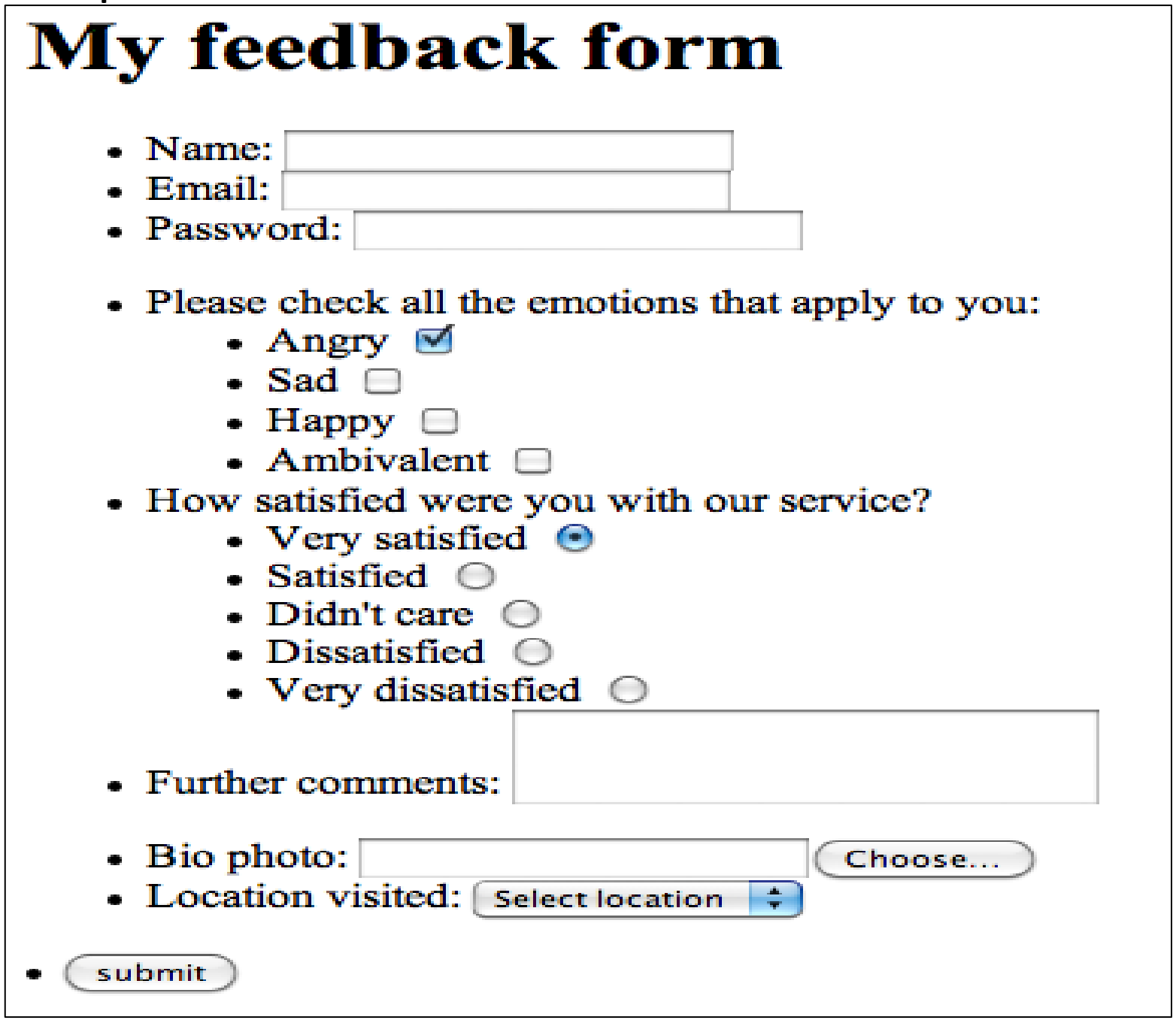
```
<form action="/action_page.php">
  <label for="vol">Volume (between 0 and 50):</label>
  <input type="range" id="vol" name="vol" min="0" max="50">
  <input type="submit">
</form>
```



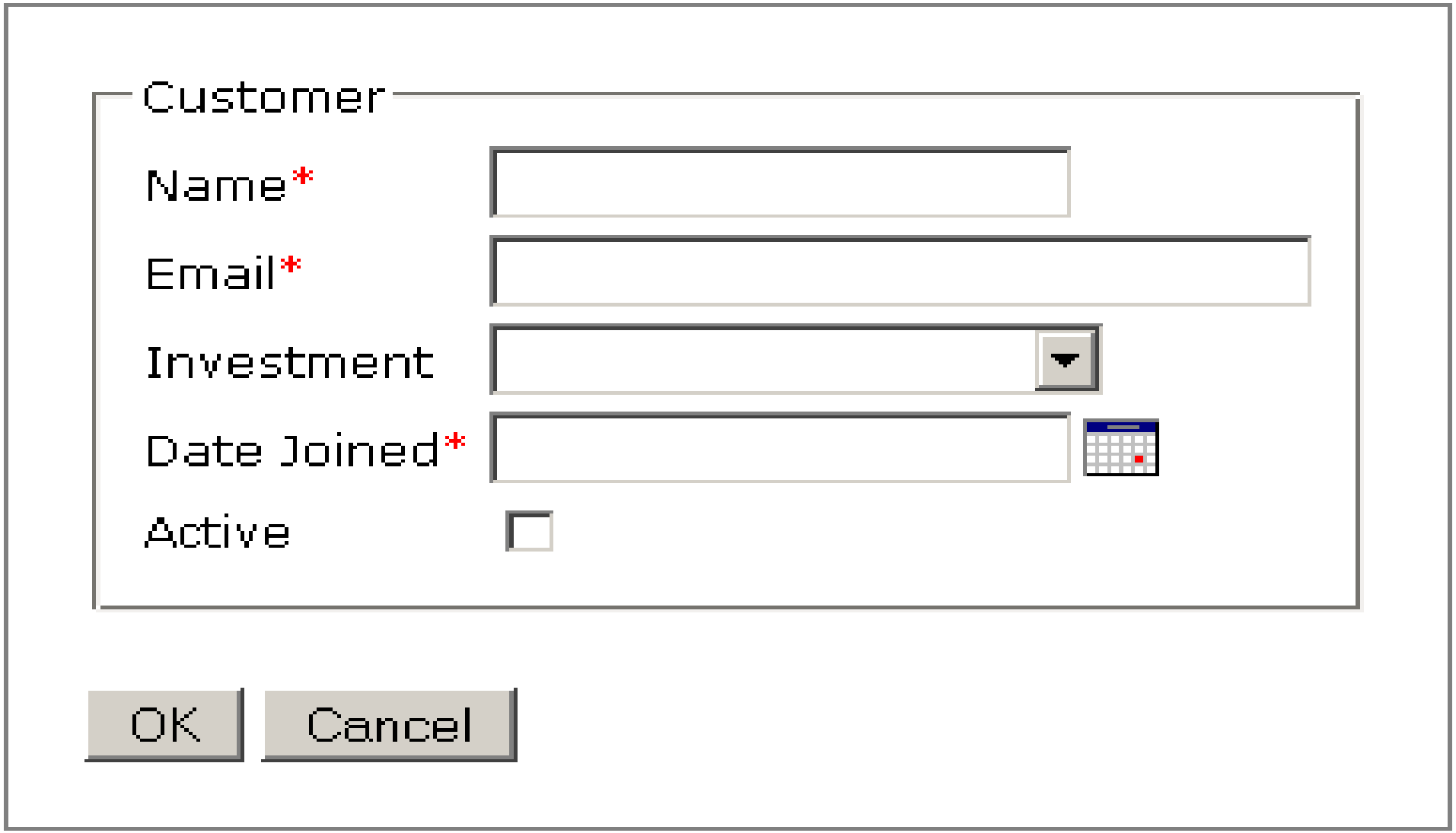
<input type="tel">

```
<form action="/action_page.php">
  <label for="phone">Enter a phone number:</label><br><br>

  <input type="tel" id="phone" name="phone" placeholder="123-45-6789" pattern="[0-9]{3}-[0-9]{2}-[0-9]{4}"
required><br><br>

  <small>Format: 123-45-678</small><br><br>
  <input type="submit">
</form>
```

Practice Example 1:



Practice Example 2:



Practice Example 3:

Name	Value
Name	
Sex	○ Male ◉ Female
Eye color	green
Check all that apply	☐ Over 6 feet tall ☐ Over 200 pounds
Describe your athletic ability:	
Enter my information	


Thank
You!

Form Events

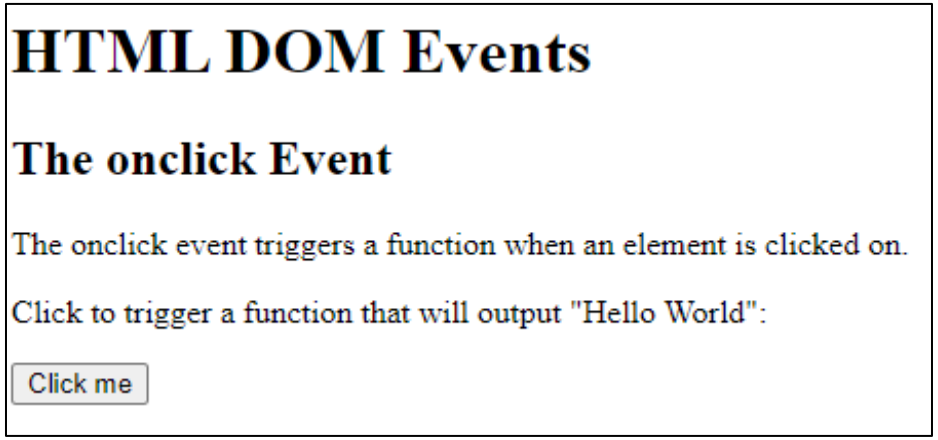
The MouseEvent Object:

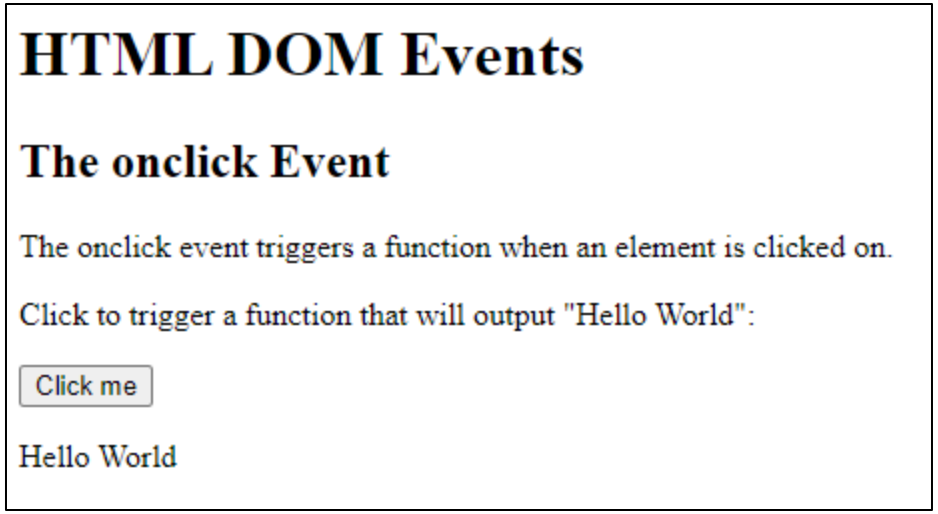
The MouseEvent Object handles events that occur when the mouse interacts with the HTML document.

Event	Occurs When
onclick	A user clicks on an element
ondblclick	A user double-clicks on an element
onmousedown	A mouse button is pressed over an element
onmouseenter	The mouse pointer moves into an element
onmouseleave	The mouse pointer moves out of an element
onmousemove	The mouse pointer moves over an element
onmouseout	The mouse pointer moves out of an element
onmouseover	The mouse pointer moves onto an element
onmouseup	A mouse button is released over an element

onclick Event-[MouseEvent]

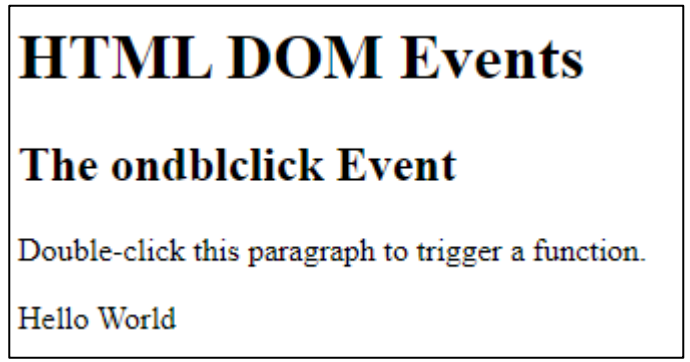
```
<html>
<body>
<h1>HTML DOM Events</h1>
<h2>The onclick Event</h2>

<p>The onclick event triggers a function when an element is clicked on.</p>
<p>Click to trigger a function that will output "Hello World":</p>

<button onclick="myFunction()">Click me</button>

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

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

</body>
</html>
```





Dblclick-MouseEvent

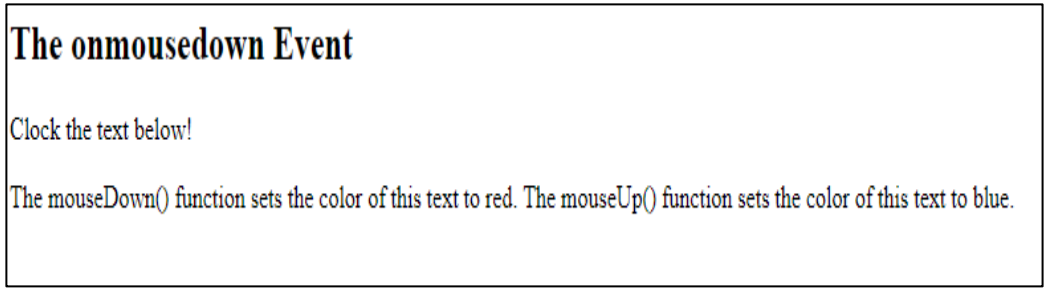
```
<html>
<body>
<h1>HTML DOM Events</h1>
<h2>The ondblclick Event</h2>

<p ondblclick="myFunction()">Double-click this paragraph to trigger
a function.</p>

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

<script>
function myFunction() {
  document.getElementById("demo").innerHTML += "Hello World ";
}
</script>

</body>
</html>
```

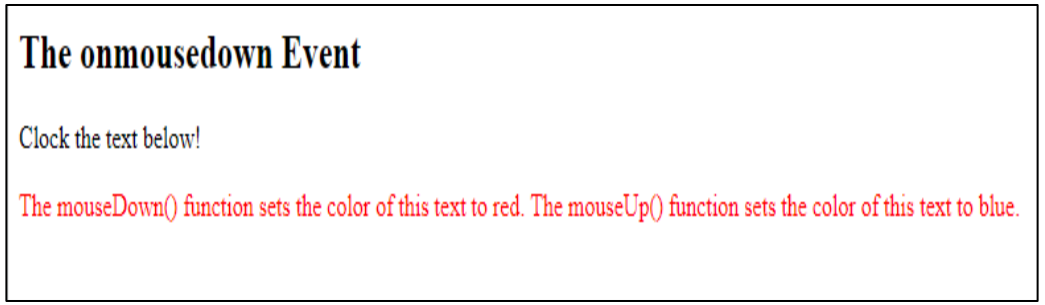


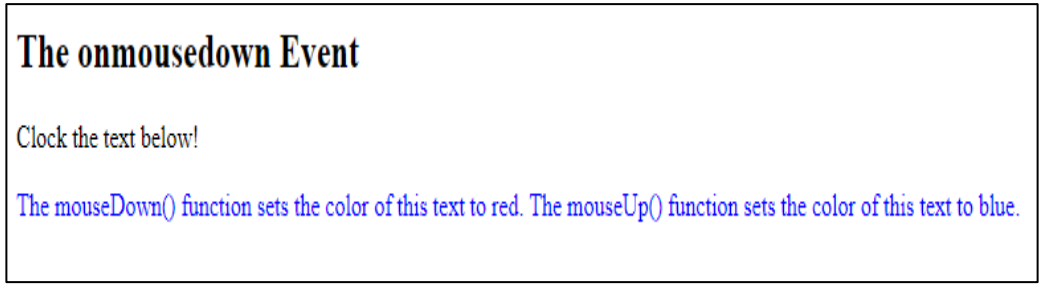
onmouseup -onmousedown Event

- The onmousedown event occurs when a user presses a mouse button over an HTML element.
- The onmouseup event occurs when a mouse button is released over an element.

```
<h2>The onmousedown Event</h2>

<p>Clock the text below!</p>
<p id="myP" onmousedown="mouseDown()“

 onmouseup="mouseUp()">

The mouseDown() function sets the color of this text to red.
The mouseUp() function sets the color of this text to blue.
</p>
<script>
function mouseDown() {
  document.getElementById("myP").style.color = "red";
}

function mouseUp() {
  document.getElementById("myP").style.color = "blue";
}
</script>
```





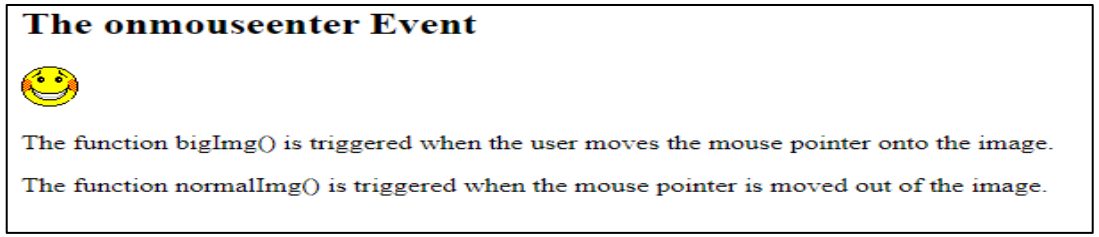


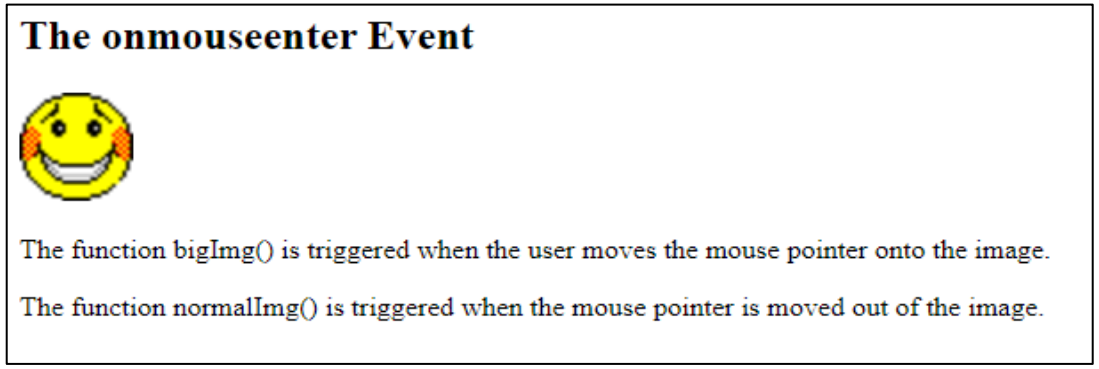
Onmouseenter- onmouseleave event

- The onmouseenter event occurs when the mouse pointer enters an element.

- The onmouseenter event is often used together with the onmouseleave event, which occurs when the mouse pointer leaves an element.

```
<html><body>
<h1>HTML DOM Events</h1>
<h2>The onmouseenter Event</h2>
<img onmouseenter="bigImg(this)" onmouseleave="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer onto the image.</p>
<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>
function bigImg(x) {
  x.style.height = "64px";
  x.style.width = "64px";
}

function normalImg(x) {
  x.style.height = "32px";
  x.style.width = "32px";
}
</script></body></html>
```

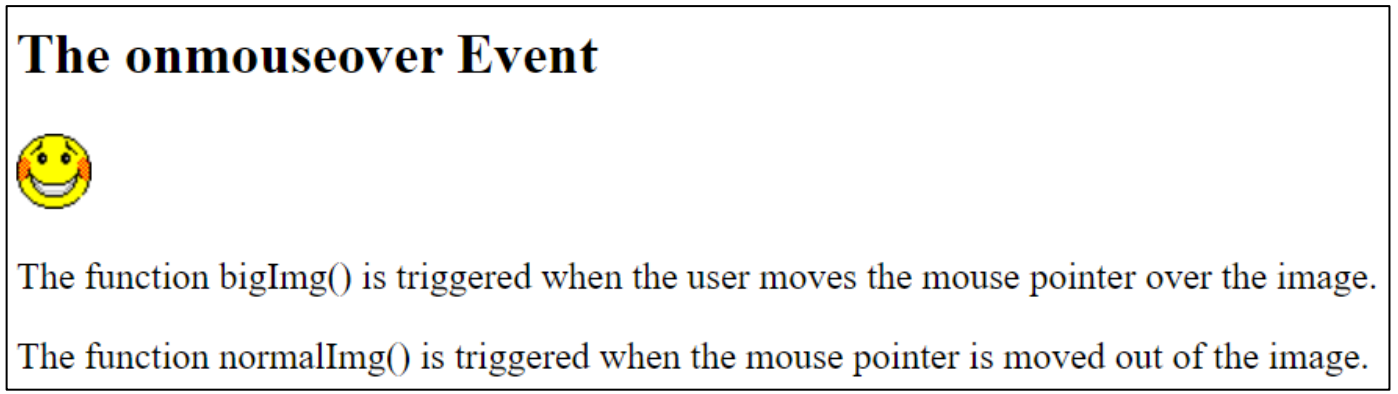


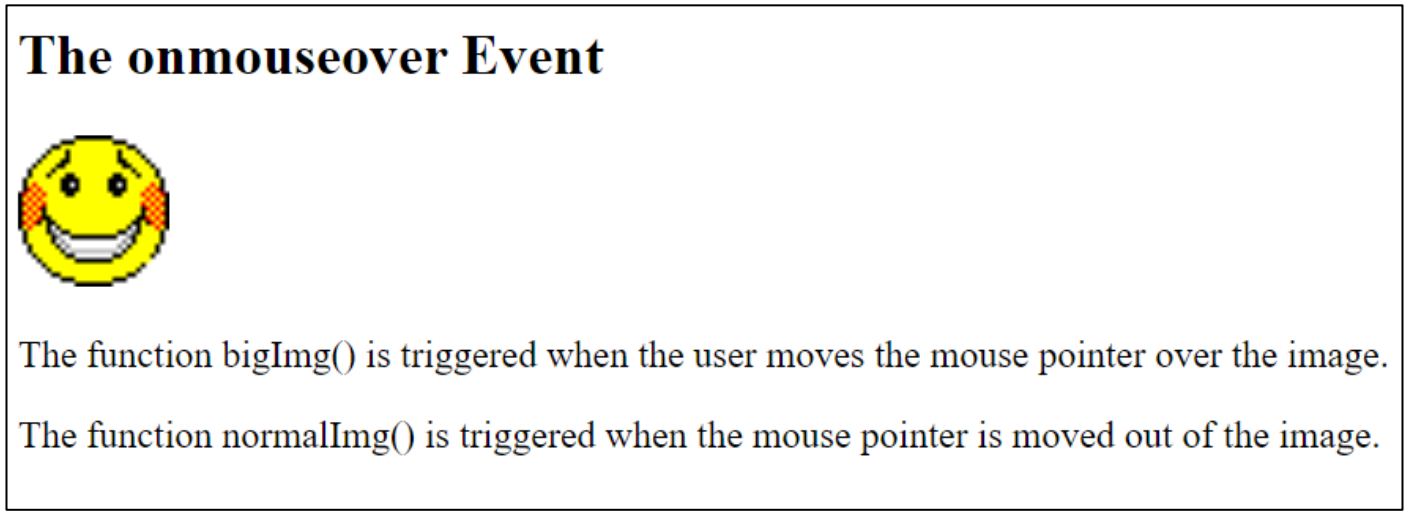


onmouseover -- onmouseout Event:

- The onmouseover event occurs when the mouse pointer enters an element.
- The onmouseover event is often used together with the onmouseout event, which occurs when the mouse pointer leaves the element.

```
<html><body>
<h2>The onmouseover Event</h2>
<img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="smiley.gif" alt="Smiley"
width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer over the image.</p>
<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>
function bigImg(x) {
  x.style.height = "64px";
  x.style.width = "64px";
}

function normalImg(x) {
  x.style.height = "32px";
  x.style.width = "32px";
}
</script></body>
</html>
```



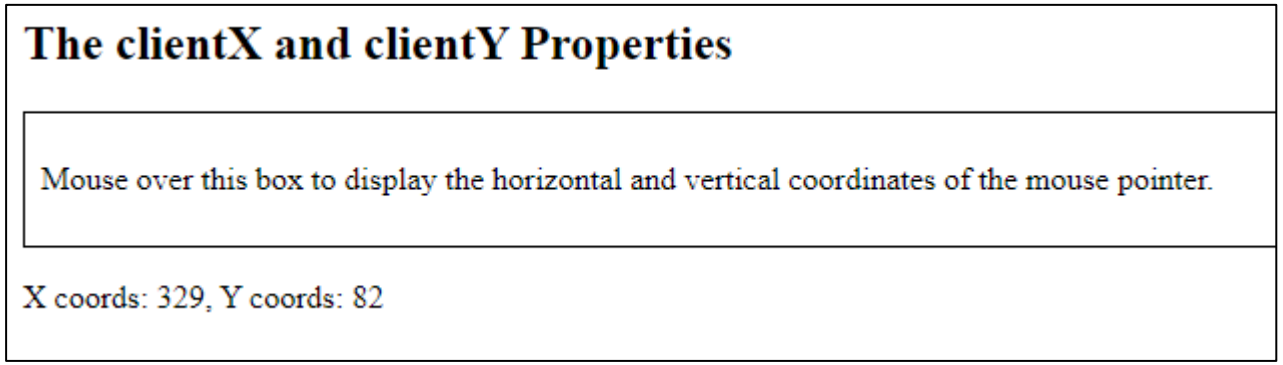


clientX and clientY Property

- The clientX property returns the horizontal client coordinate of the mouse pointer when a mouse event occurs.
- The clientX property is read-only.
- The client area is the current window.
- The clientY property returns the vertical client coordinate of the mouse pointer when a mouse event occurs.
- The clientY property is read-only.

Example:

```
<h2>The clientX and clientY Properties</h2>

<div style="border: 1px solid black;padding:8px" onmousemove="showCoords(event)" onmouseout="clearCoor()">

<p>Mouse over this box to display the horizontal and vertical coordinates of the mouse pointer.</p>
</div>

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

<script>
function showCoords(event) {
  let x = event.clientX;
  let y = event.clientY;
  let text = "X coords: " + x + ", Y coords: " + y;
  document.getElementById("demo").innerHTML = text;
}

function clearCoor() {
  document.getElementById("demo").innerHTML = "";
}
</script>
```



onmousemove Event

The onmousemove event occurs when the pointer moves over an element.

```
<html>
<style>
div {
  width: 200px;
  height: 100px;
  border: 1px solid black;
}
</style>

<body>
<h1>HTML DOM Events</h1>
<h2>The onmousemove Event</h2>

<div onmousemove="myFunction(event)" onmouseout="clearCoor()"></div>

<p>Mouse over the rectangle above, and get the coordinates of your mouse pointer.</p>

<p>When the mouse is moved over the div, the p element will display the horizontal and vertical coordinates of your mouse pointer, whose values are returned from the clientX and clientY properties on the
MouseEvent object.</p>
```

```
<p id="demo"></p>
<script>
function myFunction(e) {
  let x = e.clientX;
  let y = e.clientY;
  let coor = "Coordinates: (" + x + "," + y + ")";
  document.getElementById("demo").innerHTML = coor;
}

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

The onmousemove Event

Mouse over the rectangle above, and get the coordinates of your mouse pointer.

When the mouse is moved over the div, the p element will display the horizontal and vertical coordinates of your mouse pointer, whose values are returned from the clientX and clientY properties on the MouseEvent object.

Coordinates: (52,167)

The onmouseenter event is similar to the onmouseover event.

difference between onmousemove, onmouseenter and onmouseover

Example:

```
<html>
<style>
div {
  width: 150px;
  height: 100px;
  border: 1px solid black;
  margin: 10px;
  padding: 50px;
  text-align: center;
  background-color: lightgray;
}
</style>
<body>
<h1>HTML DOM Events</h1>
<h2>The onmouseenter Event</h2>
<p>This example demonstrates the difference between onmousemove, onmouseenter and onmouseover.</p>
```

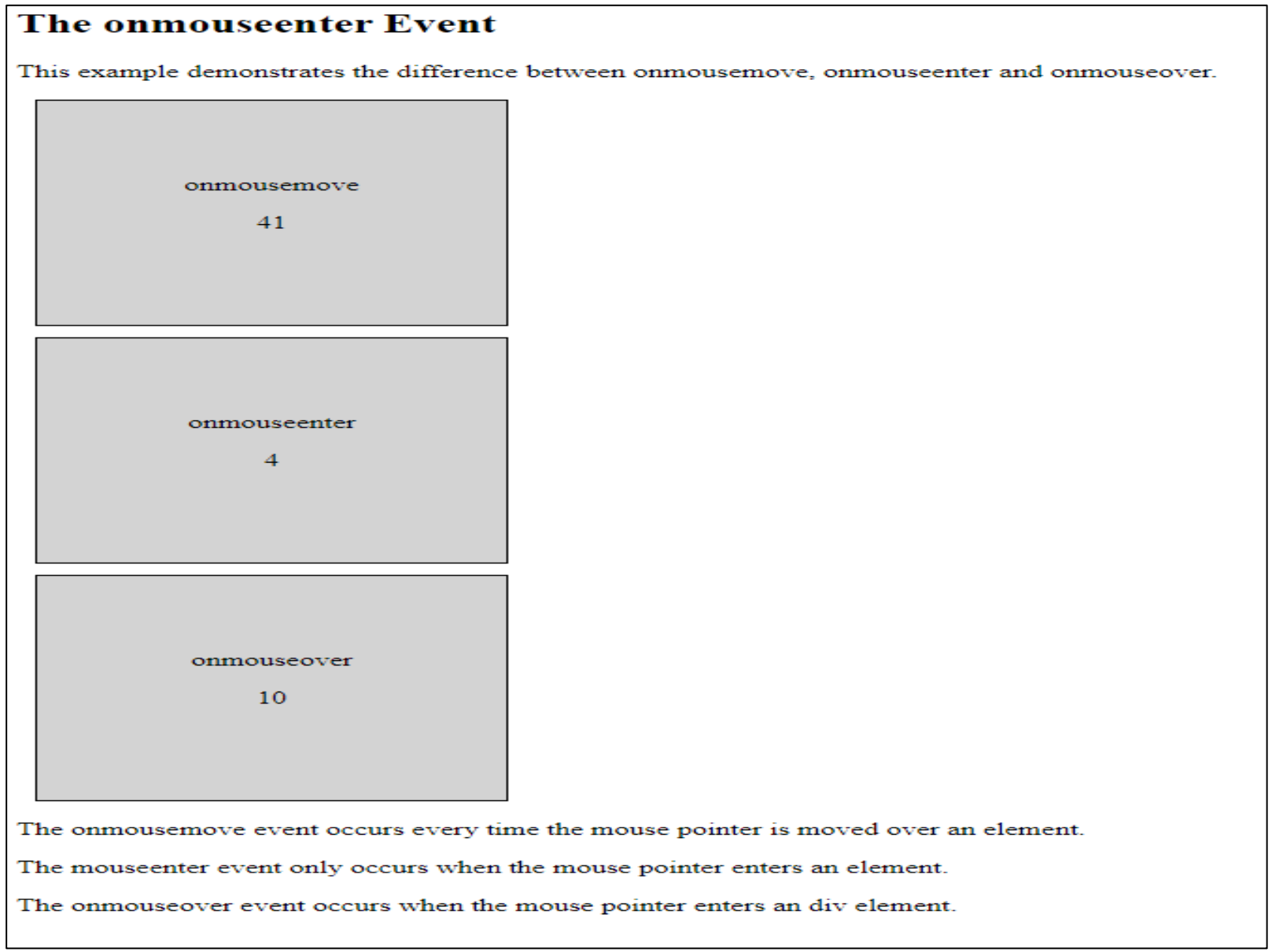
Example continued..

```
<div onmousemove="myMoveFunction()">
  <p>onmousemove</p>
  <p id="demo1">Mouse over me!</p>
</div>

<div onmouseenter="myEnterFunction()">
  <p>onmouseenter</p>
  <p id="demo2">Mouse over me!</p>
</div>

<div onmouseover="myOverFunction()">
  <p>onmouseover</p>
  <p id="demo3">Mouse over me!</p>
</div>

<p>The onmousemove event occurs every time the mouse pointer is moved over an element.</p>
<p>The mouseenter event only occurs when the mouse pointer enters an element. </p>
<p>The onmouseover event occurs when the mouse pointer enters an div element.</p>
```

Example continued..

```
<script>
let x = 0;
let y = 0;
let z = 0;

function myMoveFunction() {
  document.getElementById("demo1").innerHTML = z+=1;
}

function myEnterFunction() {
  document.getElementById("demo2").innerHTML = x+=1;
}

function myOverFunction() {
  document.getElementById("demo3").innerHTML = y+=1;
}
</script>

</body>
</html>
```

OUTPUT

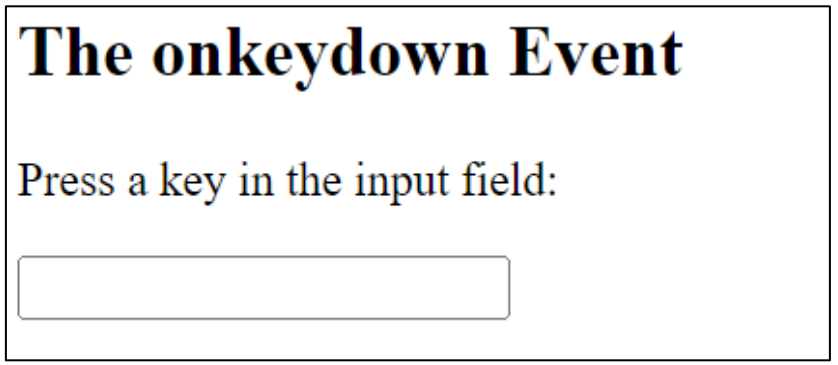

The onmouseenter Event
This example demonstrates the difference between onmousemove, onmouseenter and onmouseover.
onmousemove
41
onmouseenter
4
onmouseover
10
The onmousemove event occurs every time the mouse pointer is moved over an element.
The mouseenter event only occurs when the mouse pointer enters an element.
The onmouseover event occurs when the mouse pointer enters an div element.

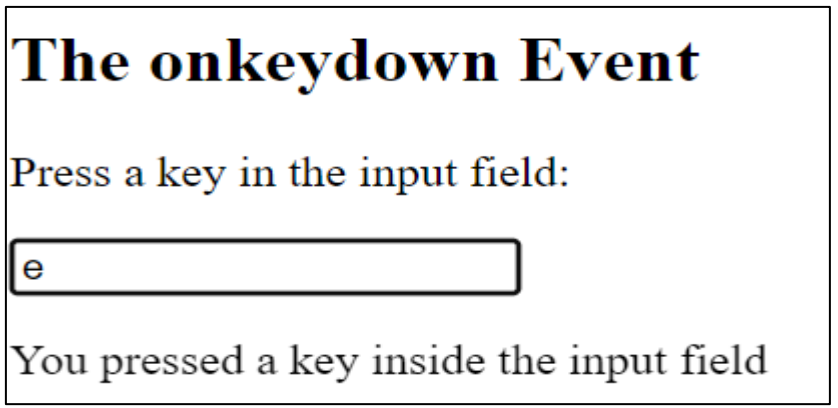
Keyboard Events:

1. onkeydown Event

The onkeydown event occurs when the user presses a key on the keyboard.

```
<html>
<body>
<h2>The onkeydown Event</h2>

<p>Press a key in the input field:</p>

<input type="text" onkeydown="myFunction()">

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

<script>
function myFunction() {
  document.getElementById("demo").innerHTML =
"You pressed a key inside the input field";
}
</script>
</body>
</html>
```





Warning

The **onkeypress** event is **deprecated**.

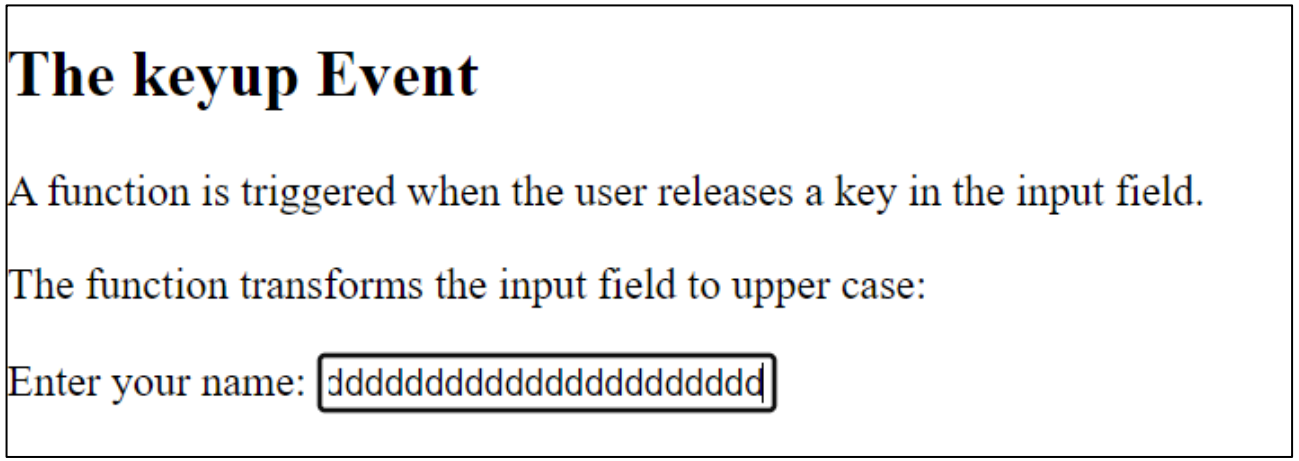
It is not fired for all keys (like ALT, CTRL, SHIFT, ESC) in all browsers.

To detect if the user presses a key, always use the **onkeydown** event. It works for all keys.

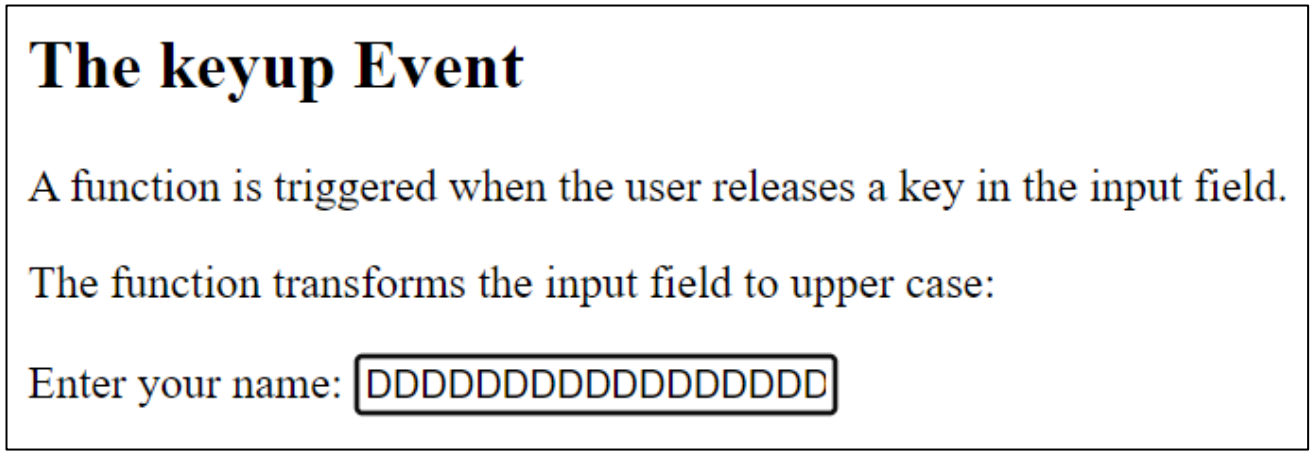
onkeyup Event

The onkeyup event occurs when the user releases a key on the keyboard.

```
<html><body>
<h2>The keyup Event</h2>
```




```
<p>A function is triggered when the user releases a key in the input field.</p>
<p>The function transforms the input field to upper case:</p>

Enter your name: <input type="text" id="fname" onkeyup="myFunction()">

<script>
function myFunction() {
  let x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script></body></html>
```



FocusEvent

The FocusEvent Object handles events that occur when elements gets or loses focus.

onblur Event

- The onblur event occurs when an HTML element loses focus.
- The onblur event is often used on input fields.
- The onblur event is often used with form validation (when the user leaves a form field).

```
<html>
<body>
<h1>HTML DOM Events</h1>
<h2>The blur Event</h2>

Enter your name: <input type="text" id="fname" onblur="myFunction()">

<p>When you leave the input field, a function is triggered which transforms the input text to upper case.</p>

<script>
function myFunction() {
  let x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}

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

onfocus Event:

- The onfocus event occurs when an element gets focus.
- The onfocus event is often used on input fields.

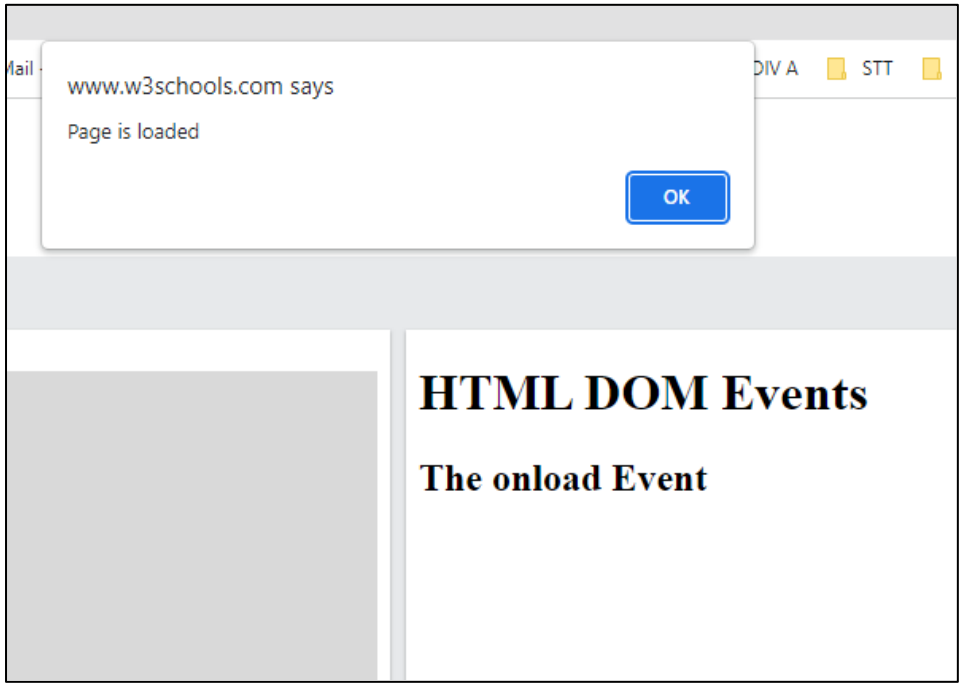
```
<html>
<body>
<h1>HTML DOM Events</h1>
<h2>The focus Event</h2>

Enter your name: <input type="text" onfocus="myFunction(this)">

<p>When the input field gets focus, a function changes the background-color.</p>

<script>
function myFunction(x) {
  x.style.background = "yellow";
}
</script>

</body>
</html>
```

Load Events:

Onload Events:

- The onload event occurs when an object has been loaded.
- onload is most often used within the <body> element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload event can also be used to deal with cookies

```
<html>
<body onload="myFunction()">
<h1>HTML DOM Events</h1>
<h2>The onload Event</h2>

<script>
function myFunction() {
  alert("Page is loaded");
}
</script>

</body>
</html>
```



Onunload event:

- The onunload event occurs once a page has unloaded (or the browser window has been closed).
- onunload occurs when the user navigates away from the page (by clicking on a link, submitting a form, closing the browser window, etc.).
- When a user tries to load a page but it is unsuccessful, the page remains unloaded, and a JavaScript onunload event is triggered. The possibility exists that the onunload event will also happen if the browsers have been closed while the page is loaded.
- When a user goes away from a page while selecting, submitting, clicking, and leaving the browser window, the onunload event typically enters the onunload state. Additionally, this incident happens when a user is trying to load a url orweb browser and accesses data while it is loaded.

```
<html>
<body onunload="myFunction()">

<h1>Welcome to my Home Page</h1>

<p>Close this window or press F5 to reload the page.</p>
<p><strong>Note:</strong> Due to different browser
settings, this event may not always work as expected.</p>

<script>
function myFunction() {
  alert("Thank you for visiting W3Schools!");
}
</script>

</body>
</html>
```

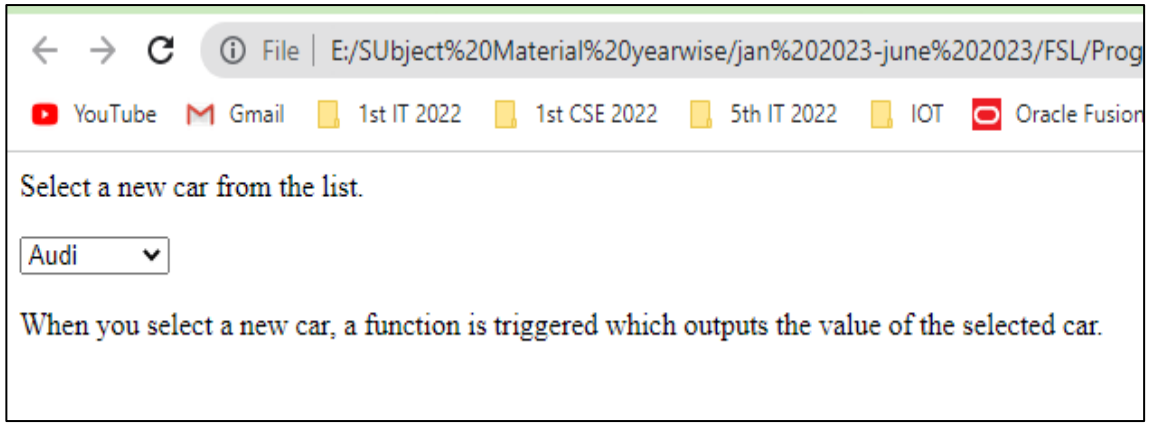
Note: Due to different browser settings, this event may not always work as expected.

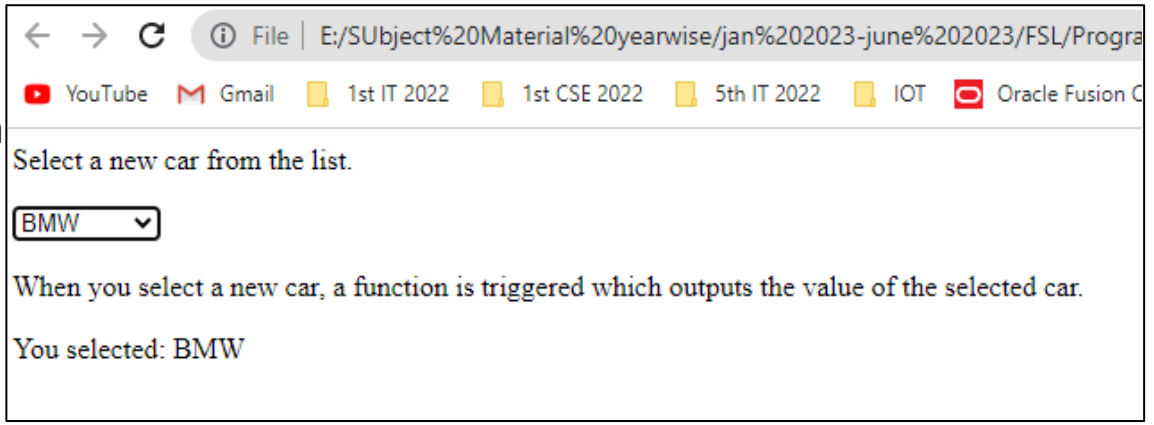
Other Events:

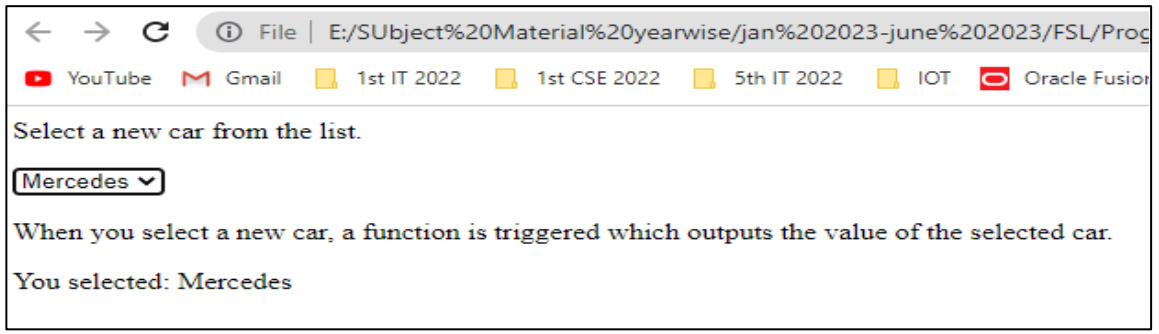
onchange Event

The onchange event occurs when the value of an HTML element is changed.

```
<html>
<body>
<p>Select a new car from the list.</p>
<select id="mySelect" onchange="myFunction()">
  <option value="Audi">Audi</option>
  <option value="BMW">BMW</option>
  <option value="Mercedes">Mercedes</option>
  <option value="Volvo">Volvo</option>
</select>
<p>When you select a new car, a function is triggered which
outputs the value of the selected car.</p>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementById("mySelect").value;
  document.getElementById("demo").innerHTML = "You
                                              selected: " + x;
}
</script>
</body>
</html>
```



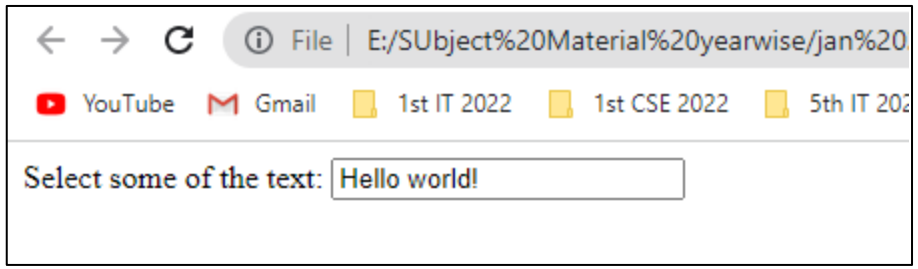


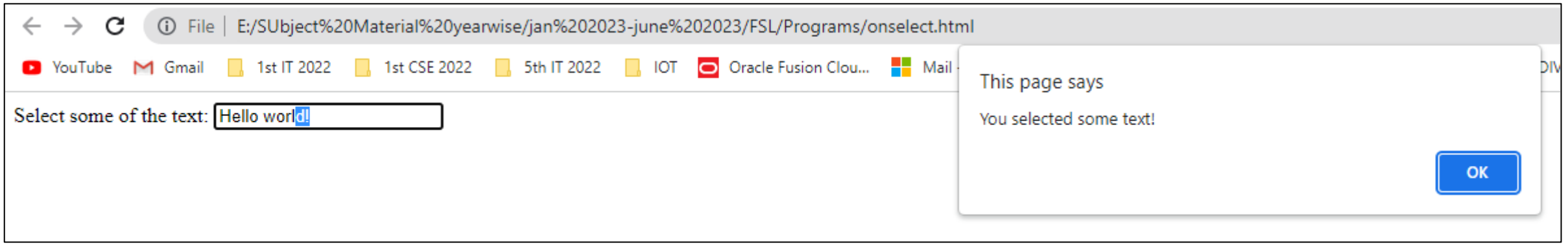


onselect Event

- The onselect event occurs after some text has been selected in an element.
- The onselect event is mostly used on <input type="text"> or <textarea> elements.

```
<html>
<body>
Select some of the text: <input type="text" value="Hello world!" onselect="myFunction()">
<script>
function myFunction() {
  alert("You selected some text!");
}
</script>
</body>
</html>
```

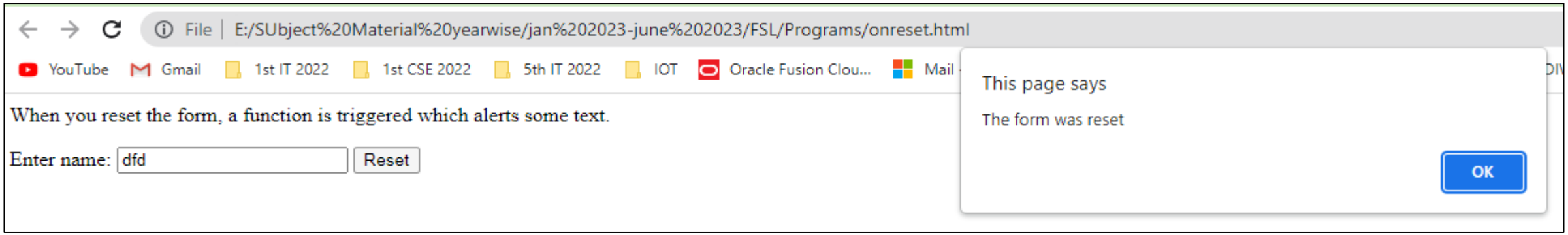




onreset Event

The onreset event occurs when a form is reset.

```
<html>
<body>
<p>When you reset the form, a function is triggered which alerts some text.</p>
<form onreset="myFunction()">
  Enter name: <input type="text">
  <input type="reset">
</form>
<script>
function myFunction() {
  alert("The form was reset");
}
</script>
</body>
</html>
```



Changing text of labels

```
<html><head><title>
      How to change the text ofa label using JavaScript ?
   </title></head>
 <body style="text-align:center;">
       <h1 style="color:green;">GeeksforGeeks</h1>
    <h4>Click on the button to changethe text of a label</h4>
    <label id = "GFG">Welcome to GeeksforGeeks</label>
       <br>
       <button onclick="myGeeks()">Click Here!</button>
    <script>
     function myGeeks()
              {
                   var x = document.getElementById("GFG");
                   if (x.innerHTML === "Welcome to GeeksforGeeks")
                   {
                     x.innerHTML = "A computer science portal for geeks";
                   }
                   else
                  {
                    x.innerHTML = "Welcome to GeeksforGeeks";
                  }
              }
   </script>
</body></html>
```

Chapter 4-Cookies

JavaScript Cookies

- A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.
- A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

What are Cookies?

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user":
- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

Cookies are saved in name-value pairs like:

username = John Doe

Create a Cookie with JavaScript

- JavaScript can create, read, and delete cookies with the document.cookie property.

```
document.cookie = "username=John Doe";
```

- You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2023 12:00:00 UTC";
```

- With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

- Read a Cookie with JavaScript

```
let x = document.cookie;
```

document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

Change a Cookie with JavaScript

```
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

The old cookie is overwritten.

Delete a Cookie with JavaScript

You don't have to specify a cookie value when you delete a cookie.

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

- You should define the cookie path to ensure that you delete the right cookie.
- Some browsers will not let you delete a cookie if you don't specify the path.

Examples:

When you assign a string to document.cookie, the browser parses it as a cookie and adds it to its list of cookies. There are several parts to each cookies, many of them optional.

Syntax: -

document.cookie = "name=value";

document.cookie = "name=value; expires=date; domain=domain; path=path; secure";

document.cookie = "name=value; max-age=inSecond; domain=domain; path=path; secure";

Ex:-

document.cookie = "username=geeky";

document.cookie = "username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC";

Note - name-value pair must not contain any whitespace character, Commas or semicolons.

Ex: - username=geeky shows;

Not allowed

Optional Cookies Attribute:-

max-age
expires
domain
path
secure

Whenever you omit the optional cookie fields, the browser fills them in automatically with reasonable defaults.

Attributes	Description
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

Creating a cookies

1st method

Example 1:

```
<html>
   <body>
      <h1>cookies</h1>
   <script>
      document.cookie="username=neha";----------Creating cookies(set cookies)
      alert(document.cookie);
   </script>
   </body>
</html>
```

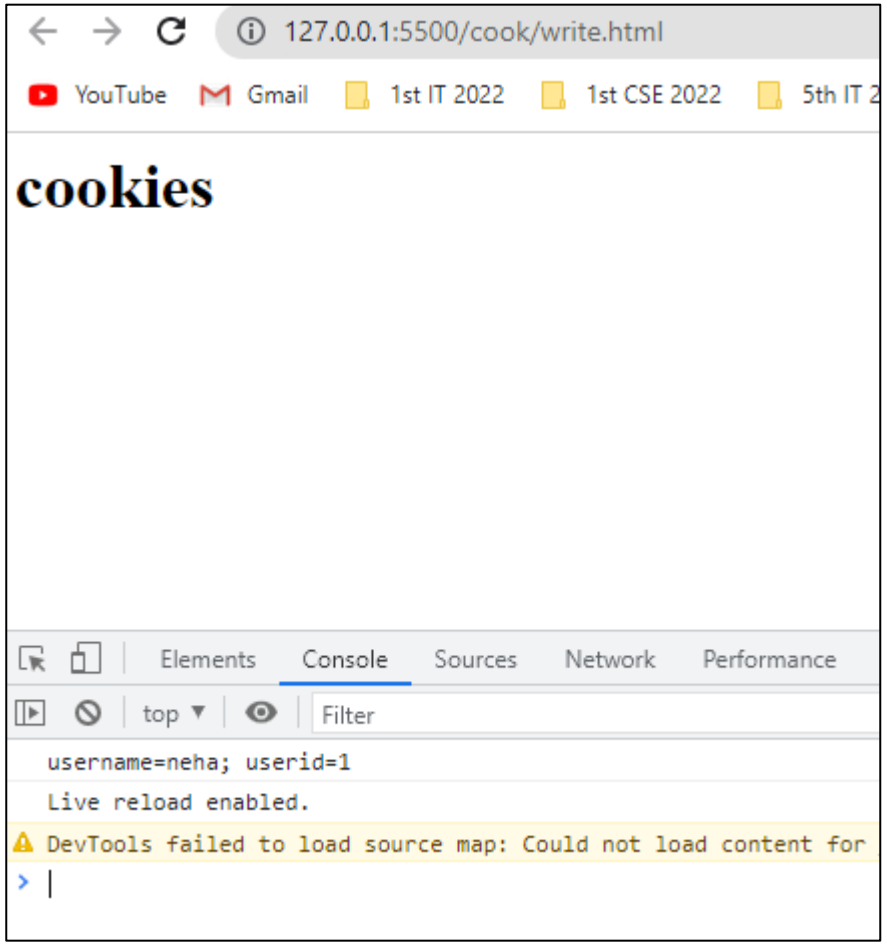
2nd Method

Terminal Help

Release Notes: 1.77.3 | write.html

cook > write.html > html > body > script

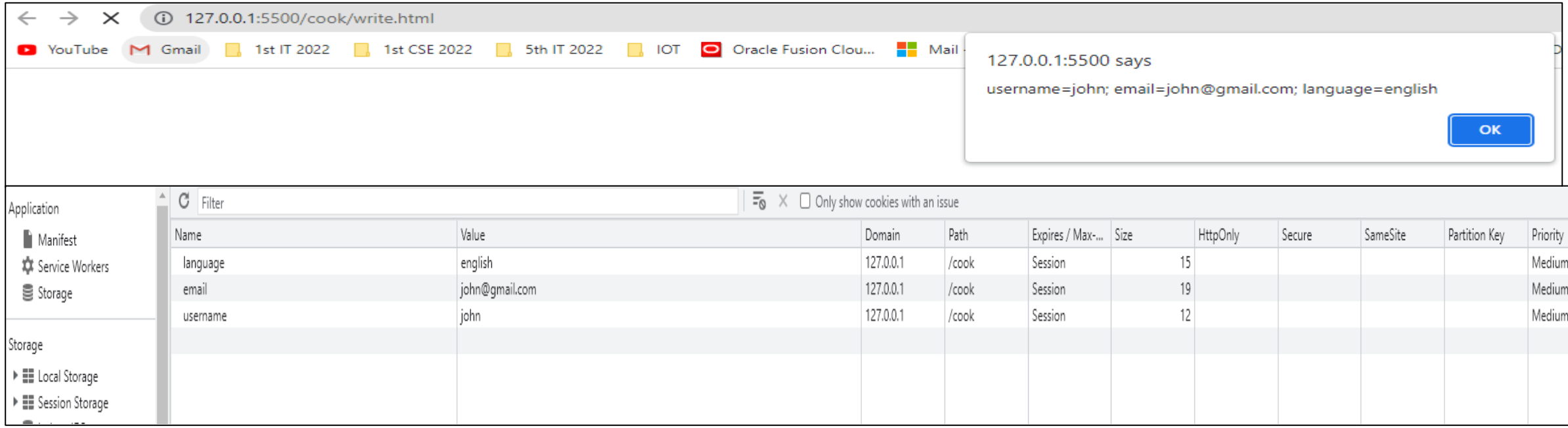
```
1   <html>
2       <body>
3           <h1>cookies</h1>
4       <script>
5           document.cookie="username=neha";
6           document.cookie="userid=1";
7           console.log(document.cookie);
8       </script>
9       </body>
10  </html>
```



Example:

```
<form action="" name="myform" method=""post">
        Name:<input type="text" name="name"><br><br>
        Email:<input type="email" name="email"><br><br>
        Language:<input type="text" name="language"><br><br>
        <input type="button" value="setcookies" name="set" onclick="setcookies()">
        <input type="button" value="getcookies" name="get" onclick="getcookies()">
      </form>
```

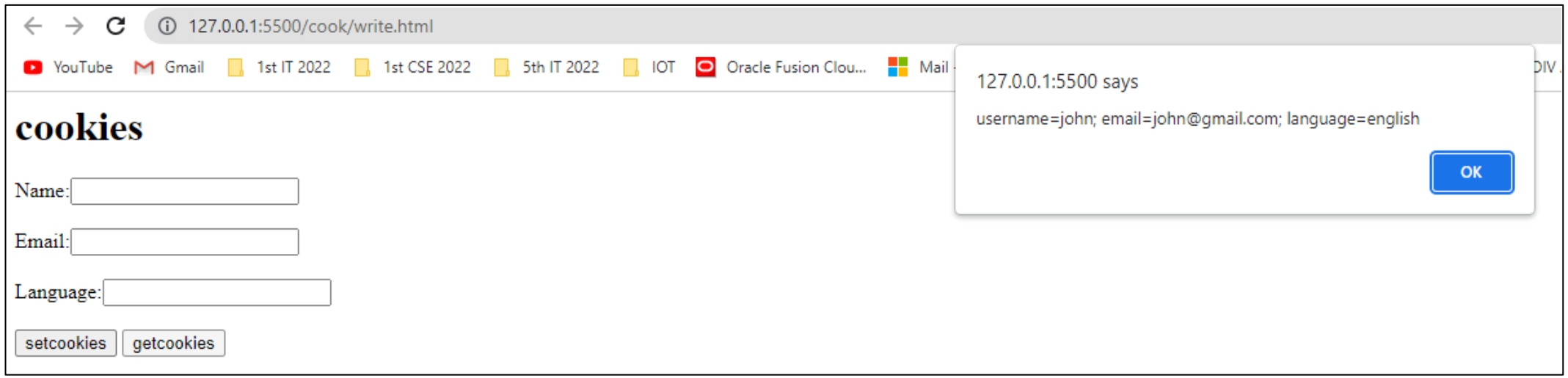
```
function setcookies()
      {  document.cookie="username=" + document.myform.name.value;
         document.cookie="email=" + document.myform.email.value;
         document.cookie="language=" + document.myform.language.value;
      }
      alert(document.cookie);
```



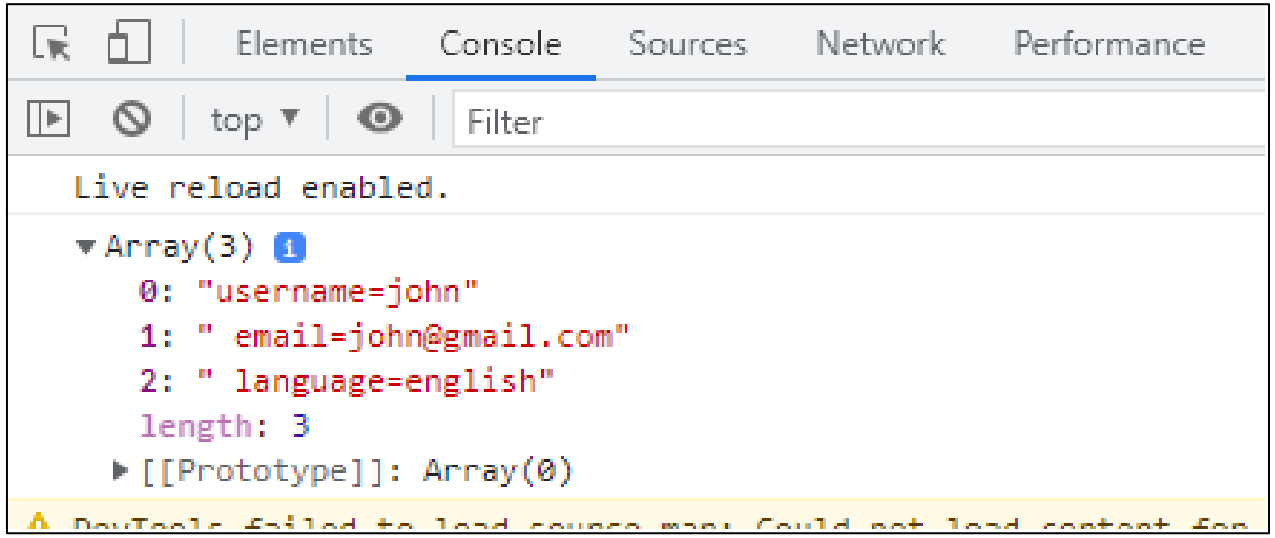
Example:

```
<html><body><h1>cookies</h1>
    <form action="" name="myform" method=""post">
      Name:<input type="text" name="name"><br><br>
      Email:<input type="email" name="email"><br><br>
      Language:<input type="text" name="language"><br><br>
      <input type="button" value="setcookies" name="set" onclick="setcookies()">
      <input type="button" value="getcookies" name="get" onclick="getcookies()">
    </form>
  <script>
    function setcookies()
    {
      document.cookie="username=" + document.myform.name.value;
      document.cookie="email=" + document.myform.email.value;
      document.cookie="language=" + document.myform.language.value;
    }
```

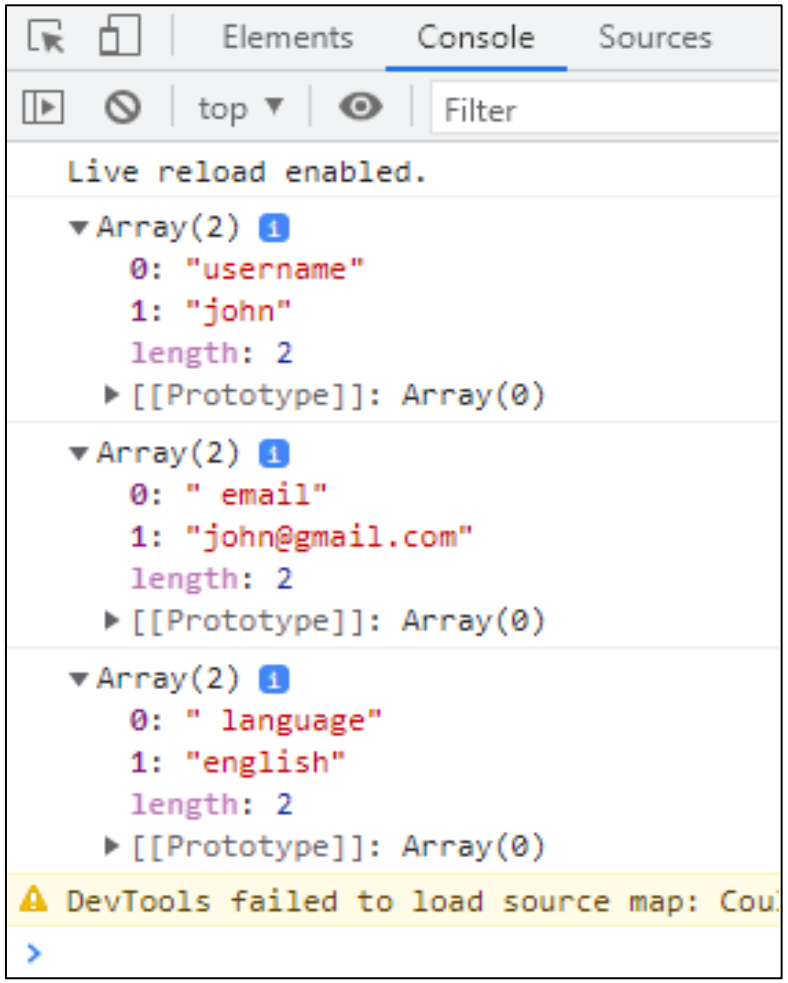
```
//username=neha;email=123@gmail.com;la
nguage=english
    function getcookies()
    {
      var cookiesarray=document.cookie;
      alert(cookiesarray);
    }
  </script></body></html>
```



```
function getcookies()
      {
         var cookiesarray=document.cookie.split(";");
         console.log(cookiesarray);
      }
```



```
function getcookies()
	{
	   var cookiesarray=document.cookie.split(";");
	   for(var i=0;i<cookiesarray.length;i++)
	   {
	   var valuearray=cookiesarray[i].split("=");
	   console.log(valuearray);
	   }
	}
```

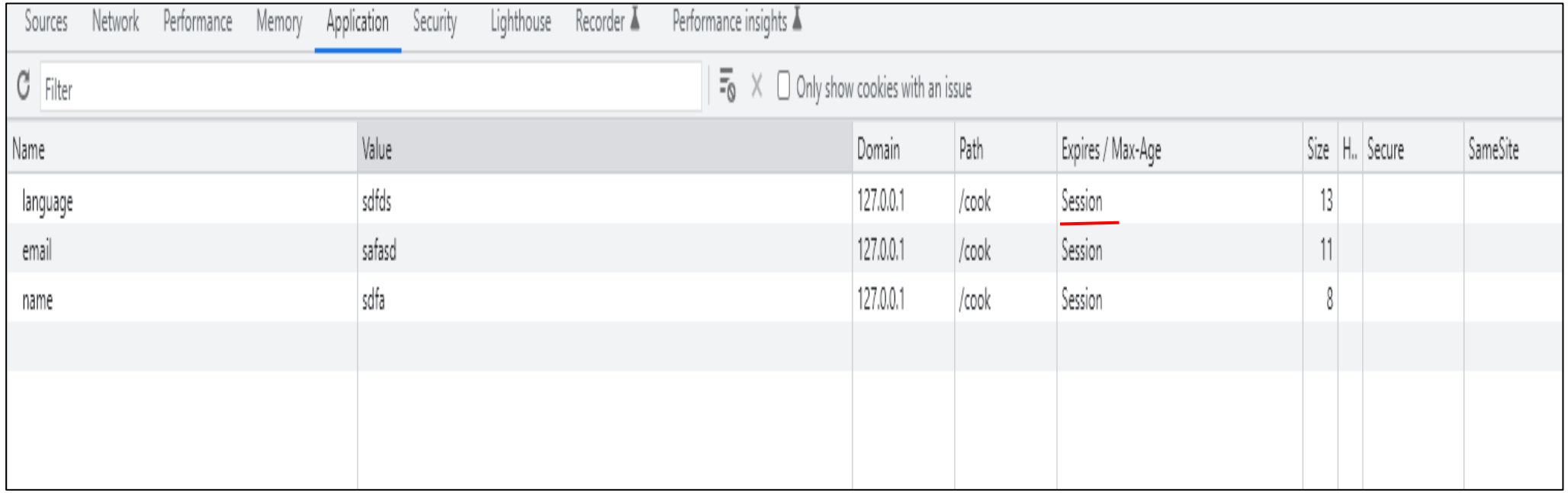


Example1

```
<html>
   <body>
      <h1>cookies</h1>
      <form action="" name="myform" method=""post">
         Name:<input type="text" name="name"><br><br>
         Email:<input type="email" name="email"><br><br>
         Language:<input type="text" name="language"><br><br>
         <input type="button" value="setcookies" name="set" onclick="setcookies()">
         <input type="button" value="getcookies" name="get" onclick="getcookies()">
      </form>
   <script>
      function setcookies()
      {
         document.cookie="name=" + document.myform.name.value;
         document.cookie="email=" + document.myform.email.value;
         document.cookie="language=" + document.myform.language.value;
      }
      //alert(document.cookie);
      //username=neha;email=123@gmail.com;language=english
```

```
function getcookies()
      {
         var cookiesarray=document.cookie.split(";");
         for(var i=0;i<cookiesarray.length;i++)
         {
         var valuearray=cookiesarray[i].split("=");
         if(valuearray[0].trim() =='name')
         {
            document.myform.name.value=valuearray[1];
         }
         else if(valuearray[0].trim() =='email')
         {
            document.myform.email.value=valuearray[1];
         }
         else  if(valuearray[0].trim() =='language')
         {
            document.myform.language.value=valuearray[1];
         }
         }
      }
   </script>
   </body>
</html>
```

Cookies value



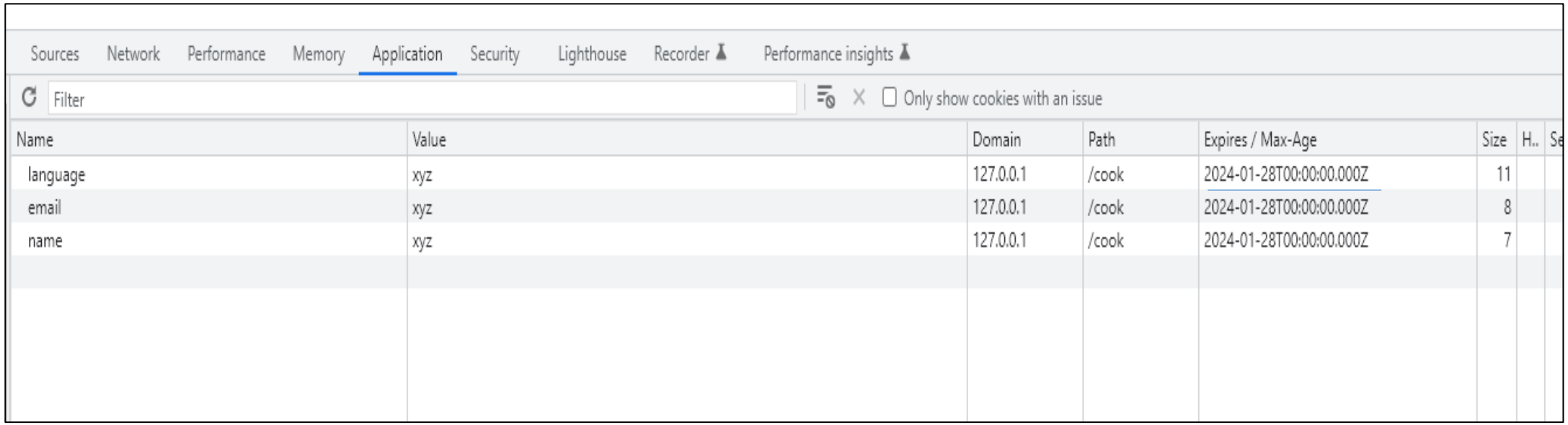
Name	Value	Domain	Path	Expires / Max-Age	Size	H..	Secure	SameSite
language	sdfds	127.0.0.1	/cook	Session	13			
email	safasd	127.0.0.1	/cook	Session	11			
name	sdfa	127.0.0.1	/cook	Session	8			

Cookies expiry property

```
<html>
  <body>
    <h1>cookies</h1>
    <form action="" name="myform" method=""post">
      Name:<input type="text" name="name"><br><br>
      Email:<input type="email" name="email"><br><br>
      Language:<input type="text" name="language"><br><br>
      <input type="button" value="setcookies" name="set" onclick="setcookies()">
      <input type="button" value="getcookies" name="get" onclick="getcookies()">
    </form>
  <script>
    function setcookies()
    {
      document.cookie="name=" + document.myform.name.value+"; expires=Thu, 28 Jan 2024 00:00:00 UTC";
      document.cookie="email=" + document.myform.email.value+"; expires=Thu, 28 Jan 2024 00:00:00 UTC";
      document.cookie="language=" + document.myform.language.value+"; expires=Thu, 28 Jan 2024 00:00:00 UTC";
    }
    //alert(document.cookie);
    //username=neha;email=123@gmail.com;language=english
```

```
function getcookies()
      {
         var cookiesarray=document.cookie.split(";");
         for(var i=0;i<cookiesarray.length;i++)
         {
         var valuearray=cookiesarray[i].split("=");
         if(valuearray[0].trim() =='name')
         {
            document.myform.name.value=valuearray[1];
         }
         else if(valuearray[0].trim() =='email')
         {
            document.myform.email.value=valuearray[1];
         }
         else  if(valuearray[0].trim() =='language')
         {
            document.myform.language.value=valuearray[1];
         }
         }
      }
   </script>
   </body>
</html>
```

Cookies value



Delete an cookie:

Set a cookie

```
<html>
   <body>
      <h1>cookies</h1>
         <script>
      document.cookie="adderss=borivali ; expires=Thu, 28 Jan 2024 00:00:00 UTC";
   </script>
   </body>
</html>
```

Delete cookie

Method 1:-setting past date

```
<script>
      document.cookie="adderss=borivali ; expires=Thu, 28 Jan 2021 00:00:00 UTC";
   </script>
```

Method 2:-setting an empty string

```
<script>
      document.cookie="adderss= ; expires=Thu, 28 Jan 2021 00:00:00 UTC";
   </script>
```

Delete an cookie:

```
<html>
    <body>
        <h1>cookies</h1>
            <script>
        document.cookie="address=borivali ; max-age="+60*60*24*10;
    </script>
    </body>
</html>
```

```
<html>
    <body>
        <h1>cookies</h1>
            <script>
        document.cookie="address=borivali ; max-age="-60;
    </script>
    </body>
</html>
```

References:

Write a cookie:
https://www.youtube.com/watch?v=Pl7p-gPGH5c

Read a cookie:
https://www.youtube.com/watch?v=IRLXrVX12Oc

Delete an cookie
https://www.youtube.com/watch?v=Q2ClLmlZb-o

JavaScript Browser Objects

JavaScript Browser Objects:

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

The Window Object

- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.

The open() method opens a new browser window, or a new tab, depending on your browser settings and the parameter values.

Syntax
window.open(*URL, name, specs, replace*)

Parameters

Parameter	Description
URL	Optional. The URL of the page to open. If no URL is specified, a new blank window/tab is opened
name	Optional. The target attribute or the name of the window. The following values are supported:

Value	Description
_blank	URL is loaded into a new window, or tab. This is the default
_parent	URL is loaded into the parent frame
_self	URL replaces the current page
_top	URL replaces any framesets that may be loaded
name	The name of the window (does not specify the title of the window)

specs	Optional. A comma-separated list of items, no whitespaces. The following values are supported:

fullscreen=yes|no|1|0	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only
height=pixels	The height of the window. Min. value is 100
left=pixels	The left position of the window. Negative values not allowed
location=yes|no|1|0	Whether or not to display the address field. Opera only
menubar=yes|no|1|0	Whether or not to display the menu bar
resizable=yes|no|1|0	Whether or not the window is resizable. IE only
scrollbars=yes|no|1|0	Whether or not to display scroll bars. IE, Firefox & Opera only
status=yes|no|1|0	Whether or not to add a status bar
titlebar=yes|no|1|0	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box
toolbar=yes|no|1|0	Whether or not to display the browser toolbar. IE and Firefox only
top=pixels	The top position of the window. Negative values not allowed
width=pixels	The width of the window. Min. value is 100

replace	Deprecated

1. Open a new window when clicking on button:

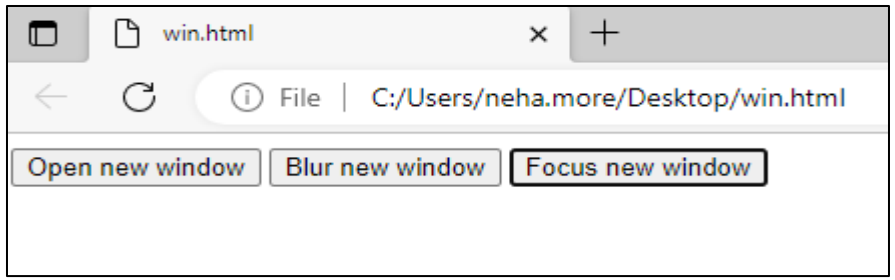
```
<script>
function openWin()
{
  window.open("https://www.w3schools.com");
}
</script>
</head>
<body>
<form>
  <input type="button" value="Open Window" onclick="openWin()">
</form>
```

Open a new window and control its appearance

```
<html>
<head>
<script>
function openWin() {
  window.open("https://www.w3schools.com","_blank","toolbar=yes, status=no, menubar=yes, scrollbars=yes,
resizable=no, width=400, height=400");
}
</script>
</head>
<body>
<form>
  <input type="button" value="Open Window" onclick="openWin()">
</form>
</body>
</html>
```

Blurr and focus a new window:

```
<html>
<head>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "", "width=400, height=200");
  }
function blurWin() {
  myWindow.blur();
}
function focusWin() {
  myWindow.focus();
}
</script>
</head>
<body>

<input type="button" value="Open new window" onclick="openWin()">
<input type="button" value="Blur new window" onclick="blurWin()">
<input type="button" value="Focus new window" onclick="focusWin()">

</body>
</html>
```



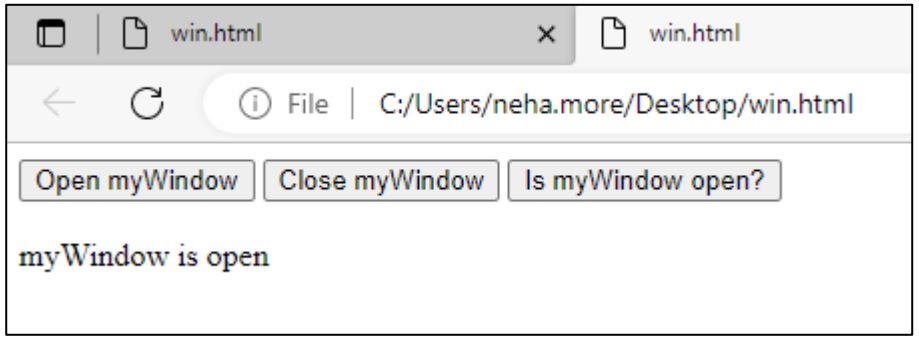
Close the new window:

```
<html>
<head>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "", "width=400, height=200");
}

function closeWin() {
  myWindow.close();
}
</script>
</head>
<body>

<input type="button" value="Open 'myWindow'" onclick="openWin()" />
<input type="button" value="Close 'myWindow'" onclick="closeWin()" />

</body>
</html>
```

Check whether new window has closed or not

```
<html>
<head>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "", "width=400 ,height=200");
}
function closeWin() {
  if (myWindow) {
    myWindow.close();
  }
}
function checkWin() {
  msg = ""
  if (!myWindow) {
    msg = "was never opened";
  } else {
    if (myWindow.closed) {
      msg = "is closed";
    } else {
      msg = "is open";
    }
    }
  document.getElementById("msg").innerHTML =
  "myWindow " + msg;
}
```

```
</script>
</head>
<body>
<button onclick="openWin()">Open myWindow</button>
<button onclick="closeWin()">Close myWindow</button>
<button onclick="checkWin()">Is myWindow open?</button>
<br><br>
<div id="msg"></div>
</body>
</html>
```



Move a new window to the specified position

```
<html>
<head>
<script>
var myWindow;
function openWin() {
  myWindow=window.open("", "", "width=400, height=200");
}

function moveWin() {
  myWindow.moveTo(300, 0);
  myWindow.focus();
}
</script>
</head>
<body>

<input type="button" value="Open myWindow" onclick="openWin()" />
<br><br>
<input type="button" value="Move myWindow" onclick="moveWin()" />

</body>
</html>
```

window.moveTo(x, y)

Parameter	Description
x	Required. A positive or negative number. The horizontal coordinate to move to.
y	Required. A positive or negative number. The vertical coordinate to move to.

Print the current page

```
<html>
<head>
<script>
function printPage() {
  window.print();
}
</script>
</head>
<body>

<input type="button" value="Print this page"
onclick="printPage()" />

</body>
</html>
```

Resize the window by the specified pixels

```
<html>
<head>
<script>
var w;
function openwindow() {
  w = window.open('','', 'width=100,height=100');
  w.focus();
}

function myFunction() {
  w.resizeBy(50, 50);
  w.focus();
}

</script>
</head>
<body>

<button onclick="openwindow()">Create window</button>
<button onclick="myFunction()">Resize window</button>

</body>
</html>
```

The resizeBy() method resizes a window by a specified amount relative to its current size. .

resizeBy(width, height)

Parameter	Description
width	Required. A positive or a negative number. The number of pixels to resize the width by.
height	Required. A positive or a negative number. The number of pixels to resize the height by.

Resize a window to a specified size

```
<html>
<head>
<script>
var w;
function openwindow() {
  w = window.open('','', 'width=100,height=100');
  w.focus();
}

function myFunction() {
  w.resizeTo(500, 500);
  w.focus();
}

</script>
</head>
<body>

<button onclick="openwindow()">Create window</button>
<button onclick="myFunction()">Resize window</button>

</body>
</html>
```

The resizeTo() method resizes a window to a new width and height.

window.resizeTo(width, height)

Parameter	Description
width	Required. The new window width, in pixels
height	Required. The new window height, in pixels

Scroll the content by the specified number of pixcel

Window scrollBy()

The scrollBy() method scrolls the document by the specified number of pixels.

Syntax:

window.scrollBy(x, y)

Or

scrollBy(*x, y*)

Parameter	Description
x	Required. Number of pixels to scroll (horizontally). Positive values scroll to the right, negative values to the left.
y	Required. Number ofpixels to scroll (vertically). Positive values scroll down, negative values scroll up.

Scroll the document 100px horizontally:

```
<html>
<style>
body {width: 5000px}
button {position:fixed}
</style>
<body>
<h1>The Window Object</h1>
<h2>The scrollBy() Method</h2>
<p>Click to scroll the document.</p>
<p>Look at the horizontal scrollbar to see the effect.</p>
<button onclick="scrollWin()" style="position:fixed">Scroll 100px horizontally!</button>
<br><br>
<script>
function scrollWin() {
  window.scrollBy(100, 0);
}
</script>
</body>
</html>
```

Scroll the document 100px vertically:

```
<html>
<body>
<h1>The Window Object</h1>
<h2>The scrollBy() Method</h2>
<p>Click to scroll the document.</p>
<button onclick="scrollWin()" style="position:fixed">Scroll 100px
vertically!</button>
<br><br>

<h3>Some line breaks to enable scrolling:</h3>
<br>
<br>
<br>
<p>When I find myself in times of trouble</p>
<br>
<br>
<br>
<p>Mother Mary comes to me</p>
<br>
<br>
<br>
<p>Speaking words of wisdom, let it be</p>
<br>
<br>
<br>
<p>And in my hour of darkness she is standing right in front of me</p>
<br>
<br>
<br>
<p>Speaking words of wisdom, let it be</p>
<br>
<br>
<br>
<p>ROCK AND ROLL</p>
<br>
<br>
<br>
<p>SCROLL SCROLL SCROLL</p>
<br>
<br>
<br>
<p>ROCK AND ROLL</p>
<br>
<br>
<br>
<p>SCROLL SCROLL SCROLL</p>
<br>
<br>
<br>
<p>ROCK AND ROLL</p>
<br>
<br>
<br>
<p>ROCK AND ROLL</p>
<br>
<br>
<br>
<script>
function scrollWin() {
  window.scrollBy(0, 100);
}
</script>
</body>
</html>
```

The scrollTo()

This method scrolls the document to specified coordinates.

Syntax
`window.scrollTo(x, y)`
or
`just:scrollTo(x, y)`

Parameter	Description
x	Required. The coordinate to scroll to (horizontally), in pixels.
y	Required. The coordinate to scroll to (vertically), in pixels.

Scroll the document to the horizontal position 200:

```
<style>
body {
  width: 5000px;
}
</style>

<body>
<h1>The Window Object</h1>
<h2>The scrollTo() Method</h2>

<p>Click to scroll the document.</p>

<button onclick="scrollWin()" style="position:fixed">Scroll to 200
horizontally!</button><br><br>

<script>
function scrollWin() {
  window.scrollTo(200, 0);
}
</script>
```

Scroll the document to the vertical position 500:

```
<style>
body {
  height: 5000px;
}
</style>

<body>
<h1>The Window Object</h1>
<h2>The scrollTo() Method</h2>

<p>Click to scroll the document.</p>

<button onclick="scrollWin()" style="position:fixed">Scroll to 500
vertically!</button><br><br>

<script>
function scrollWin() {
  window.scrollTo(0, 500);
}
</script>
```

Window scrollX

- The scrollX property returns the pixels a document has scrolled from the upper left corner of the window.
- The scrollX property is read-only.

Syntax
`window.scroll`
or
just:
`scrollX`

Window scrollY

- The scrollY property returns the pixels a document has scrolled from the upper left corner of the window.
- The scrollY property is read-only.

Syntax
`window.scrollY`
or just:
`scrollY`

Window scrollX and scrollY

```
<html>
<head>
<style>
div {
  background-color: lightblue;
  height: 2000px;  width: 2000px;
}
</style>
</head>
<body>
<h1>The Window Object</h1>
<h2>The scrollX and scrollY Properties</h2>
<p>Click the button to scroll the document window 100px horizontally and vertically.</p>
<button onclick="myFunction()" style="position:fixed;">Click me to scroll</button><br><br>
<div></div>
<script>
function myFunction() {
  window.scrollBy(100, 100);
  alert("pageXOffset: " + window.scrollX + ", scrollY: " + window.scrollY);
}
</script>
</body>
</html>
```

Screen Object

The screen object contains information about the visitor's screen.

Screen Object Properties

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

availHeight

- The availHeight property returns the height of the user's screen.
- The availHeight property returns the height in pixels.
- The availHeight property returns the height minus interface features like the Windows Taskbar.

```
screen.availHeight
```

Example:

```
<p id="demo"></p>
<script>
height = screen.availHeight;
document.getElementById("demo").innerHTML = height + "px";
</script>
```

1040px

availWidth

- The availWidth property returns the width of the user's screen.
- The availWidth property returns the width in pixels.
- The availWidth property returns the width minus interface features like the Windows Taskbar.

Example:

```
<p id="demo"></p>
<script>
let width = screen.availWidth;
document.getElementById("demo").innerHTML = width + "px";
</script>
```

1920px

Window screen.height

- The height property returns the total height of the user's screen.
- The height property returns the height in pixels.
- The height property is read only.

```
<p id="demo"></p>
<script>
let height = screen.height;
document.getElementById("demo").innerHTML =  height + "px";
</script>
```

Window screen.width

- The width property returns the total width of the user's screen.
- The width property returns width in pixels.
- The width property is read-only.

```
<p id="demo"></p>
<script>
let width = screen.width;
document.getElementById("demo").innerHTML =  width + "px";
</script>
```

Window screen.colorDepth

- The colorDepth property returns the screen's color depth.
- The colorDepth property returns the depth in bits per pixel.
- The colorDepth property is read-only.

```
<p id="demo"></p>
<script>
let depth = screen.colorDepth;
document.getElementById("demo").innerHTML =  depth + " bits per pixel";
</script>
```

24 bits per pixel

Window screen.pixelDepth

- The pixelDepth property returns the screen's color depth.
- The pixelDepth property returns the color depth in bits per pixel.
- The pixelDepth property is read-only.

```
<p id="demo"></p>
<script>
let depth = screen.pixelDepth;
document.getElementById("demo").innerHTML = depth + " bits per pixel";
</script>
```

24 bits per pixel

Location Object

- The location object contains information about the current URL.
- The location object is a property of the window object.
- The location object is accessed with:

  window.location or just location

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

Window location.hash

- The location.hash property sets or returns the anchor part of a URL, including the hash sign (#).
- When location.hash is used to set the anchor part, do not include the hash sign (#).

```
<html>
<body>
<h1>The Window Location Object</h1>
<h2>The hash Property</h2>
<p><a id="w3s" href="/js/js_es6.asp#mark_array_from">JavaScript
2015 Array.from()</a><p>
<p id="demo"></p>
<script>
let url = document.getElementById("w3s");
document.getElementById("demo").innerHTML = "The anchor portion
of the URL is: " + url.hash;
</script>
</body>
</html>
```

The Window Location Object
The hash Property
JavaScript 2015 Array.from()
The anchor portion of the URL is: #mark_array_from

```
<p><a id="w3s" href="/js/js_es6.asp#mark_array_from">JavaScript
2015 Array.from()</a><p>
<p id="demo"></p>
<script>
location.hash = "mark_array_find";
document.getElementById("demo").innerHTML = "The anchor part is
now: " + location.hash;
</script>
```

JavaScript 2015 Array.from()
The anchor part is now: #mark_array_find

Window location.host

The location.host property returns the host (IP adress or domain) and port of a URL.

```
<h1>The Window Location Object</h1>
<h2>The host Property</h2>
<p id="demo"></p>
<script>
let host = location.host;
document.getElementById("demo").innerHTML = host;
</script>
```

If the port number is not specified in the URL, or if it is a default port (80 for http) or (443 for https), most browsers will return an empty string.

Port 53: Domain Name System (DNS). ...
Port 80: Hypertext Transfer Protocol (HTTP). ...
Port 123: Network Time Protocol (NTP).

Window location.href

The location.href property sets or returns the entire URL of the current page.

```
<html>
<body>
<h1>The Window Location Object</h1>
<h2>The href Property</h2>
<p id="demo"></p>
<script>
let url = location.href;
document.getElementById("demo").innerHTML = url;
</script>
</body>
</html>
```

Set the URL of the current page:

```
<p>Click the button to set the href value to https://www.w3schools.com.</p>
<button onclick="myFunction()">Take me to w3schools.com</button>
<script>
function myFunction() {
  location.href = "https://www.w3schools.com";
}
</script>
```

Window location.origin

- The origin property returns the protocol, hostname and port number of a URL.
- The origin property is read-only.
- If the port number is not in the URL, or if it is a default port like 80 (Http), or 443 (https), some browsers will not display the port number.

```
<p id="demo"></p>
<script>
let origin = location.origin;
document.getElementById("demo").innerHTML = origin;
</script>
```

Window location.pathname

The pathname property sets or returns the pathname of a URL (page).

```
<html>
<body>
<h1>The Window Location Object</h1>
<h2>The pathname Property</h2>
<p id="demo"></p>
<script>
let path = location.pathname;
document.getElementById("demo").innerHTML = path;
</script>
</body>
</html>
```

Window location.port

- The port property sets or returns the port number of a URL.
- If the port number is not specified in the URL, or if it is a default port (80 for http) or (443 for https), most browsers will return an empty string.

```
<h1>The Window Location Object</h1>
<h2>The port Property</h2>

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

<p><b>Note: </b>If the port number is default (80 for http and 443 for https), most browsers return nothing.</p>

<script>
let port = location.port;
document.getElementById("demo").innerHTML = "The port number of the current page is: " + port;
</script>

</body>
</html>
```

Window location.protocol

- The protocol property sets or returns the protocol of the current URL, including the colon (:).
- The protocol is a standard that specifies how data are transmitted between computers.

Parameter	Description
protocol	The protocol of the URL. •Examples:file: •ftp: •http: •https: •mailto:

```
<p id="demo"></p>
<script>
let protocol = location.protocol;
document.getElementById("demo").innerHTML = protocol;
</script>
```

Window location.reload()

- The reload() method reloads the current document.
- The reload() method does the same as the reload button in your browser.

```
<script>
location.reload();
</script>
```

Window location.search

- The search property returns the querystring part of a URL, including the question mark (?).
- The search property can also be used to set the querystring.
- The querystring part is the part of the URL after the question mark (?).
- The querystring is used for parameter passing.

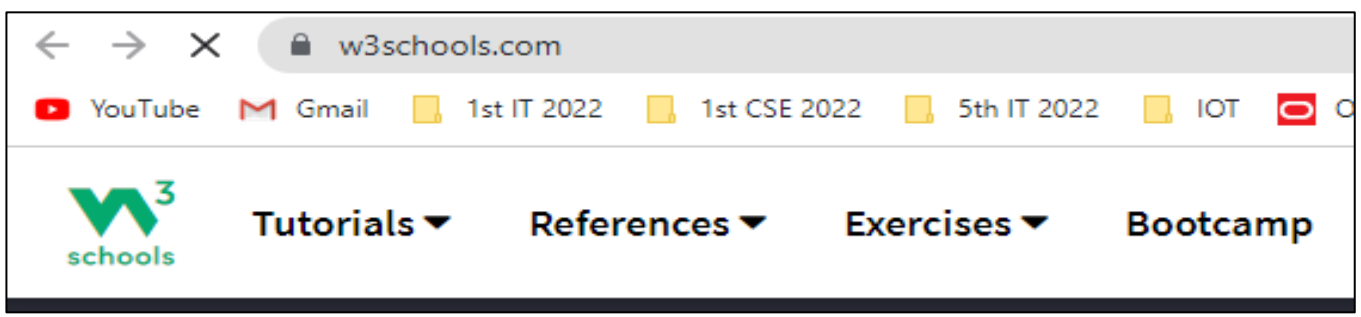
```
<p><a id="w3s" href="https://www.w3schools.com/?answer=yes">
https://www.w3schools.com/?answer=yes</a></p>
<p id="demo"></p>
<script>
let anchor = document.getElementById("w3s");
let query = anchor.search;
document.getElementById("demo").innerHTML = "The query portion of the URL is: " + query;
</script>
```

The Window Location Object
The search Property
https://www.w3schools.com/?answer=yes
The query portion of the URL is: ?answer=yes

Window location.replace()

- The replace() method replaces the current document with a new one.

```
<button onclick="myFunction()">Replace document</button>
<script>
function myFunction() {
  location.replace("https://www.w3schools.com")
}
</script>
```



JavaScript Timing Events

- The window object allows execution of code at specified time intervals.

- These time intervals are called timing events.

- The two key methods to use with JavaScript are:

- setTimeout(function, milliseconds)
  Executes a function, after waiting a specified number of milliseconds.

- setInterval(function, milliseconds)
  Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() Method

```
window.setTimeout(function, milliseconds);
```

- The window.setTimeout() method can be written without the window prefix.
- The first parameter is a function to be executed.
- The second parameter indicates the number of milliseconds before execution.

Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<html>
<body>
<h2>JavaScript Timing</h2>
<p>Click "Try it". Wait 3 seconds, and the page will alert "Hello".</p>
<button onclick="setTimeout(myFunction, 3000);">Try it</button>
<script>
function myFunction() {
  alert('Hello');
}
</script>
</body>
</html>
```

Stop the Execution

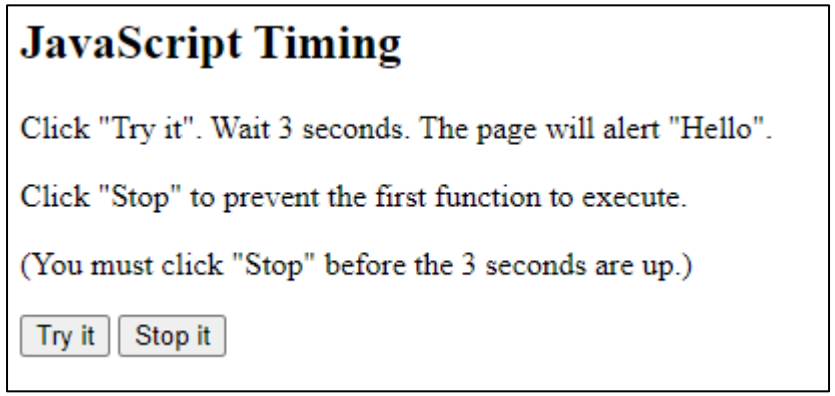
The clearTimeout() method stops the execution of the function specified in setTimeout().

```
window.clearTimeout(timeoutVariable)
```

- The window.clearTimeout() method can be written without the window prefix.
- The clearTimeout() method uses the variable returned from setTimeout():

If the function has not already been executed, you can stop the execution by calling the clearTimeout() method:

```
<html>
<body>
<h2>JavaScript Timing</h2>
<p>Click "Try it". Wait 3 seconds. The page will alert "Hello".</p>
<p>Click "Stop" to prevent the first function to execute.</p>
<p>(You must click "Stop" before the 3 seconds are up.)</p>
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>
<button onclick="clearTimeout(myVar)">Stop it</button>
<script>
function myFunction() {
  alert("Hello");
}
</script>
</body>
</html>
```

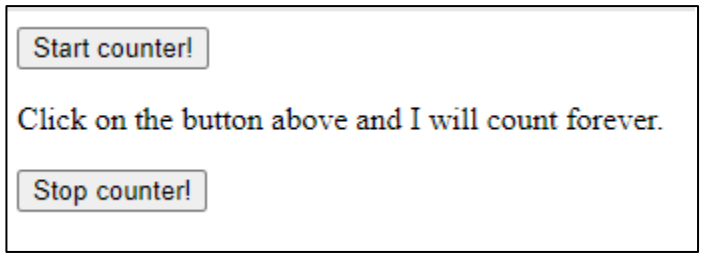


Timing event in an infinite loop

```
<html>
<body>
<button onClick="setInterval(myCounter, 1000)">Start counter!</button>
<p id="demo">Click on the button above and I will count forever.</p>
<script>
var c = 0;
function myCounter() {
  document.getElementById("demo").innerHTML = ++c;
}
</script>
</body>
</html>
```

Timing event in an infinite loop-with stop button

```
<html>
<body>
<button onClick="myTimer = setInterval(myCounter, 1000)">Start
counter!</button>
<p id="demo">Click on the button above and I will count forever.</p>
<button onClick="clearInterval(myTimer)">Stop counter!</button>
<script>
var c = 0;
function myCounter() {
  document.getElementById("demo").innerHTML = ++c;
}
</script>
</body>
</html>
```



The setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

```
window.setInterval(function, milliseconds);
```

- The window.setInterval() method can be written without the window prefix.
- The first parameter is the function to be executed.
- The second parameter indicates the length of the time-interval between each execution.
- This example executes a function called "myTimer" once every second (like a digital watch).

```
<html>
<body>
<h2>JavaScript Timing</h2>
<p>A script on this page starts this clock:</p>
<p id="demo"></p>
<script>
setInterval(myTimer, 1000);
function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
</body>
</html>
```

toLocaleTimeString():The toLocaleTimeString() method returns the time portion of a date object as a string, using locale conventions.

Display "Hello" every second (1000 milliseconds):

```
<html>
<body>
<h1>The Window Object</h1>
<h2>The setInterval() Method</h2>
<p id="demo"></p>
<script>
setInterval(displayHello, 1000);
function displayHello() {
  document.getElementById("demo").innerHTML += "Hello";
}
</script>
</body>
</html>
```

The Window History Object

- The history object contains the URLs visited by the user (in the browser window).
- The history object is a property of the window object.
- The JS history object contains an array of URLs visited by the user. By using the history object, you can load previous, forward, or any particular page using various methods.
- The history object is accessed with:

window.history or just history

History Object Properties and Methods

Property/Method	Description
back()	Loads the previous URL (page) in the history list
forward()	Loads the next URL (page) in the history list
go()	Loads a specific URL (page) from the history list
length	Returns the number of URLs (pages) in the history list

Windows history.length:

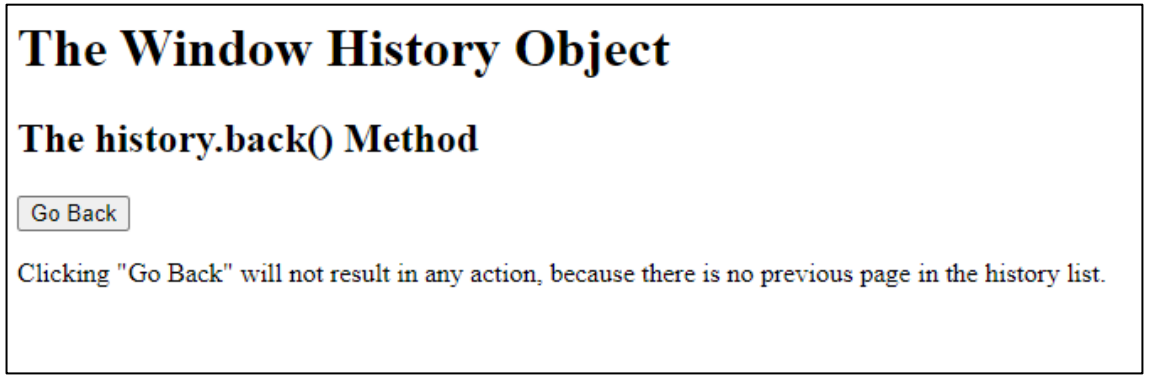
Get the number of URLs in the history list:

```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.length Property</h2>

<p>Number of URLs in history list:</p>
<p id="demo"></p>

<p>Since this is a new window frame, history.length will always return 1.</p>

<script>
let length = history.length;
document.getElementById("demo").innerHTML = length;
</script>

</body>
</html>
```

Window history.back()

- The history.back() method loads the previous URL (page) in the history list.
- The history.back() method only works if a previous page exists.

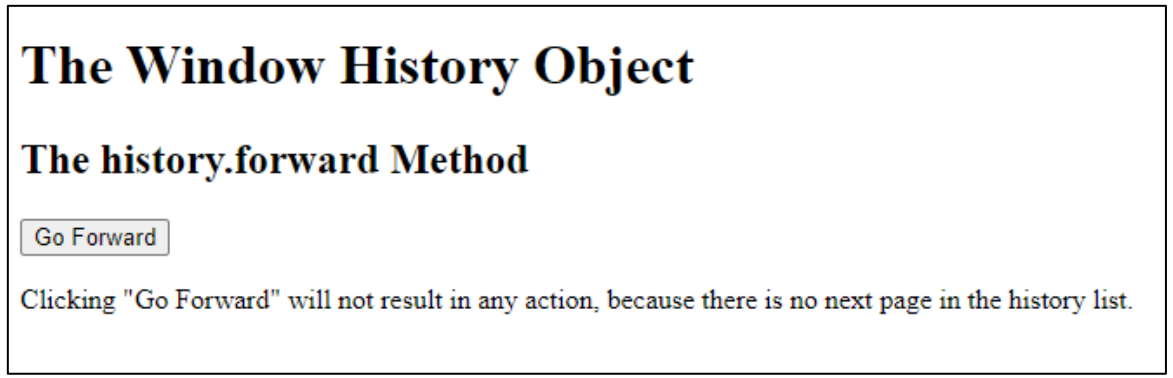
```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.back() Method</h2>

<button onclick="history.back()">Go Back</button>

<p>Clicking "Go Back" will not result in any action, because there is no previous page in the history list.</p>

</body>
</html>
```



Window history.forward()

- The history.forward() method loads the next URL (page) in the history list.
- The history.forward() method only works if a next page exists.

```
<html>
<body>

<h1>The Window History Object</h1>
<h2>The history.forward Method</h2>

<button onclick="history.forward()">Go Forward</button>

<p>Clicking "Go Forward" will not result in any action, because there is no next page in the history list.</p>

</body>
</html>
```



Window history.go()

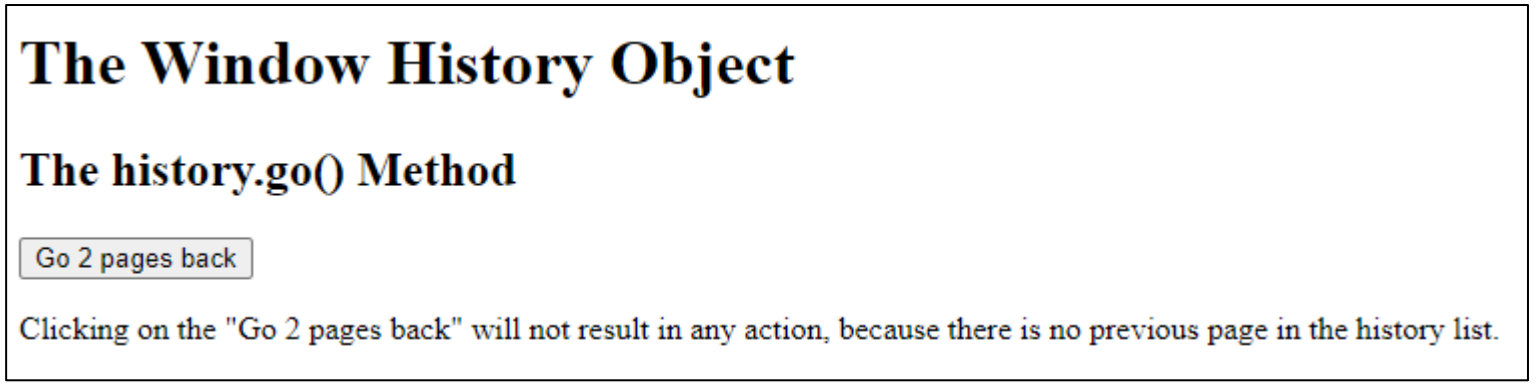
- The history.go() method loads a URL (page) from the history list.
- The history.go() method only works if the page exist in the history list.

```
<html>
<body>
<h1>The Window History Object</h1>
<h2>The history.go() Method</h2>
<button onclick="history.go(-2)">Go 2 pages back</button>
<p>Clicking on the "Go 2 pages back" will not result in any action, because there is no previous page in the history list.</p>
</body>
</html>
```

history.go(0) reloads the page.

history.go(-1) is the same as history.back().

history.go(1) is the same as history.forward().



Chapter 5-Frames

Frames:

- You may have visited web sites in which you were able to scroll the main portion of the web page while a smaller section containing navigation remained stationary on the screen. Although this looked as though it were all contained on a single web page, actually multiple web pages appeared on the screen at the same time, and each was displayed in a frame.

- Frames are created using HTML, but you can interact and manipulate frames using a JavaScript. You'll see how this is done in this chapter.

- All frames contain at least three web pages. The first frame surrounds the other frames, and this entire collection is called the *frameset*. The other frames are within the frameset, and each is referred to as a *child*. You can give each child a unique name so you can later refer to it in your application.

- JavaScript refers to the frameset as the *top* or the *parent*. The parent frame is al- ways at the top of the display. Child windows appear within the parent window. You can nest frames many layers deep—so the top level may actually still be a child frame of another frameset.

Example1:

```
<html>
<head>
<title>Create a Frame</title>
</head>

<frameset rows="50%,50%">

<frame src="WebPage1.html" name="topPage"/>
<frame src="WebPage2.html" name="bottomPage"/>

</frameset>

</html>
```

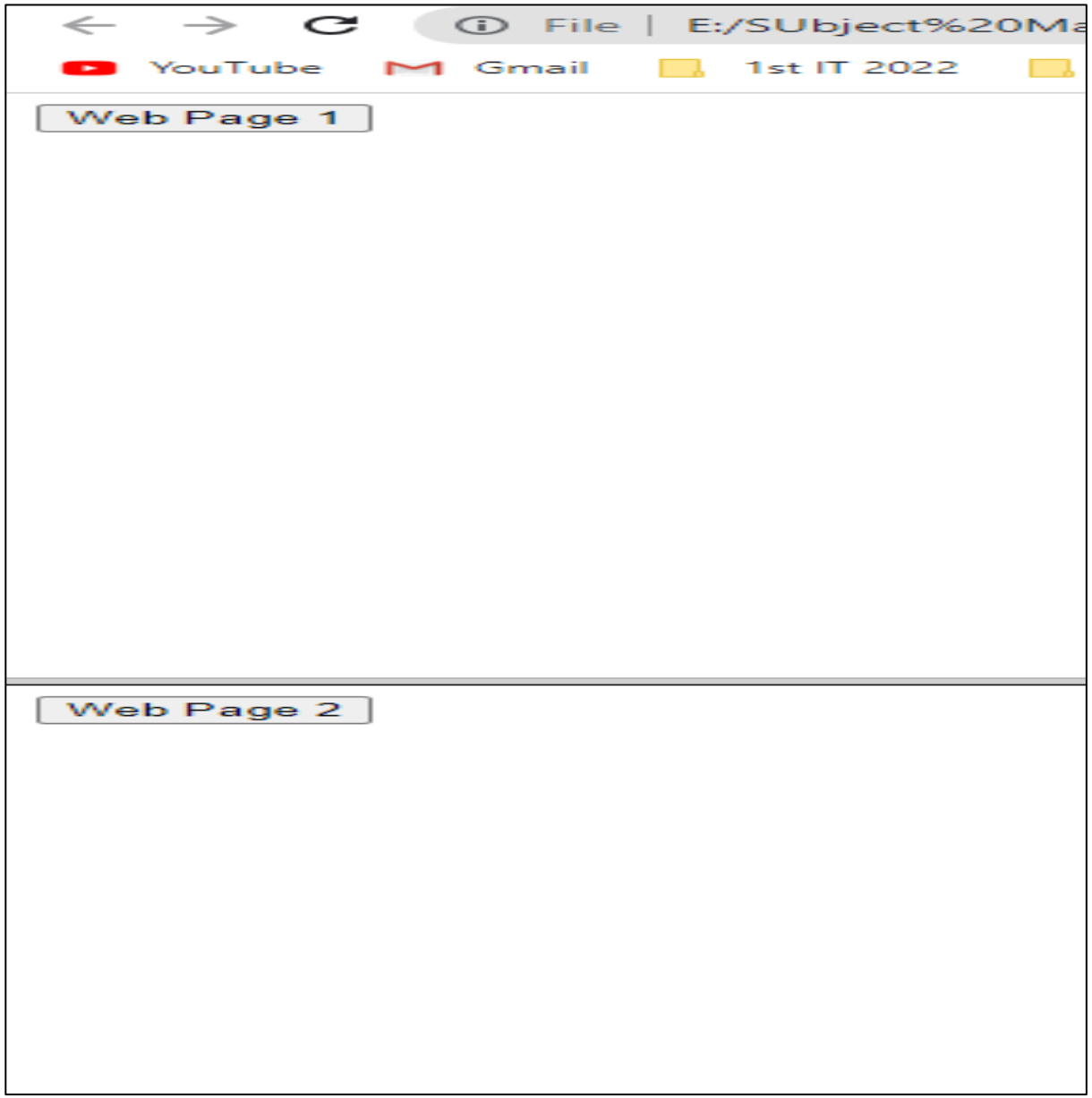
WebPage1.html

```
<html>
<head>
<title>Web Page 1</title>
</head>
<body>
<FORM action="http://abc.com" method="post">

<P>

<INPUT name="WebPage1" value="Web Page 1"
type="button" />

</P>
</FORM>
</body>
</html>
```

WebPage2.html

```
<html>
<head>
<title>Web Page 2</title>
</head>
<body>
<FORM action="http://xyz.com" method="post">

<P>

<INPUT name="WebPage2" value="Web Page 2"
type="button" />

</P>
</FORM>
</body>
</html>
```

OUTPUT

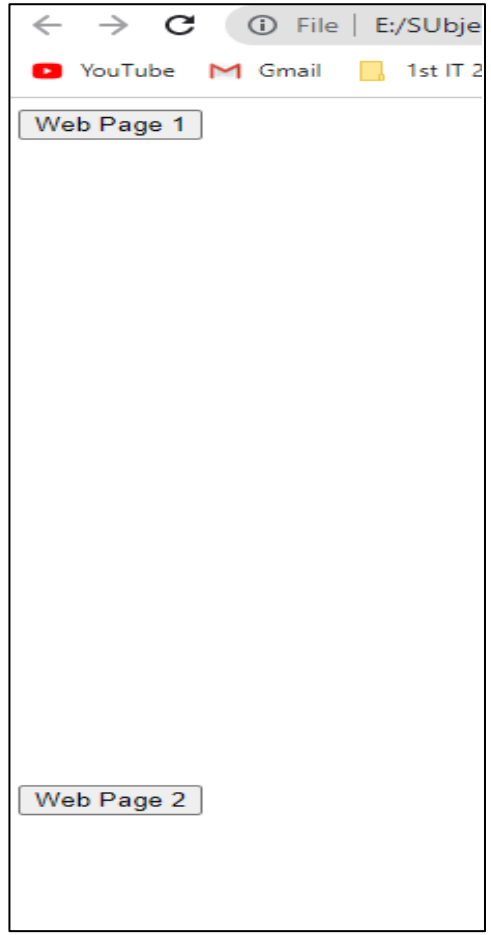

File | E:/SUbject%20Ma
YouTube
Gmail
1st IT 2022
Web Page 1
Web Page 2

Invisible Borders

- You can make it less obvious that you are using frames by hiding the borders around the child windows within your frameset. The result appears as one web page on the screen, even though in reality each of multiple web pages appears in its own child window.

- The border can be hidden by setting the frameborder and border attributes of the <frame> tag to zero (0).

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="50%,50%">
<frame src="WebPage1.html" name="topPage" frameborder="0" border="0" />
<frame src="WebPage2.html" name="bottomPage" frameborder="0" border="0" />
</frameset>
</html>
```



•border: This attribute of frameset tag defines the width of the border of each frame in pixels. Zero value is used for no border.

•frameborder: This attribute of frameset tag is used to specify whether a three-dimensional border should be displayed between the frames or not for this use two values 0 and 1, where 0 defines no border and value 1 signifies for yes there will be a border.

Calling a Child Window's JavaScript Function

WebPage1.html

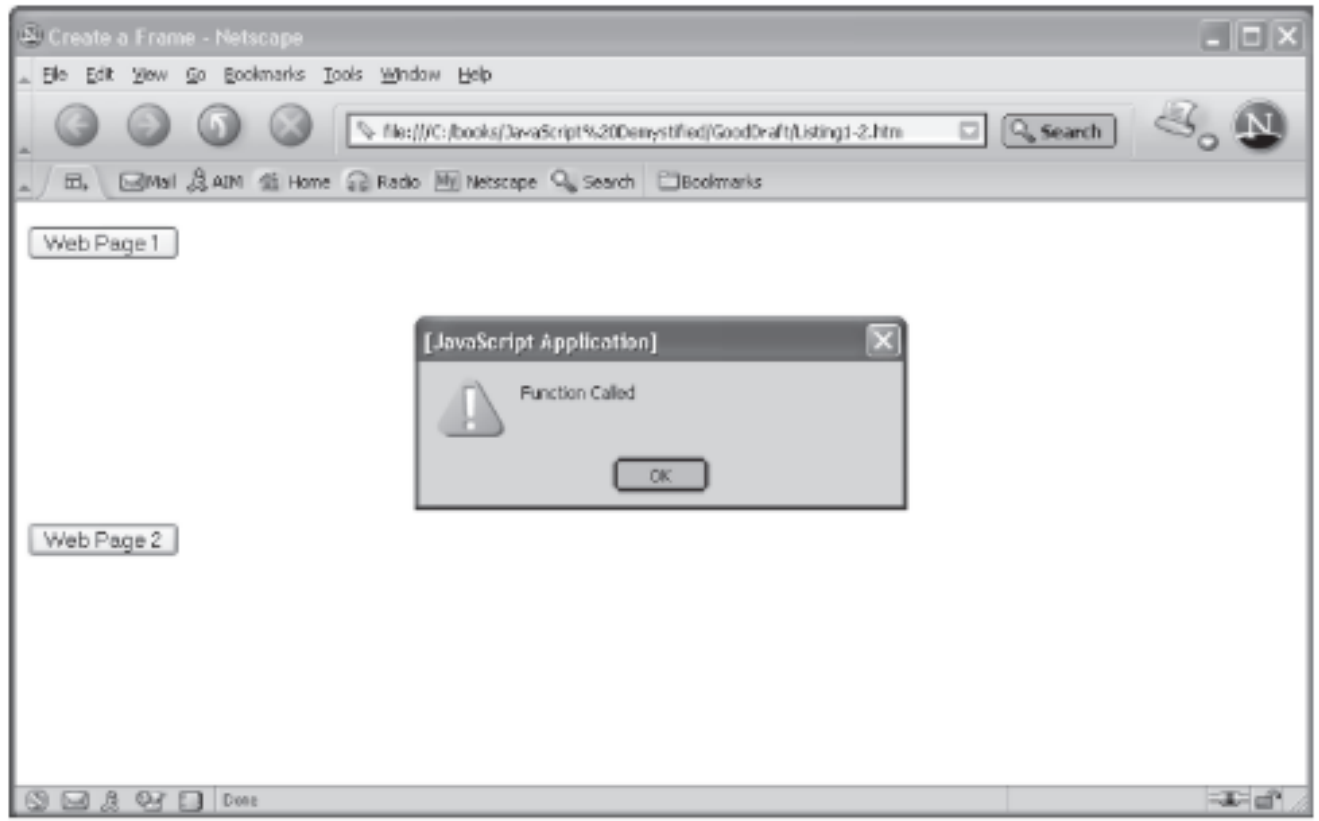
```
<html>
<head>
<title>Web Page 1</title>
<script>
<!--
function ChangeContent()
{
alert("Function Called");
}
-->
</script>

</head>
<body>
<FORM action="http://www.jimkeogh.com"
method="post">
<P>
<INPUT name="WebPage1" value="Web
Page 1" type="button" />
</P>
</FORM>
</body>
</html>
```

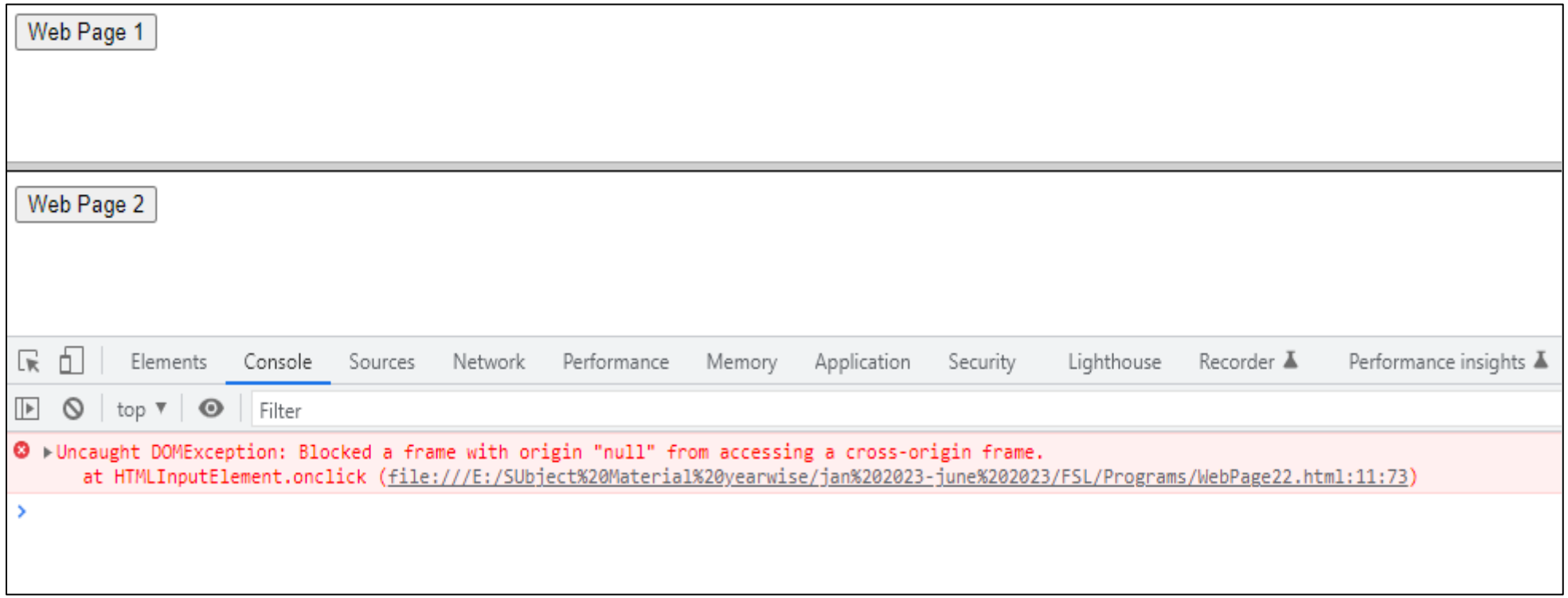
WebPage2.html

```
<html>
<head>
<title>Web Page 2</title>
</head>
<body>
<FORM action="http://www.jimkeogh.com"
method="post">
<P>

<INPUT name="WebPage2"
value="Web Page 2"
type="button"
onclick="parent.topPage.ChangeContent()"
/>

</P>
</FORM>
</body>
</html>
```

Frame.html

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="50%,50%">

<frame src="WebPage1.html"
name="topPage"/>

<frame src="WebPage2.html"
name="bottomPage"/>

</frameset>
</html>
```


Create a Frame - Netscape
File Edit View Go Bookmarks Tools Window Help
file:///C:/books/JavaScript%20Demystified/GoodDraft/Listing1-2.htm
Search
Mail AIM Home Radio Netscape Search Bookmarks
Web Page 1
[JavaScript Application]
Function Called
OK
Web Page 2
Done


Web Page 1
Web Page 2
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
top
Filter
Uncaught DOMException: Blocked a frame with origin "null" from accessing a cross-origin frame.
at HTMLInputElement.onclick (file:///E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/WebPage22.html:11:73)

Changing the Content of a Child Window

You can change the content of a child window from a JavaScript function by modifying the source web page for the child window. To do this, you must assign the new source to the child window's href attribute.

WebPage1.html

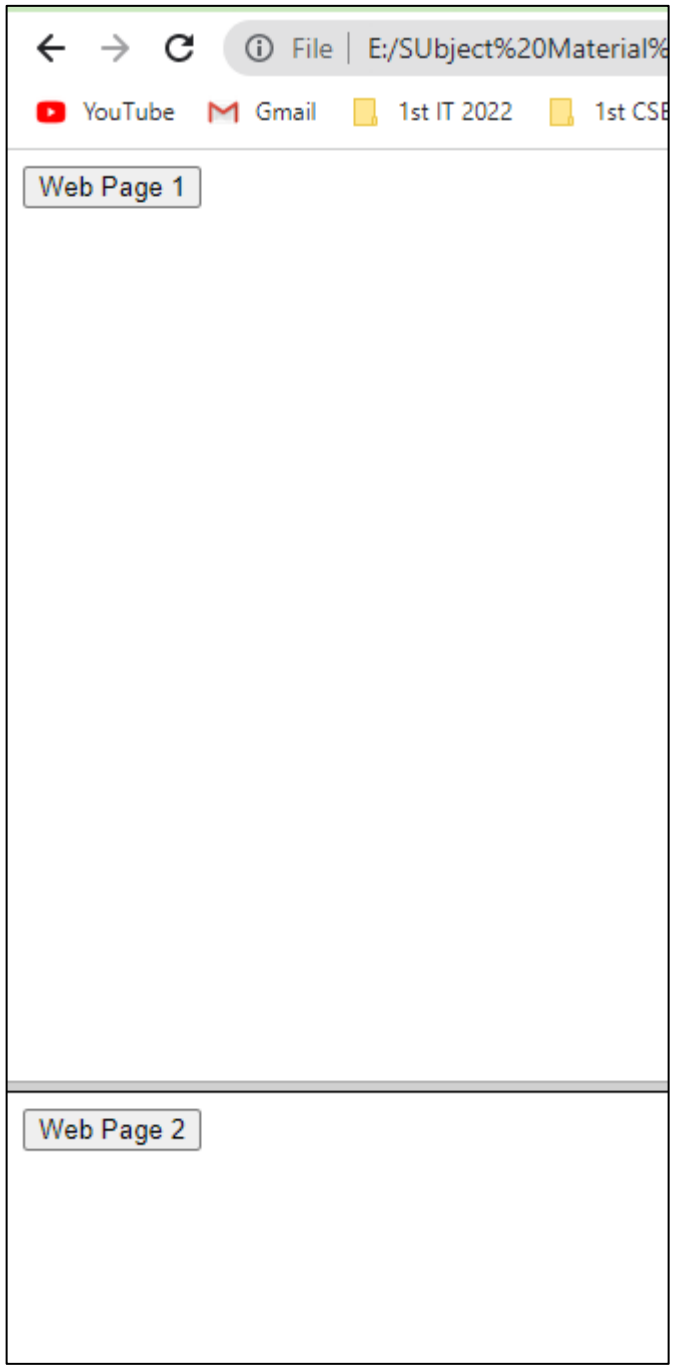
```
<html>
<head>
<title>Web Page 1</title>
<script language="Javascript" type="text/javascript">
<!--
function ChangeContent()
{
parent.topPage.location.href='WebPage3.html'
}
-->
</script>
</head>
<body>
<FORM action="http://www.jimkeogh.com"
method="post">
<P>
<INPUT name="WebPage1" value="Web Page 1"
type="button" onclick="ChangeContent()"/>
</P>
</FORM>
</body></html>
```

WebPage2.html

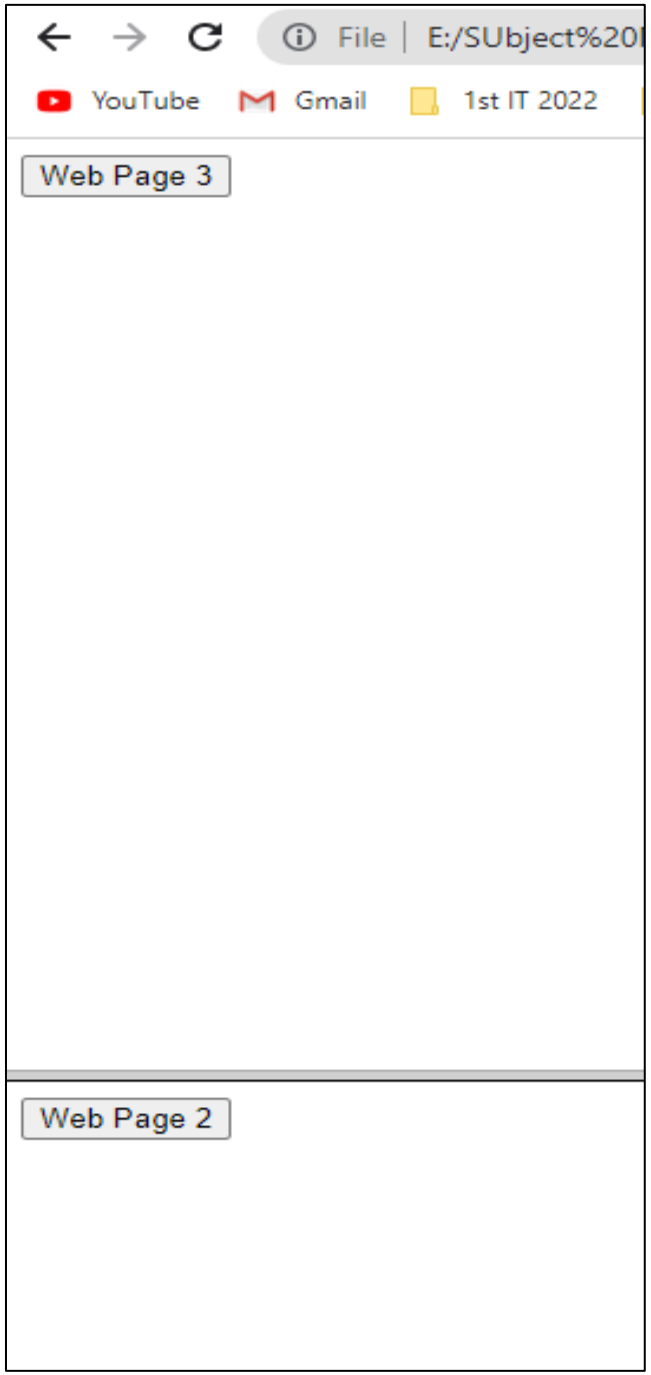
```
<html>
<head>
<title>Web Page 2</title>
</head>
<body>
<FORM
action="http://www.jimkeogh.co
m" method="post">
<P>
<INPUT name="WebPage2"
value="Web Page 2"
type="button" />
</P>
</FORM>
</body>
</html>
```

WebPage3.html

```
<html>
<head>
<title>Web Page 3</title>
</head>
<body>
<FORM
action="http://www.jimkeog
h.com" method="post">
<P>

<INPUT name="WebPage3"
value="Web Page 3"
type="button" />

</P>
</FORM>
</body>
</html>
```



Click on **Web Page1**



Writing to a Child Window from a JavaScript

Typically, the content of a child window is a web page that exists on the web server.However, you can dynamically create the content when you define the frameset bydirectly writing to the child window from a JavaScript. The JavaScript must be de-fined in the HTML file that defines the frameset and called when the frameset is loaded.

```
<html><head>
<title>Create a Frame</title>
<script language="Javascript" type="text/javascript">
<!--
function ChangeContent()
{
window.topPage.document.open()
window.topPage.document.writeln(
'<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN">')
window.topPage.document.writeln(
'<html>')
window.topPage.document.writeln('<head>')
window.topPage.document.writeln(
'<title>Web Page 3</title>')
window.topPage.document.writeln('</head>')
window.topPage.document.writeln('<body>')
window.topPage.document.writeln(
'<FORM action="http://www.jimkeogh.com" method="post">')
window.topPage.document.writeln('<P>')
```

---------- The writeln() method writes directly to an open (HTML) document stream.

```
window.topPage.document.writeln(
'<INPUT name="WebPage3" value="Web Page 3"
type="button" />')
window.topPage.document.writeln('</P>')
window.topPage.document.writeln('</FORM>')
window.topPage.document.writeln('</body>')
window.topPage.document.writeln('</html>')
window.topPage.document.close()
}
-->
</script>
</head>
<frameset rows="50%,50%" onload="ChangeContent()">
<frame src="WebPage1.html" name="topPage" />
<frame src="WebPage2.html" name="bottomPage" />
</frameset>
</html>
```

- To write dynamic content to a child window, you must assign a source file to each frame of the frameset, even though you are dynamically creating the source for at least one of those frames. You'll notice in this example that WebPage1.html is assigned to the topPage frame. WebPage1.html must be a real file, although it won't appear in the topPage frame because the JavaScript function writes the content to that frame.

- The JavaScript function is defined in the <head> tag and is called when the onload event occurs. The topPage child window must be opened before the Ja- vaScript function can write to the window. You open the child window by calling the open()method for that frame, as shown here:

```
window.topPage.document.open()
```

- Once opened, call the write() method to write HTML content to the child window to create the web page. This example displays the Web Page 3 button on aform. The final step is to call the close()method to close the window, as shownhere:

```
window.topPage.document.close()
```

RollOvers in Javascript

RollOvers

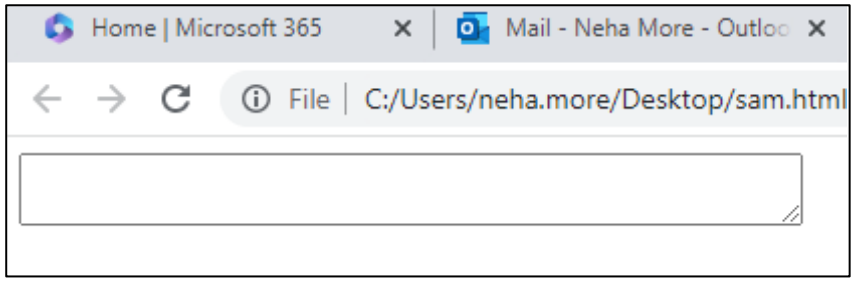
Rollover means a webpage changes when the user moves his or her mouse over an object on the page. It is often used in advertising.There are two ways to create rollover, using plain HTML or using a mixture of JavaScript and HTML.

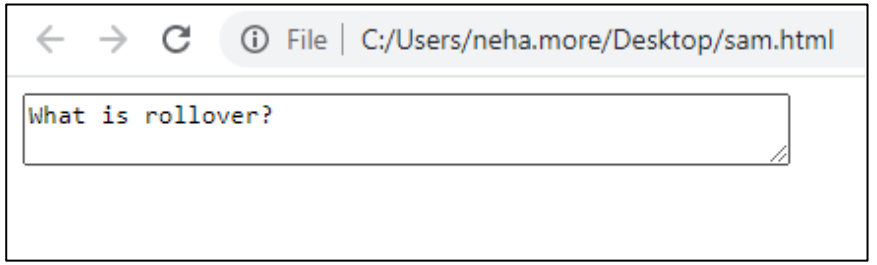
Creating Rollovers using HTML

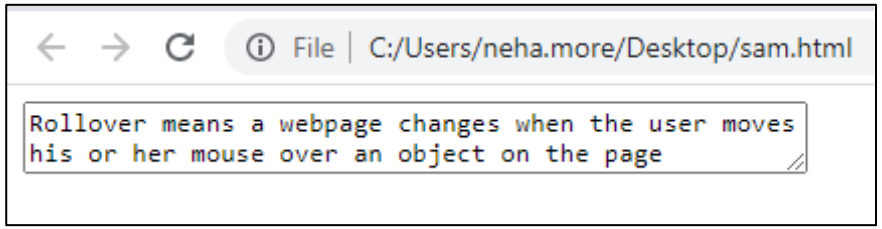
The keyword that is used to create rollover is the <onmousover> event. For example, we want to create a rollover text that appears in a text area. The text *"What is rollover?"* appears when the user place his or her mouse over the text area and the rollover text changes to *"Rollover means a webpage changes when the user moves his or her mouse over an object on the page"* when the user moves his or her mouse away from the text area.

EXAMPLE

```
<HTML>
<head></head>
<Body>
<textarea rows="2" cols="50" name="rollovertext"
onmouseover="this.value='What is rollover?'"
onmouseout="this.value='Rollover means a webpage changes
when the user moves his or her mouse over an object on the
page'"></textarea>
</body>
</html>
```

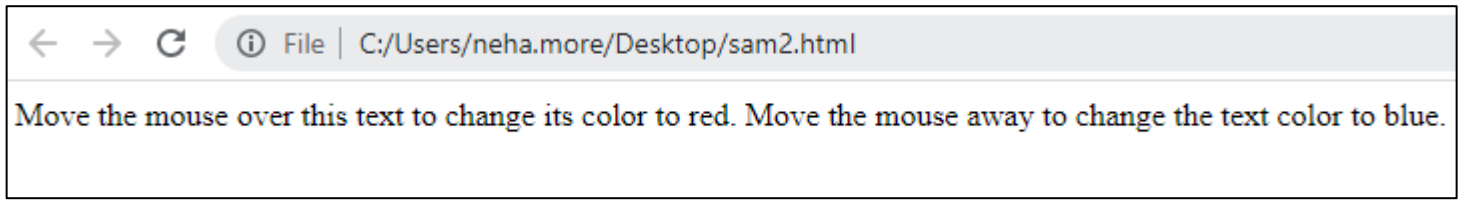


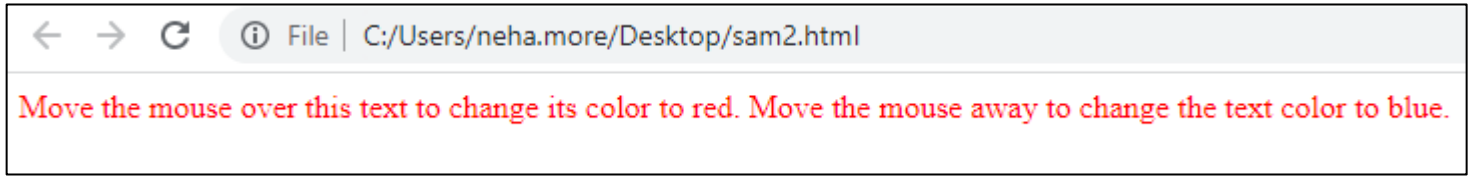


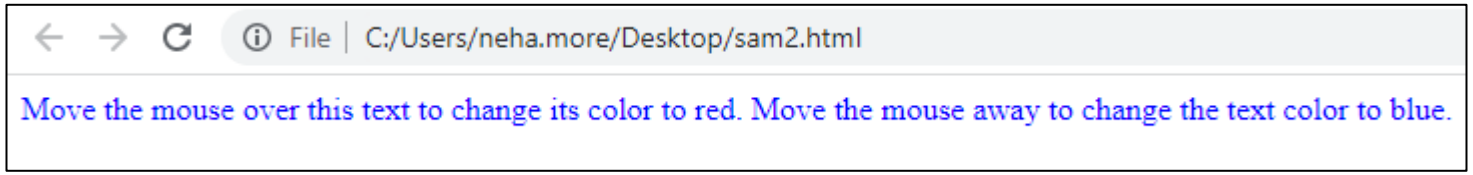


create a rollover effect that can change the color of its text using the style attribute.

```
<html>
<body>
<p
onmouseover="this.style.color='red'"
onmouseout="this.style.color='blue'">
Move the mouse over this text to change its color to red. Move
the mouse away to
change the text color to blue.
</p>
</body>
</html>
```

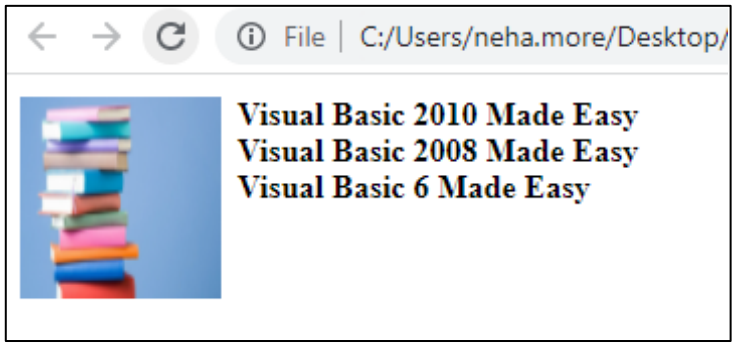


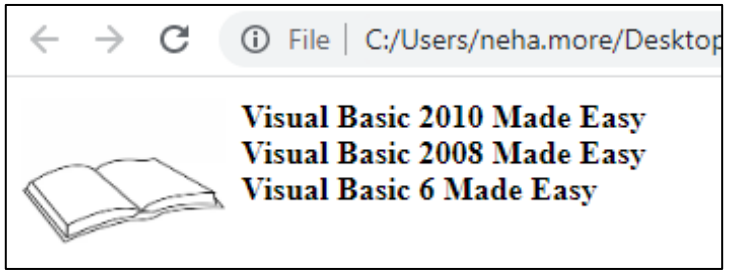


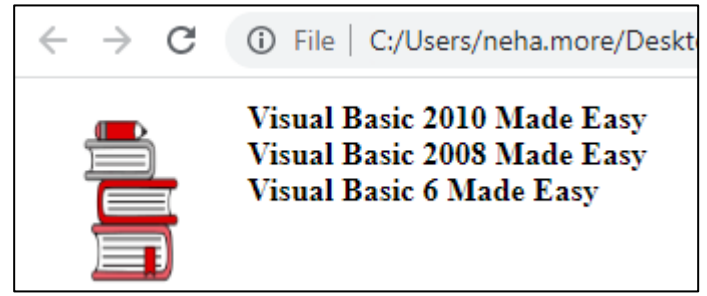


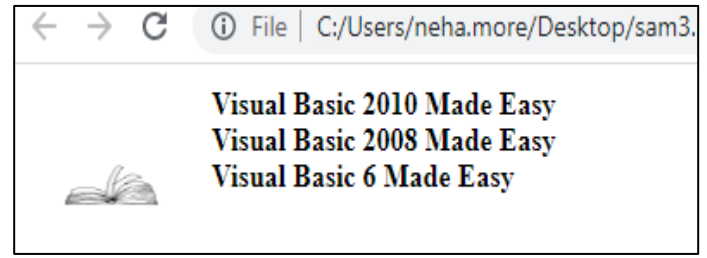
Example 3: To create rollover effect that involves text and images. When the user places his or her mouse pointer over a book title, the corresponding book image appears.

```
<html>
<head>
<title>Rollover Effect</title>
</head>
<body>
<table>
<tbody>
<tr valign="top">
<td width="50">
<a><img src="abc.jpg" name="book" height="100" width="100"></a>
</td><td></td>
<td><a onmouseover="document.book.src='a.png'"><b>Visual Basic 2010 Made
Easy</b></a>
<br>
<a onmouseover="document.book.src='b.png'"><b>Visual Basic 2008 Made Easy</b></a>
<br>
<a onmouseover="document.book.src='c.png'"><b>Visual Basic 6 Made Easy</b></a>
<br>
</td>
</tr>
</tbody>
</table>
</body>
</html>
```









Creating Rollovers Using JavaScript

Though HTML can be used to create rollovers , it can only performs simple actions. If you wish to create more powerful rollovers, you need to use JavaScript. To create rollovers in JavaScript, we need to create a JavaScript function.

Example 4 is similar to Example 3 but we have added some JavaScript code. In this example, we have create an array MyBooks to store the images of three book covers. Next, we create a ShowCover(book) to display the book cover images on the page. Finally, we call the ShowCover function using the onmouseover event.

Example:4

```
<html>
<head>
<script language="Javascript">
MyBooks=new Array('c.png','a.png','b.png')
book=0
function ShowCover(book){document.DisplayBook.src=MyBooks[book]
}</script></head>
<body>
<body>
<P align="center"><img src="abc.jpg" name="DisplayBook" height="50" width="50"><p>
<center>
<table border=0>
<tr>
<td align=center><a onmouseover="ShowCover(0)"><b>Visual Basic 2010 Made
Easy</b></a><br>
<a onmouseover="ShowCover(1)"><b>Visual Basic 2008 Made Easy</b></a><br>
<a onmouseover="ShowCover(2)"><b>Visual Basic 6 Made Easy</b></a><br>
</td>
</tr>
</table>
</body>
</html>
```

Visual Basic 2010 Made Easy
Visual Basic 2008 Made Easy
Visual Basic 6 Made Easy

Visual Basic 2010 Made Easy
Visual Basic 2008 Made Easy
Visual Basic 6 Made Easy

Visual Basic 2010 Made Easy
Visual Basic 2008 Made Easy
Visual Basic 6 Made Easy

Visual Basic 2010 Made Easy
Visual Basic 2008 Made Easy
Visual Basic 6 Made Easy

Regular Expression

RegExp Object

- A regular expression is a **pattern** of characters.
- The pattern is used to do pattern-matching **"search-and-replace"** functions on text.
- In JavaScript, a **RegExp Object** is a pattern with **Properties** and **Methods**.

Syntax

```
/pattern/modifier(s);
```

Example 1

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Do a case-insensitive search for "w3schools" in a string:</p>
<p id="demo"></p>
<script>
let text = "Visit W3Schools";
let pattern = /w3schools/i;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

W3schools : The pattern to search for
/w3schools/ : A regular expression
/w3schools/i : A case-insensitive regular expression

Modifiers:

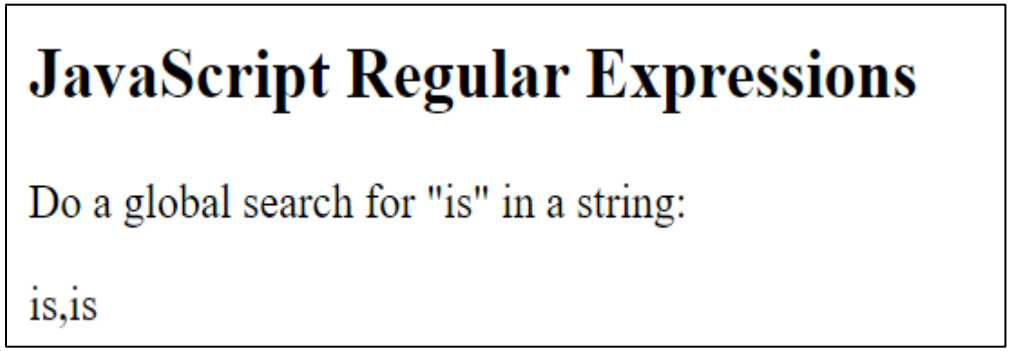
Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

JavaScript RegExp g Modifier:

The "g" modifier specifies a global match.
A global match finds all matches.

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>Do a global search for "is" in a string:</p>

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

<script>
let text = "Is this all there is?";
let pattern = /is/g;
let result = text.match(pattern);

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

</body>
</html>
```



JavaScript RegExp i Modifier:

The "i" modifier specifies a case-insenitive match.

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>The i modifier performs a case-insenitive match:</p>

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

<script>
let text = "Visit W3Schools";
let pattern = /w3schools/i;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions

The i modifier performs a case-insenitive match:
W3Schools

JavaScript RegExp m Modifier

- The "m" modifier specifies a **multiline match**.
- It only affects the behavior of start **^** and end **$**.
- **^** specifies a match at the start of a string.
- **$** specifies a match at the end of a string.
- With the "m" set, ^ and $ also match at the beginning and end of each line.

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Do a multiline search for "is" at the beginning of each line
in a string:</p>
<p id="demo"></p>
<script>
let text = `Is this
all there
is`
let pattern = /^is/m;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions

Do a multiline search for "is" at the beginning of each line in a string:

is

The "m" modifier is case-sensitive and not global.
To perform a global, case-insensitive search, use "m" with "g" and "i".

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>

<p>Do a global, multiline search for "is" at the beginning of
each line in a string.</p>

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

<script>
let text = `Is this
all there
is`

let pattern = /^is/gm;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
Do a global, multiline search for "is" at the beginning of each line in a string.
is

A global, case-insensitive, multiline search for "is" at the beginning of each string line:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>Do a global, case-insensitive, multiline search for "is" at the beginning of each line in a string.</p>

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

<script>
let text = `Is this
all there
is`

let pattern = /^is/gmi;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

JavaScript Regular Expressions

Do a global, case-insensitive, multiline search for "is" at the beginning of each line in a string.

Is,is

A global, multiline search for "is" at the end of each string line

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>

<p>Do a global, multiline search for "is" at the end of each line
in a string.</p>

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

<script>
let text = `Is this
all there
is`

let pattern = /is$/gm;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
Do a global, multiline search for "is" at the end of each line in a string.

is,is

Regular Expression Search Methods:

In JavaScript, a regular expression text search, can be done with different methods.
With a **pattern** as a regular expression, these are the most common methods:

Example	Description
text.match(pattern)	The String method match()
text.search(pattern)	The String method search()
pattern.exec(text)	The RexExp method exec()
pattern.test(text)	The RegExp method test()

JavaScript String match()

- The match() method matches a string against a regular expression **
- The match() method returns an array with the matches.
- The match() method returns null if no match is found.

Syntax:

```
string.match(match)
```

Parameters:

Parameter	Description
match	Required. The search value. A regular expression (or a string that will be converted to a regular expression).

Return Values

Type	Description
An array or null	An array containing the matches. null if no match is found.

A search for "ain" using a string:

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The match() Method</h2>

<p>match() searches for a match in a string.</p>

<p>Do a search for "ain":</p>

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

<script>
let text = "The rain in SPAIN stays mainly in the plain";
let result = text.match("ain");

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

</body>
</html>
```

JavaScript Strings
The match() Method
match() searches for a match in a string.

Do a search for "ain":

ain

A search for "ain" using a regular expression:

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The match() Method</h2>

<p>match() searches for a match against a regular
expression.</p>

<p>Do a search for "ain":</p>

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

<script>
let text = "The rain in SPAIN stays mainly in the plain";
let result = text.match(/ain/);

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

</body>
</html>
```

JavaScript Strings
The match() Method
match() searches for a match against a regular expression.

Do a search for "ain":

ain

JavaScript String search():

- The search() method matches a string against a regular expression **
- The search() method returns the index (position) of the first match.
- The search() method returns -1 if no match is found.
- The search() method is case sensitive.

Syntax

```
string.search(searchValue)
```

Parameters

Parameter	Description
searchValue	Required. The search value. A regular expression (or a string that will be converted to a regular expression).

Return Value

Type	Description
A number	The position of the first match. -1 if no match.

Example:

Search for "Blue":

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The search() Method</h2>

<p>search() searches a string for a value and returns the
position of the first match:</p>

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

<script>
let text = "Mr. Blue has a blue house"
let position = text.search("Blue");

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

</body>
</html>
```

JavaScript Strings
The search() Method
search() searches a string for a value and returns the
position of the first match:

4

Search for "blue":

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The search() Method</h2>

<p>search() searches a string for a value and returns the
position of the first match:</p>

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

<script>
let text = "Mr. Blue has a blue house"
let position = text.search("blue");

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

</body>
</html>
```

JavaScript Strings
The search() Method
search() searches a string for a value and returns the position of the first match:

15

Search for /Blue/:

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The search() Method</h2>

<p>search() searches a string for a value and returns the
position of the first match:</p>

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

<script>
let text = "Mr. Blue has a blue house"
let position = text.search(/Blue/);

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

</body>
</html>
```

JavaScript Strings
The search() Method
search() searches a string for a value and returns the position of the first match:

4

Search for /blue/:

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The search() Method</h2>

<p>search() searches a string for a value and returns the
position of the first match:</p>

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

<script>
let text = "Mr. Blue has a blue house"
let position = text.search(/blue/);

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

</body>
</html>
```

JavaScript Strings
The search() Method
search() searches a string for a value and returns the position of the first match:

15

Search case insensitive:

```
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The search() Method</h2>

<p>search() searches a string for a value and returns the
position of first the match:</p>

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

<script>
let text = "Mr. Blue has a blue house"
let position = text.search(/blue/i);

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

</body>
</html>
```

JavaScript Strings
The search() Method
search() searches a string for a value and returns the position of first the match:

4

The Difference Between

String search() and String match()

The search() method returns the position of the first match.

The match() method returns an array of matches.

JavaScript RegExp exec()

- The exec() method tests for a match in a string.
- If it finds a match, it returns a result array, otherwise it returns null.

Syntax

```
RegExpObject.exec(string)
```

Parameter Values

Parameter	Description
string	Required. The string to be searched

Return Value

Type	Description
Array	An array containing the matched text if it finds a match, otherwise it returns null

Example: Search a string for the character "e":

```
<html>
<body>

<h2>JavaScript RegExp</h2>

<p>The exec() method tests for a match in a string:</p>

<p>Search a string for the character "e":</p>

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

<script>
let text = "The best things in life are free";
let result = /e/.exec(text);
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

JavaScript RegExp

The exec() method tests for a match in a string:

Search a string for the character "e":

e

Example:
Do a global search for "Hello" and "W3Schools" in a string:

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>The exec() method tests for a match in a string:</p>

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

<script>
let text = "Hello world!";

// look for "Hello"
let result1 = /Hello/.exec(text);

// look for "W3Schools"
let result2 = /W3Schools/.exec(text);

document.getElementById("demo").innerHTML =
result1 + "<br>" + result2;
</script>
</body>
</html>
```

JavaScript Regular Expressions
The exec() method tests for a match in a string:

Hello
null

JavaScript RegExp test():

- The test() method tests for a match in a string.
- If it finds a match, it returns true, otherwise it returns false.

Syntax

```
RegExpObject.test(string)
```

Parameter Values

Parameter	Description
string	Required. The string to be searched

Return Value

Type	Description
Boolean	Returns true if it finds a match, otherwise false

Example 1

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>The test() method returns true if it finds a match, otherwise
false.</p>

<p>Search a string for the character "e":</p>
<p id="demo"></p>

<script>
let text = "The best things in life are free";
let pattern = /e/;
let result = pattern.test(text);

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

</body>
</html>
```

JavaScript Regular Expressions
The test() method returns true if it finds a match, otherwise false.

Search a string for the character "e":

true

Example 2

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Do a global search for "Hello" and "W3Schools" in a string:</p>
<p id="demo"></p>

<script>
let text = "Hello world!";

// look for "Hello"
let pattern1 = /Hello/g;
let result1 = pattern1.test(text)

// look for "W3Schools"
let pattern2 = /W3Schools/g;
let result2 = pattern2.test(text);

document.getElementById("demo").innerHTML = result1 + "<br>" + result2;
</script>
</body>
</html>
```

JavaScript Regular Expressions

Do a global search for "Hello" and "W3Schools" in a string:

true
false

JavaScript RegExp Group [abc]

- Brackets [abc] specifies matches for the characters inside the brackets.
- Brackets can define single characters, groups, or character spans:

[abc]	Any of the characters a, b, or c
[A-Z]	Any character from uppercase A to uppercase Z
[a-z]	Any character from lowercase a to lowercase z
[A-z]	Any character from uppercase A to lowercase z

Example:

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>A global search for the character "h" in a string:</p>
<p id="demo"></p>
<script>
let text = "Is this all there is?";
let pattern = /[h]/g;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for the character "h" in a string:

h,h

Example

Global search for the characters "i" and "s" in a string:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for the characters "i" and "s" in a string:</p>

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

<script>
let text = "Do you know if this is all there is?";
let pattern = /[is]/gi;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global search for the characters "i" and "s" in a string:

i,i,s,i,s,i,s

Example

A global search for the character span from lowercase "a" to lowercase "h" in a string:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for the character span [a-h] in a string:</p>

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

<script>
let text = "Is this all there is?";
let pattern = /[a-h]/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global search for the character span [a-h] in a string:

h,a,h,e,e

Example

Do a global search for the character-span from uppercase "A" to uppercase "E":

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>
<p>A global search for any character from uppercase A to
uppercase E.</p>
<p id="demo"></p>
<script>
let text = "I SCREAM FOR ICE CREAM!";
let pattern = /[A-E]/g;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for any character from uppercase A to uppercase E.

C,E,A,C,E,C,E,A

Example

A global search for the character span from uppercase "A" to lowercase "e" (**will search for all uppercase letters, but only lowercase letters from a to e**.)

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for any character from uppercase "A" to lowercase "e":</p>

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

<script>
let text = "I Scream For Ice Cream, is that OK?!";
let pattern = /[A-e]/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global search for any character from uppercase "A" to lowercase "e":

I,S,c,e,a,F,I,c,e,C,e,a,a,O,K

Example

A global, case-insensitive search for the character span [a-s]:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global, case-insensitive search for the character span [a-s]:</p>

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

<script>
let text = "I Scream For Ice Cream, is that OK?!";
let pattern = /[a-s]/gi;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global, case-insensitive search for the character span [a-s]:

I,S,c,r,e,a,m,F,o,r,I,c,e,C,r,e,a,m,i,s,h,a,O,K

A "g" and "gi" search for characters:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>
<p>Perform a search for the characters "THIS" in a string.</p>
<p id="demo"></p>

<script>
let text = "THIS This this";

let result1 = text.match(/[THIS]/g);
let result2 = text.match(/[THIS]/gi);

document.getElementById("demo").innerHTML =
result1 + "<br>" +result2;
</script>
</body>
</html>
```

JavaScript Regular Expressions
Perform a search for the characters "THIS" in a string.

T,H,I,S,T
T,H,I,S,T,h,i,s,t,h,i,s

JavaScript RegExp Group [^abc]

Brackets [^abc] specifies matches for any character NOT between the brackets.

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>Do a global search for that characters that are not "h":</p>

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

<script>
let text = "Is this all there is?";
let pattern = /[^h]/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
Do a global search for that characters that are not "h":

I,s, ,t,i,s, ,a,l,l, ,t,e,r,e, ,i,s,?

Example

Do a global search for characters that are NOT "i" and "s" in a string:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for characters that are NOT "i" or "s":</p>

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

<script>
let text = "Do you know if this is all there is?";
let pattern = /[^is]/gi;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global search for characters that are NOT "i" or "s":

D,o, ,y,o,u, ,k,n,o,w, ,f, ,t,h, , ,a,l,l, ,t,h,e,r,e, ,?

Example:

Do a global search for the character-span NOT from lowercase "a" to lowercase "h" in a string:

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>

<p>A global search for characters NOT in the span from
lowercase "a" to lowercase "h":</p>

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

<script>
let text = "Is this all there is?";
let pattern = /[^a-h]/g;
let result = text.match(pattern);

document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for characters NOT in the span from
lowercase "a" to lowercase "h":

I,s, ,t,i,s, ,l,l, ,t,r, ,i,s,?

Example

Do a global search for the character-span NOT from uppercase "A" to uppercase "E":

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>A global search for characters NOT in the span, from
uppercase A to uppercase E:</p>

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

<script>
let text = "I SCREAM FOR ICE CREAM!";
let pattern = /[^A-E]/g;
let result = text.match(pattern);

document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for characters NOT in the span, from uppercase A to uppercase E:

I, ,S,R,M, ,F,O,R, ,I, ,R,M,!

Example

Do a global search for the character-span NOT from uppercase "A" to lowercase "e":

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for characters NOT in the span, from
uppercase "A" to lowercase "e":</p>

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

<script>
let text = "I Scream For Ice Cream, is that OK?!";
let pattern = /[^A-e]/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions
A global search for characters NOT in the span, from uppercase "A" to lowercase "e":

,r,m, ,o,r, , ,r,m,,, ,i,s, ,t,h,t, ,?,!

Example:

Do a global, case-insensitive search for the character-span that's NOT [a-s]:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global, case-insensitive search for the characters NOT in
the span [a-s]:</p>

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

<script>
let text = "I Scream For Ice Cream, is that OK?!";
let pattern = /[^a-s]/gi;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions

A global, case-insensitive search for the characters NOT in the span [a-s]:

, , , ,,, , ,t,t, ,?,!

JavaScript RegExp Group [0-9]

- The [0-9] expression is used to find any character between the brackets.
- The digits inside the brackets can be any numbers or span of numbers from 0 to 9.

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>

<p>A global search for the numbers 1 to 4:</p>

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

<script>
let text = "123456789";
let pattern = /[1-4]/g;
let result = text.match(pattern);

document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for the numbers 1 to 4:

1,2,3,4

JavaScript RegExp Group [^0-9]

A global search for numbers that are NOT from 1 to 4:

```
<html>
<body>

<h2>JavaScript Regular Expressions</h2>

<p>A global search for numbers that are NOT from 1 to 4:</p>

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

<script>
let text = "123456789";
let pattern = /[^1-4]/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions

A global search for numbers that are NOT from 1 to 4:

5,6,7,8,9

JavaScript RegExp Group (x|y)

- The (x|y) expression is used to find any of the alternatives specified.
- The alternatives can be of any characters.

A global search for any of the alternatives (red|green):

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>A global search for the specified alternatives
(red|green):</p>
<p id="demo"></p>
<script>

let text = "re, green, red, green, gren, gr, blue, yellow";
let pattern = /(red|green)/g;
let result = text.match(pattern);

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

</body>
</html>
```

JavaScript Regular Expressions

A global search for the specified alternatives (red|green):

green,red,green

Example

Global search to find any of the specified alternatives (0|5|7):

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>A global search for any of the specified alternatives
(0|5|7):</p>
<p id="demo"></p>

<script>
let text = "01234567890123456789";
let pattern = /(0|5|7)/g;
let result = text.match(pattern);

document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript Regular Expressions
A global search for any of the specified alternatives
(0|5|7):

0,5,7,0,5,7

Metacharacter

JavaScript RegExp . Metacharacter

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for "h.t":</p>

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

<script>
let text = "That's hot!";
let pattern = /h.t/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for "h.t":
hat,hot

```
let text = "Thaat's hot!";
let pattern = /h..t/g;
let result = text.match(pattern);
```

A global search for "h.t":
haat

JavaScript RegExp \w Metacharacter

- The \w metacharacter matches word characters.
- A word character is a character a-z, A-Z, 0-9, including _ (underscore).

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for word characters:</p>

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

<script>
let text = "Give 100%!";
let pattern = /\w/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for word characters:

G,i,v,e,1,0,0

JavaScript RegExp \W Metacharacter

- The \W metacharacter matches non-word characters:
- A word character is a character a-z, A-Z, 0-9, including _ (underscore).

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for non-word characters:</p>

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

<script>
let text = "Give 100%!";
let pattern = /\W/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for non-word characters:

,%,!

JavaScript RegExp \d Metacharacter

The \d metacharacter matches digits from 0 to 9.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for digits:</p>

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

<script>
let text = "Give 100%!";
let pattern = /\d/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for digits:

1,0,0

JavaScript RegExp \D Metacharacter

```
let text = "Give 100%!";
let pattern = /\D/g;
```

G,i,v,e, ,%,!

JavaScript RegExp \s Metacharacter

```
<h2>JavaScript Regular Expressions</h2>
<p>A global search for whitespace characters:</p>
<p id="demo"></p>

<script>
let text = "Is this all there is?";
let pattern = /\s/g;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions

A global search for whitespace characters:

, , ,

JavaScript RegExp \S Metacharacter

```
let text = "Is this all there is?";
let pattern = /\S/g;
```

I,s,t,h,i,s,a,l,l,t,h,e,r,e,i,s,?

JavaScript RegExp \b Metacharacter

The \b metacharacter matches at the beginning or end of a word.

Search for "LO" at the beginning of a word:

```
<h2>JavaScript Regular Expressions</h2>
<p>Search for the characters "LO" in the <b>beginning</b> of a word:</p>
<p>"HELLO, LOOK AT YOU!"</p>
<p id="demo"></p>
<script>
let text = "HELLO, LOOK AT YOU!";
let pattern = /\bLO/;
let result = text.search(pattern);
document.getElementById("demo").innerHTML = "Found in position: " + result;
</script>
```

JavaScript Regular Expressions
Search for the characters "LO" in the beginning of a word:

"HELLO, LOOK AT YOU!"

Found in position: 7

- Search for the pattern LO at the beginning of a word like this:

\bLO

- Search for the pattern LO at the end of a word like this:

LO\b

JavaScript RegExp \B Metacharacter

```
<h2>JavaScript Regular Expressions</h2>

<p>Search for the characters "LO" the phrase: "HELLO, LOOK AT YOU!" and return the first position where it is present, NOT in the <b>beginning</b> of a word:</p>

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

<script>
let text = "HELLO, LOOK AT YOU!";
let pattern = /\BLO/;
let result = text.search(pattern);

document.getElementById("demo").innerHTML = "Found in position: " + result;
</script>
```

JavaScript Regular Expressions

Search for the characters "LO" the phrase: "HELLO, LOOK AT YOU!" and return the first position where it is present, NOT in the **beginning** of a word:

Found in position: 3

The **\B** metacharacter matches NOT at the beginning/end of a word.

- Search for the pattern LO, not at the beginning of a word like this:

\BLO

- Search for the pattern LO, not at the end of a word like this:

LO\B

JavaScript RegExp \0 Metacharacter

The \0 metacharacter maches NUL characters.

```
<h2>JavaScript Regular Expressions</h2>

<p>Find the position where a NUL character is found:</p>

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

<script>
let text = "Visit W3Schools.\0Learn JavaScript.";
let pattern = /\0/;
let result = text.search(pattern);

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

JavaScript Regular Expressions

Find the position where a NUL character is found:

16

JavaScript RegExp \n Metacharacter

The \n character matches newline characters.

```
<h2>JavaScript Regular Expressions</h2>

<p>find the position where a newline character is found:.</p>

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

<script>
let text = "Visit W3Schools.\nLearn JavaScript.";
let pattern = /\n/;
let result = text.search(pattern);

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

JavaScript Regular Expressions
find the position where a newline character is found:.
16

Quantifier

JavaScript RegExp + Quantifier

The n+ quantifier matches any string that contains at least one n.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for at least one "o" in a string:</p>

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

<script>
let text = "Hellooo World! Hello W3Schools!";
let pattern = /o+/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions

A global search for at least one "o" in a string:

ooo,o,o,oo

JavaScript RegExp * Quantifier

The **n*** quantifier matches any string that contains zero or more occurrences of n.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for an "l", followed by zero or more "o"
characters:</p>

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

<script>
let text = "Hellooo World! Hello W3Schools!";
let pattern = /lo*/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for an "l", followed by zero or more "o" characters:

l,looo,l,l,lo,l

```
<script>
let text = "1, 100 or 1000?";
let pattern = /10*/g;
let result = text.match(pattern);

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

1,100,1000

JavaScript RegExp ? Quantifier

The n? quantifier matches any string that contains zero or one occurrences of n.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for a "1", followed by zero or one "0"
characters:</p>

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

<script>
let text = "1, 100 or 1000?";
let pattern = /10?/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions

A global search for a "1", followed by zero or one "0" characters:

1,10,10

JavaScript RegExp {X} Quantifier

- The n{X} quantifier matches any string that contains a sequence of X n's.
- X must be a number.

```
<h2>JavaScript Regular Expressions</h2>
<p>A global search for sequence of four digits:</p>
<p id="demo"></p>

<script>
let text = "100, 1000 or 10000?";
let pattern = /\d{4}/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global search for sequence of four digits:

1000,1000

JavaScript RegExp {X,Y} Quantifier

- The n{X,Y} quantifier matches any string that contains a sequence of X to Y n's.
- X and Y must be a number.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for a sequence of three to four digits:</p>

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

<script>
let text = "100, 1000 or 10000? 10";
let pattern = /\d{3,4}/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions

A global search for a sequence of three to four digits:

100,1000,1000

JavaScript RegExp {X,} Quantifier

- The n{X,} quantifier matches any string that contains a sequence of at least X n's.
- X must be a number.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global search for a sequence of at least three digits:</p>

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

<script>
let text = "100, 1000 or 10000? 10000000000 10";
let pattern = /\d{3,}/g;
let result = text.match(pattern);

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

JavaScript Regular Expressions

A global search for a sequence of at least three digits:

100,1000,10000,10000000000

JavaScript RegExp $ Quantifier

The *n*$ quantifier matches any string with *n* at the end of it.

```
<h2>JavaScript Regular Expressions</h2>
<p>A search for "is" at the end of a string:</p>
<p id="demo"></p>

<script>
let text = "Is this his";
let pattern = /is$/;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions
A search for "is" at the end of a string:

is

The **^n** quantifier matches any string with n at the beginning of it.

```
<h2>JavaScript Regular Expressions</h2>
<p>A search for "Is" at the beginning of a string:</p>
<p id="demo"></p>
<script>
let text = "Is this his";
let pattern = /^Is/g;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions
A search for "Is" at the beginning of a string:
Is

JavaScript RegExp ?= Quantifier

The ?=n quantifier matches any string that is followed by a specific string n.

A search for "**is**" followed by " **all**":

```
<h2>JavaScript Regular Expressions</h2>
<p>A search for "is" followed by " all".</p>
<p id="demo"></p>
<script>
let text = "Is this all there isall is all ";
let pattern = /is(?= all)/g;
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions
A search for "is" followed by " all".
is,is

JavaScript RegExp ?! Quantifier

```
<p>A global, case insensitive search for "is" not followed by " all":</p>

let text = "Is this all there is";
let pattern = /is(?! all)/gi;
```

A global, case insensitive search for "is" not followed by " all":

Is,is

JavaScript RegExp ?! Quantifier

The ?!*n* quantifier matches any string that is not followed by a specific string *n*.

```
<h2>JavaScript Regular Expressions</h2>

<p>A global, case insensitive search for "is" not followed by "
all":</p>

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

<script>
let text = "Is this all there is";
let pattern = /is(?! all)/gi;
let result = text.match(pattern);

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

JavaScript Regular Expressions
A global, case insensitive search for "is" not followed by " all":
Is,is

JavaScript RegExp constructor Property

- The constructor property returns the function that created the RegExp prototype.
- For a regular expression the constructor property returns:
- function RegExp() { [native code] }

```
<h2>JavaScript Regular Expressions</h2>

<p>The constructor property returns the function that created the RegExp prototype:</p>

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

<script>
let pattern = /Hello World/g;
let text = pattern.constructor;

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

JavaScript Regular Expressions

The constructor property returns the function that created the RegExp prototype:

function RegExp() { [native code] }

Note: Native code is binary data compiled to run on a processor, such as an Intel x86-class processor. The code is written in all 1s and 0s that must conform to the processor's instruction set architecture (ISA). Native code provides instructions to the processor that describe what tasks to carry out.

JavaScript RegExp global

```
<h2>JavaScript Regular Expressions</h2>

<p>The global property returns true if the "g" modifier is set,
otherwise false:</p>

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

<script>
let pattern = /W3S/g;
let result = pattern.global;

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

JavaScript Regular Expressions

The global property returns true if the "g" modifier is set, otherwise false:

true

JavaScript RegExp ignoreCase

The ignoreCase property specifies whether or not the "i" modifier is set.

```
<h2>JavaScript Regular Expressions</h2>

<p>The ignoreCase property returns true if the "i" modifier is set, otherwise false:</p>

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

<script>
let text = "Visit W3Schools!";
let pattern = /W3S/i;
let result = pattern.ignoreCase;
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions
The ignoreCase property returns true if the "i" modifier is set, otherwise false:

true

JavaScript lastIndex Property:

- The lastIndex property specifies the index at which to start the next match.

Note: This property only works if the "g" modifier is set.

- This property returns an integer that specifies the character position immediately after the last match found by exec( ) or test( ) methods.

Note: exec( ) and test( ) reset lastIndex to 0 if they do not get a match.

```
<h2>JavaScript Regular Expressions</h2>
<p>The lastIndex property specifies the index at which to start the next match:</p>

<p id="demo"></p>
<script>
let text = "The rain in Spain stays mainly in the plain";
let pattern = /ain/g;
let result = "";
while (pattern.test(text)==true)  {
  result += "Found at position " + pattern.lastIndex + "<br>";
}
document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Regular Expressions

The lastIndex property specifies the index at which to start the next match:

Found at position 8
Found at position 17
Found at position 28
Found at position 43

JavaScript multiline Property

- The multiline property specifies whether or not the m modifier is set.
- This property returns true if the "m" modifier is set, otherwise it returns false.

```
<h2>JavaScript Regular Expressions</h2>

<p>The multiline property returns true if the "m" modifier is
set, otherwise false:</p>

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

<script>
let text = "Visit W3Schools!";
let pattern = /W3S/gi;
let result = pattern.multiline;

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

JavaScript Regular Expressions

The multiline property returns true if the "m" modifier is set, otherwise false:

false

JavaScript source Property

The source property returns the text of the RegExp pattern.

```
<h2>JavaScript Regular Expressions</h2>

<p>The source property returns the text of a RegExp
pattern:</p>

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

<script>
let text = "Visit W3Schools";
let pattern = /W3S/g;
let result = pattern.source;

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

JavaScript Regular Expressions

The source property returns the text of a RegExp pattern:

W3S

Regular expression to check whether first letter is capital or not

```
<html>
<head>
   <title>Check First Letter Capital</title>
   <script>
      function checkFirstLetterCapital() {
         // Get the input value from the textbox
         var inputText = document.getElementById("textInput").value;

         // Regular expression to match the first letter being capitalized
         var regex = /^[A-Z]/;

         // Check if the first letter is capitalized or not
         if (inputText.match(regex)) {
            alert("First letter is capitalized!");
         } else {
            alert("First letter is not capitalized!");
         }
      }
   </script>
</head>
<body>
   <input type="text" id="textInput" placeholder="Enter a string">
   <button onclick="checkFirstLetterCapital()">Check</button>
</body>
</html>
```

Regular expression to enter only digits between 0-5

```
<html>
<head>
   <title>Accept Numbers 0 to 5</title>
</head>
<body>
   <input type="text" id="numberInput" placeholder="Enter a number between 0 and 5">
   <button onclick="validateInput()">Check</button>
   <script>
      function validateInput() {
         var inputValue = document.getElementById("numberInput").value;

         if (inputValue.match(/^[0-5]$/)) {
            // Input is valid
            alert("Input is valid: " + inputValue);
         } else {
            // Input is invalid
            document.getElementById("numberInput").value = ""; // Clear the input value
            alert("Please enter a number between 0 and 5.");
         }
      }
   </script>
</body>
</html>
```

Form Validation using Regular Expression---email validation

```
<html>

<head>
   <title>creating mailing system</title>
   <style>
      legend {
         display: block;
         padding-left: 2px;
         padding-right: 2px;
         border: none;
      }
   </style>

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

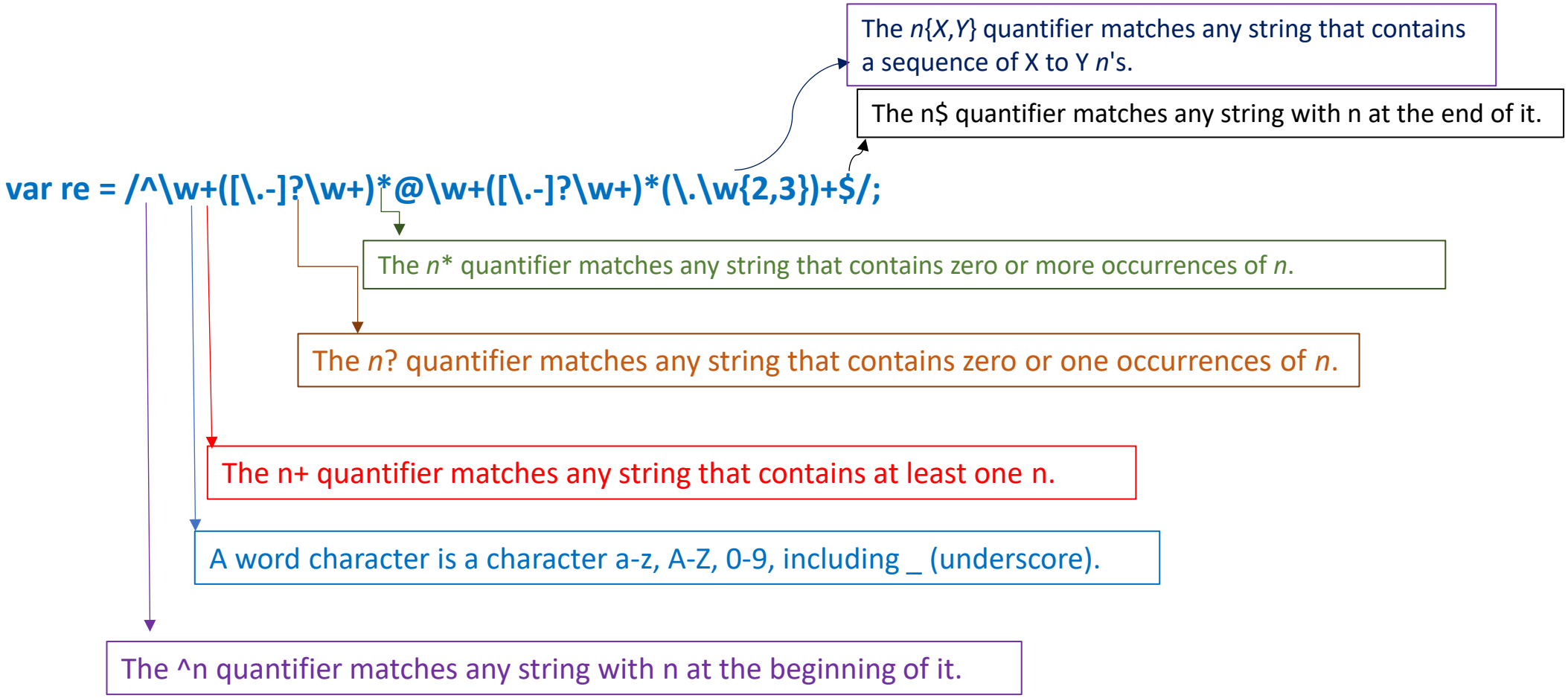
```
function validate() {

        var user = document.getElementById("e").value;
        var user2 = document.getElementById("e");

        var re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
                        ^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$

        if (re.test(user))
            {
            alert("done");
            return true;
            }

        else
            {
           user2.style.border = "red solid 3px";
           return false;
        }
      }
   </script>
</head>
```

- The **^n** quantifier matches any string with n at the beginning of it.
- The **\w** metacharacter matches word characters.
- A word character is a character a-z, A-Z, 0-9, including _ (underscore).
- The ***n*+** quantifier matches any string that contains at least one *n*.
- The ***n*?** quantifier matches any string that contains zero or one occurrences of *n*.
- The **n*** quantifier matches any string that contains zero or more occurrences of n.
- The **n$** quantifier matches any string with n at the end of it.

```
<body bgcolor="cyan">
   <center>
      <h1>Email Registration</h1>
      <form>
         <fieldset style="width:300px">
            <legend>Registation Form</legend>
            <table>
               <tr>
                  <input type="text" placeholder="firstname" maxlength="10">
               </tr>
               <br><br>
               <tr>
                  <input type="text" placeholder="lastname" maxlength="10">
               </tr>
               <br><br>
               <tr>
                  <input type="email"
                     placeholder="username@gmail.com" id="e">
               </tr>
               <br><br>
               <tr>
                  <input type="password" placeholder="password">
               </tr>
```

```
<br><br>
            <tr>
              <input type="password" placeholder="confirm">
            </tr>
            <br><br>
            <tr>
              <input type="text" placeholder="contact">
            </tr>
            <br><br>
            <tr>
              <label>Gender:</label>
              <select id="gender">
                <option value="male">male</option>
                <option value="female">female</option>
                <option value="others">others</option>
              </select>
            </tr>
            <br><br>
            <tr><input type="submit" onclick="validate()" value="create">
            </tr>
          </table>
        </fieldset>
      </form>
   </center>
</body>
 </html>
```

Regular for Email verification:



var re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;

^: Asserts the start of the string.

\w+: Matches one or more word characters (letters, digits, or underscores). This represents the username part of the email address.

([\.-]?\w+)*: This part represents the domain name.

[\.-]?: Matches an optional **dot (.) or hyphen (-).**
\w+: Matches one or more word characters.
*: Allows for zero or more occurrences of the preceding group, allowing for multiple domain segments.

@: Matches the @ symbol, which separates the username from the domain name.

\w+: Matches one or more word characters. This represents the domain name.

([\.-]?\w+)*: Similar to the username part, this allows for multiple domain segments separated by dots or hyphens.

(\.\w{2,3})+: This part matches the top-level domain (TLD), which consists of a dot followed by 2 or 3 word characters (e.g., .com, .org, .net).

\.\w{2,3}: Matches a dot followed by 2 or 3 word characters.

+: Allows for one or more occurrences of the preceding group, enabling support for multiple TLDs like .co.uk or .com.au.
$: Asserts the end of the string.

Example1:

txt = "hello world"

#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":

```
let pattern = /he..o/g;
let result = text.match(pattern);
```

Examples:

JavaScript Regex that you will use for phone number validation:

```
/^\(?(\d{3})\)?[- ]?(\d{3})[- ]?(\d{4})$/
```

I. /^\(?: The number may start with an open parenthesis.

II. (\d{3}): Then three numeric digits must be entered for valid formats. If there is no parenthesis, the number must start with these digits.

III. \)?: It allows you to include a close parenthesis.

IV. [-]?: The string may optionally contain a hyphen. It can be placed either after the parenthesis or following the first three digits. For example, (123)- or 123-.

V. (\d{3}): Then the number must contain another three digits. For example, it can look like this: (123)-456, 123-456, or 123456.

VI. [-]?: It allows you to include an optional hyphen in the end, like this: (123)-456-, -123- or 123456-.

VII. (\d{4})$/: Finally, the sequence must end with four digits. For example, (123)-456-7890, 123-456-7890, or 123456-7890.

(123) 456-7890

(123)456-7890

123-456-7890

1234567890

Chapter 6:
Navigation and javascript framework

Status bar in javascript:

The window.statusbar property returns a Statusbar object representing the status bar of browser. It appears at the bottom of browser. However, it is almost impossible for you to interact with the Statusbar via Javascript

```
window.statusbar
// Or simple:
Statusbar
```

The tendency of modern browsers is to make the Viewport window as wide as possible, therefore, they remove other components like Statusbar, or make Menubar smaller

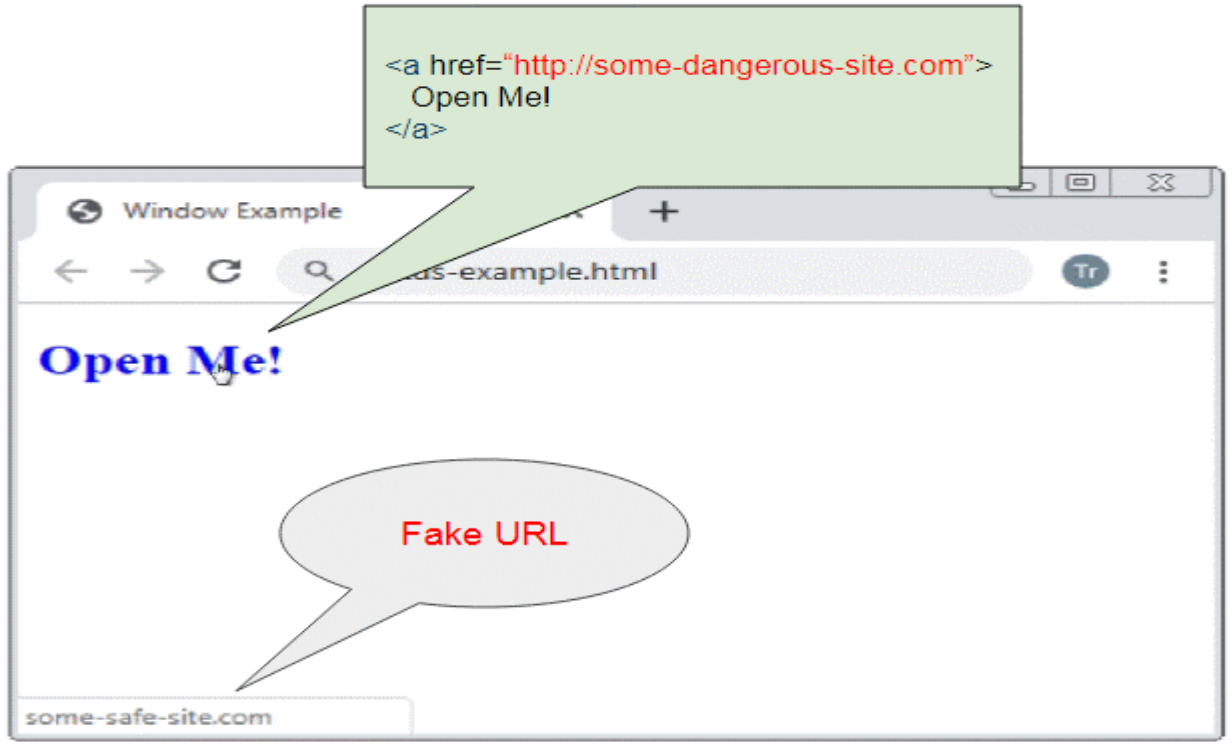
For modern browsers, you see the status bar appear only when a user moves the mouse on the surface of a link.

statusbar.visible

The statusbar.visible property returns true if the status bar is displayed on browser. However, this is unreliable property. You get a true value, which does not mean that you are seeing the status bar.

```
<html>
<head>
   <title>Statusbar</title>
   <meta charset="UTF-8">

</head>
<body>
   <h3>statusbar.visible</h3>

   <br/><br/>
   <button onclick="alert(statusbar.visible)">
      alert(statusbar.visible)
   </button>

</body>
</html>
```

The status property of the window object help you establish a content of text shown on he status bar. By default, for security reasons, most browsers disable this feature for JavaScript. However, if an user wants, they are able to enable this feature for JavaScript by entering the "Options" of the browser.



Before clicking on a link, users often move the mouse over the link's surface to preview its address displayed in the status bar, and only click this link when they feel safe. Some websites may take advantage of window.status to display a fake content.

The status property is deprecated.
It should be avoided to prevent RUN-TIME ERRORS in the future.

Status Bar Example:1

```
<html>
   <head>
     <title>JavaScript Status Bar</title></head>
     <body onLoad="window.status='Welcome!';return true">
   </body>
 </html>
```

onLoad tells the browser that as soon as your page finished loading, it will display in your current window's status bar (window.status) the message "Welcome!". The return true is necessary because with out, it won't work.

Status Bar Example: 2

```
<html>
 <head>
  <title>JavaScript Status Bar</title>
 </head>
 <body>
  <a href="http://www.htmlcenter.com"
  onMouseOver="window.status='HTMLcenter';return true"
  onMouseOut="window.status= '';return true">

    HTMLcenter

  </a>
 </body>
</html>
```

When the user moves his mouse over the link, it will display “HTMLcenter” in the status bar. When he moves his mouse away from the link the status bar will display nothing.

Moving the message along the status bar

```
<html>    <head>    <title>Javascript ScrollText</title>
      <script language="javascript">
         msg="This is an example of scrolling message";
         spacer="..... .....";
         pos=0;
         function ScrollMessage()
         {
            window.status=msg.substring(pos,msg.length)+spacer;
            pos++;
            if(pos>msg.length)
               pos=0;
            window.setTimeout("ScrollMessage()",100);
         }
         ScrollMessage();
      </script>
   </head>
   <body>
      <p>Scrolling Message Example</p>
      <p>Look at the status line at the bottom of the page</p>
   </body>
</html>
```

JavaScript frameworks

- JavaScript frameworks are a collection of libraries containing code written in JavaScript, making life a lot easier for software developers.
- Each JavaScript framework offers pre-built codes for different areas and different purposes in software development, saving time for the developer.

JavaScript frameworks

JavaScript frameworks provide structure for your code, so in that sense, they're pretty essential to your programming. There are many useful JavaScript frameworks that developers regularly use, and we will cover some of these in this article. A JavaScript framework provides a blueprint, so you have a guide to follow rather than having to start coding your website from scratch.

Most frameworks are open-source, meaning they are constantly being improved by the community that uses them, so they are always up to date. They are by no means set in stone either; you are free to tweak the framework you choose to suit your own website or application.

Why use a JavaScript framework?

- Developers created JavaScript frameworks to make life easier for themselves. They allow programmers to use the most up-to-date JavaScript features and tools without having to go through the arduous task of coding them from scratch by themselves.
- These frameworks are templates that provide a foundation for software applications. It collects shared resources like libraries, reference documents, images and more and packages them for developers to use. With these frameworks, programmers can add better functionality and more to a web page and website.

Popular JavaScript Framework libraries

AngularJS

AngularJS is an open-source framework that came into being in October 2010 and is the oldest available. It's a good one to choose when you're thinking about which framework would be best to go for; brilliantly, it is supported by Google! There are even apps built into cars made by General Motors that have been developed in Angular – it has emerged as a bit of a market leader in JavaScript frameworks. Google's lead Angular developer, Igor Minar, believes that Angular is the most widely used framework because it, more so than others, encourages regular updates and developments.

FRONT-END FRAMEWORKS

REACT

React.js is an efficient and flexible JavaScript library for building user interfaces created by Facebook. Technically, React is a JS library, but it is often discussed as a web framework and is compared to any other open source JavaScript framework.

React makes it easy to create interactive user interfaces because it has predictable JavaScript code that is easy to debug. Furthermore, it provides a REACT component system where blocks of JavaScript code can be written once and reused repeatedly in different parts of the application or even other applications.

ANGULAR

AngularJS is a popular enterprise-level JavaScript framework used for developing large and complex business applications. It is an open-source web framework created by Google and supported by both Google and Microsoft.

VUE

Vue.js is a progressive framework for building user interfaces. It is an up-and-coming framework that helps developers in integrating with other libraries and existing projects. It has an ecosystem of libraries that allow developers to create complex and solid single-page applications.

BACK-END FRAMEWORKS

EXPRESS

Express.js is a flexible, minimalistic, lightweight, and well-supported framework for Node.js applications. It is likely the most popular framework for server-side Node.js applications. Express provides a wide range of HTTP utilities, as well as high-performance speed. It is great for developing a simple, single-page application that can handle multiple requests at the same time.

NEXT.JS

Next.js is a minimalistic framework that allows a a JavaScript developer to create a server-side rendering and static web applications using React.js. It is one of the newest and hottest frameworks that takes pride in its ease of use. Many of the problems developers experience while building applications using React.js are solved using Next.js. It has many important features included “out of the box,” and makes development a JavaScript breeze.

Banner Ads

Displaying banners ads is a common practice for showing advertisements on web pages to the visitors. Banners ads are normally created using standard graphic tools such as Photoshop, Paintbrush Pro, and other software. Banner ads can be static or animated. Animated images are animated GIF files or flash movies. Flash movies are created using Macromedia Flash and the browsers must have installed flash plugin to view the movies. On the other hand, you can create some animated effect using JavaScript, like rotating static banner ads at a certain time interval.

Document images

- The images property returns a collection of all <img> elements in a document.
- The images property returns an HTMLCollection.
- The images property is read-only.

HTMLCollection

- An HTMLCollection is an array-like collection (list) of HTML elements.
- The elements in a collection can be accessed by index (starts at 0).
- The length Property returns the number of elements in the collection.

Syntax:

document.images

Properties:

Property	Description
length	The number of <img> elements in the collection.

Methods:

Method	Description
[*index*]	Returns the element with the specified index (starts at 0). Returns null if the index is out of range.
item(*index*)	Returns the element with the specified index (starts at 0). Returns null if the index is out of range.
namedItem(*id*)	Returns the element with the specified id. Returns null if the id does not exist.

Return Value:

Type	Description
Object	An HTMLCollection Object. All <img> elements in the document. The elements are sorted as they appear in the document.

The images Property

```
<html>
<body>

<h1>The Document Object</h1>
<h2>The images Property</h2>

<img src="klematis.jpg" alt="flower" width="150" height="113">
<img src="klematis2.jpg" alt="flower" width="152" height="128">
<img src="smiley.gif" alt="Smiley face" width="42" height="42">

<p>The number of images is:</p>
<p id="demo"></p>

<script>
let numb = document.images.length;
document.getElementById("demo").innerHTML = numb;
</script>

</body>
</html>
```

The Document Object

The images Property

The number of images is:

3

Example:

```
<html>
<body>
<h1>The Document Object</h1>
<h2>The images Property</h2>

<img src="klematis.jpg" alt="flower" width="150" height="120">
<img src="klematis2.jpg" alt="flower" width="150" height="120">

<p>Display the URL of each img element:</p>
<p id="demo"></p>

<script>
const myImages = document.images;
let text = "";
for (let i = 0; i < myImages.length; i++) {
  text +=  myImages[i].src + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The Document Object

The images Property

Display the URL of each img element:

https://www.w3schools.com/jsref/klematis.jpg
https://www.w3schools.com/jsref/klematis2.jpg

Example:

```
<html>
<body>

<h1>The Document Object</h1>
<h2>The images Property</h2>

<img src="klematis.jpg" alt="flower" width="150" height="120">
<img src="klematis2.jpg" alt="flower" width="150" height="120">

<p>The URL of the first image is:</p>
<p id="demo"></p>

<script>
let src= document.images.item(0).src;
document.getElementById("demo").innerHTML = src;
</script>

</body>
</html>
```

The Document Object

The images Property

The URL of the first image is:

https://www.w3schools.com/jsref/klematis.jpg

Add a black border to the first <img> element:

```
<html>
<body>

<h1>The Document Object</h1>
<h2>The images Property</h2>

<p>Add a black border to the first image in the document:</p>
<img src="klematis.jpg" alt="flower" width="150" height="120">
<img src="klemais2.jpg" alt="flower" width="150" height="120">

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

<script>
const img = document.images[0];
img.style.border = "10px solid black";
</script>

</body>
</html>
```

The Document Object

The images Property

Add a black border to the first image in the document:

Window setTimeout()

- The setTimeout() method calls a function after a number of milliseconds.
- 1 second = 1000 milliseconds**.**

```
<html>
<body>
<h1>The Window Object</h1>
<h2>The setTimeout() Method</h2>

<p>Wait 5 seconds for the greeting:</p>

<h2 id="demo"></h2>

<script>
const myTimeout = setTimeout(myGreeting, 5000);

function myGreeting() {
  document.getElementById("demo").innerHTML = "Happy Birthday!"
}
</script>
</body>
</html>
```

The Window Object
The setTimeout() Method
Wait 5 seconds for the greeting:

The Window Object
The setTimeout() Method
Wait 5 seconds for the greeting:
Happy Birthday!

Creating Rotating Banner Ads

Rotating banners ads comprises several banner images that constantly rotate on a webpage at a fix time interval. You can create these banner images using standard graphics tools.

```
<html>
<head>
<script language="Javascript">
MyBanners=new Array('banner1.jpg','banner2.jpg','banner3.jpg','banner4.jpg')
banner=0
function ShowBanners()
{ if (document.images)
{ banner++
if (banner==MyBanners.length) {
banner=0}
document.ChangeBanner.src=MyBanners[banner]
setTimeout("ShowBanners()",5000)
}}
</script>
<body onload="ShowBanners()">
<center>
<img src="banner1.jpg" width="900" height="120" name="ChangeBanner"/>
</center>
</body>
</html>
```

Creating Rotating Banner Ads with URL Links

Creating rotating banner images will provide the visitor to your webpage with some basic information. However, if you want the visitor to get more information by clicking on the banner images, you need to create rotating banner ads that contain URL links.

```
<html>
<head>
<script language="Javascript">
MyBanners=new Array('1.jpg','2.jpg','3.jpg')
MyBannerLinks=new Array('http://www.a.net/','http://www.b.com/','http://c.com/')
```

```
 banner=0
function ShowLinks()
{
        document.location.href="http://www."+ MyBannerLinks[banner]
}
function ShowBanners()
{
        if (document.images)
        {
                banner++
        if (banner==MyBanners.length)
        {
                banner=0
        }
        document.ChangeBanner.src=MyBanners[banner]
        setTimeout("ShowBanners()",5000)
        }
}
</script>
<body onload="ShowBanners()">
<center>
<a href="javascript: ShowLinks()">
<img src="1.jpg" width="900" height="120" name="ChangeBanner"/></a>
</center></body></html>
```

Slide Show

The JavaScript code for the slideshow is almost similar to the JavaScript code of the rotating banners but it gives control to the user to choose the banner ads he or she wants to see by clicking on the forward and backward buttons.

To create the JavaScript slideshow, first of all, you need to create a few banner images using some graphics tools, or you can snap some photos with your digital camera or your smartphone.

```
<html ><head>
<script language="Javascript">
MySlides=new Array('1.jpg','2.jpg','3.jpg')
Slide=0
function ShowSlides(SlideNumber)
{
Slide=Slide+SlideNumber ;
if (Slide>MySlides.length-1)
{
Slide=0
}
if (Slide<0)
{
Slide=MySlides.length-1;
}
document.DisplaySlide.src=MySlides[Slide];
}
 </script>
```

```
</head>
<body>
<P align="center"><img src="1.jpg" name="DisplaySlide" width="900″ height="120″ /><p>
<center>
<table border=0>
<tr>
<td align=center>
<input type="button" value="Back" onclick="ShowSlides(-1)">
<input type="button" value="Forward" onclick="ShowSlides(1)">
</td>
</tr>
</table>
</center>
</body>
</html>
```

Chapter 6:

Menus, navigation and web page protection

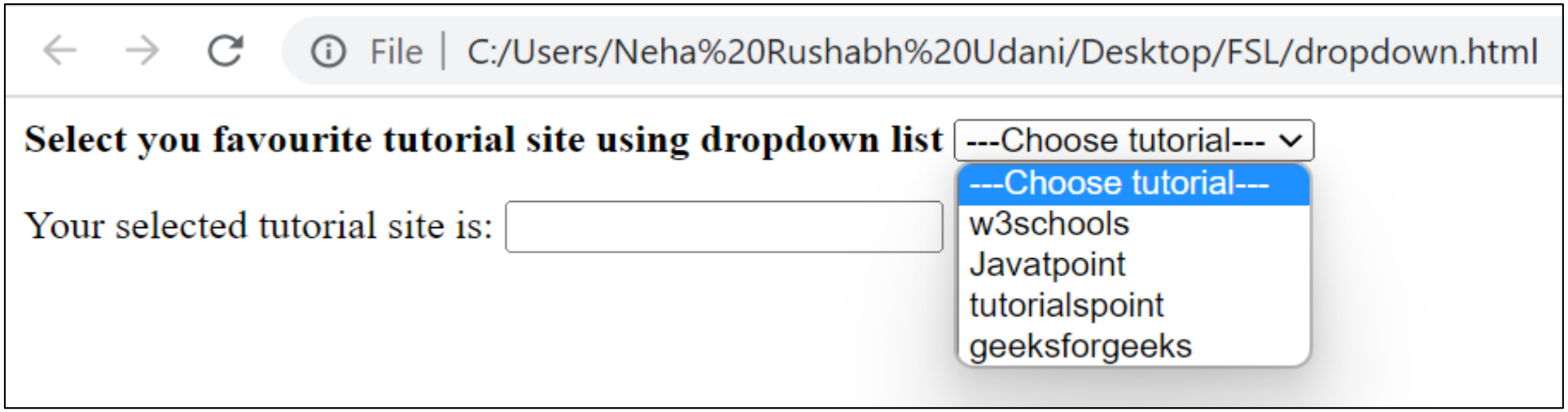
dropdown list using <select> tab

```
<html><head><title>dropdown menu using select tab</title></head>
<script>
function favTutorial() {
let mylist1 = document.getElementById("myList");
document.getElementById("favourite").value = mylist1.options[mylist1.selectedIndex].text;
}
</script><body><form>
<b> Select you favourite tutorial site using dropdown list </b>
<select id = "myList" onchange = "favTutorial()" >
<option> ---Choose tutorial--- </option>
<option> w3schools </option>
<option> Javatpoint </option>
<option> tutorialspoint </option>
<option> geeksforgeeks </option>
</select>
<p> Your selected tutorial site is:
<input type = "text" id = "favourite" size = "20" </p>
</form>
</body>
</html>
```

The selectedIndex property sets or returns the index of the selected option in a drop-down list.

The index starts at 0.

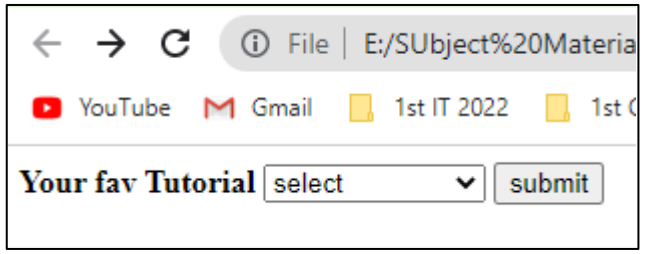
Note: If the drop-down list allows multiple selections it will only return the index of the first option selected.

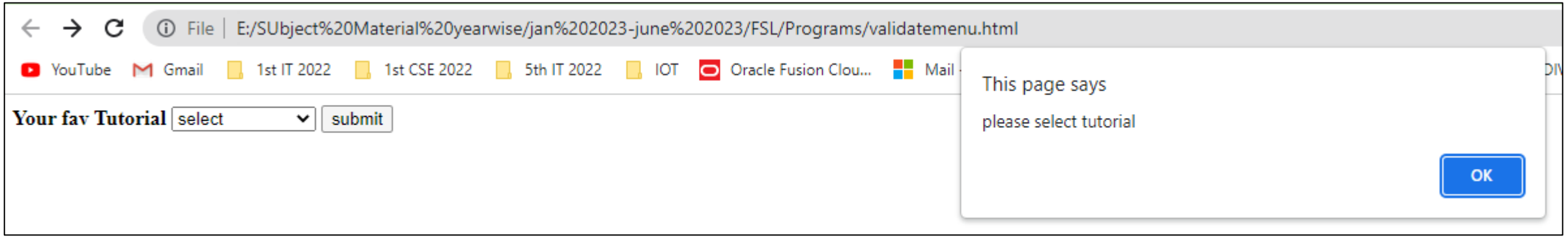


validating menu selection

```
<html><title>validating menu selection</title>
<script>
function validate()
{
        var s1=document.getElementById('myList').value;
        if(s1=="select")
        {
                alert("please select tutorial");
                return false;
        }
                return true;
}
</script>
```

```
<body>
<form name="form1" onsubmit = "return validate()"
action="google.com">
<b>Your fav Tutorial</b>
<select id = "myList">
<option>select</option>
<option> w3schools </option>
<option> Javatpoint </option>
<option> tutorialspoint </option>
<option> geeksforgeeks </option>
</select>
<input type="submit" value="submit">
</body></html>
```





Dropdown list using button and div tab

Block vs. Inline

HTML tags like , <p>, <ul> take full-width of space and each starts with a new line, whereas other HTML tags like , <img> or <a> don't need a new line and can be placed side by side.

This is because of the different display behaviors: Block or inline.

```
<body>
  <p>I'm a paragraph</p>
  <p>I'm a paragraph too</p>
  <span>I'm a word</span>
  <span>I'm a word too.</span>
</body>
```

I'm a paragraph

I'm a paragraph too

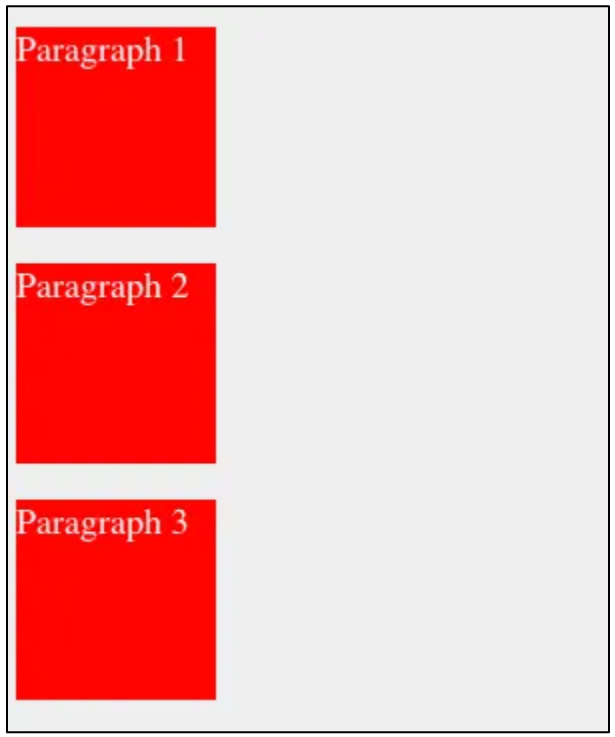
I'm a word I'm a word too.

Block-level elements

- Take full-width (100% width) by default
- Each gets displayed in a new line
- Width & height properties can be set

Since <p> tags are block-level elements, width & height properties can be set:

```
p {
  height: 100px;
  width: 100px;
  background: red;
  color: white;
  display: block;
}
```



If no width was declared here, then the default width of <p> would be 100%. However, We declared a 100px width and the next <p> element still starts with a new line:

Inline elements

- Take only as much space as they need
- Displayed side by side
- Don’t accept width or height properties, and top-bottom margin

```
p {
  height: 100px;
  width: 100px;
  background: red;
  color: white;
  display: inline;
}
```

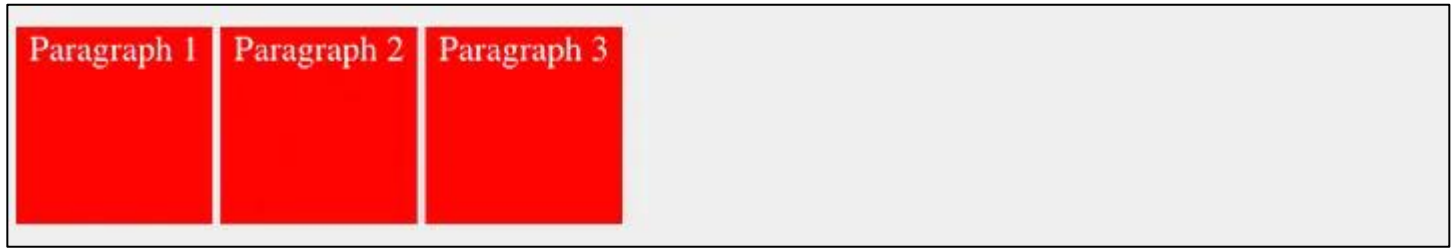
Since our <p> tag is now an inline element, they will be placed side by side and the width & height properties have no effect anymore:

Display: Inline-block

In some cases, both of the display values may not be enough for better web design. At that point, a third display behavior comes to the rescue and also makes alignment much easier: display: inline-block .

display: inline-block declaration shows both the characteristics of inline and block-level elements.

```
p {
  display: inline-block;
  height: 100px;
  width: 100px;
  background: red;
  color: white;
}
```

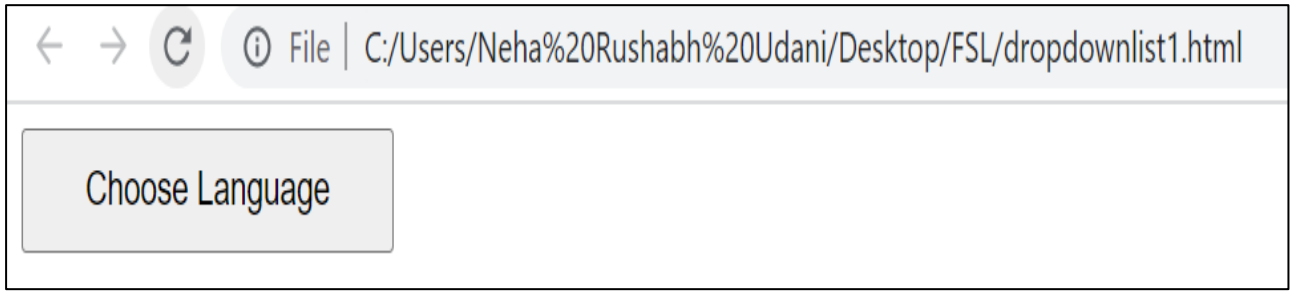


Dropdown list using button and div tab

```
<html> <head> <title>dropdown menu using
button</title> </head> <style>

/* set the position of dropdown */
.dropdown {
    position: relative;
    display: inline-block;
}
/* set the size and position of button on web */
.button {
    padding: 10px 30px;
    font-size: 15px;
}
/* provide css to background of list items */
#list-items {
   display: none;
   position: absolute;
   background-color: white;
   min-width: 185px;
}
```

```
/* provide css to list items */
#list-items a {
    display: block;
    font-size: 18px;
    background-color: #ddd;
    color: black;
    text-decoration: none;
    padding: 10px;
}
</style>
```

```
<script>
    //show and hide dropdown list item on button click
    function show_hide() {
      var click = document.getElementById("list-items");
      if(click.style.display ==="none") {
        click.style.display ="block";
      } else {
        click.style.display ="none";
      }
    }
  </script>
```




```
<body>
<div class="dropdown">
  <button onclick="show_hide()" class="button">Choose Language</button>
   <center>
     <!-- dropdown list items will show when we click the drop button -->
     <div id="list-items">
       <a href="#"> Hindi </a>
       <a href="#"> English </a>
       <a href="#"> Spanish </a>
       <a href="#"> Chinese </a>
       <a href="#"> Japanese </a>
     </div>
   </center>

</body>
</html>
```

dynamically changing a menu-Chained Menu:

Chain of pull-down menus in which the option selected from the first pull-down menu determines the options that are available in the second pull-down menu.

Document createElement()

The createElement() method creates an element node.

```
<html>
<body>
<h1>The Document Object</h1>
<h2>The createElement() Method</h2>
<p>Create a p element with some text:</p>
<script>
// Create element:
const para = document.createElement("p");
para.innerText = "This is a paragraph.";
// Append to body:
document.body.appendChild(para);
</script>
</body>
</html>
```

The Document Object
The createElement() Method
Create a p element with some text:

This is a paragraph.

```
const para = document.createElement("h1");
para.innerText = "This is a paragraph.";
```

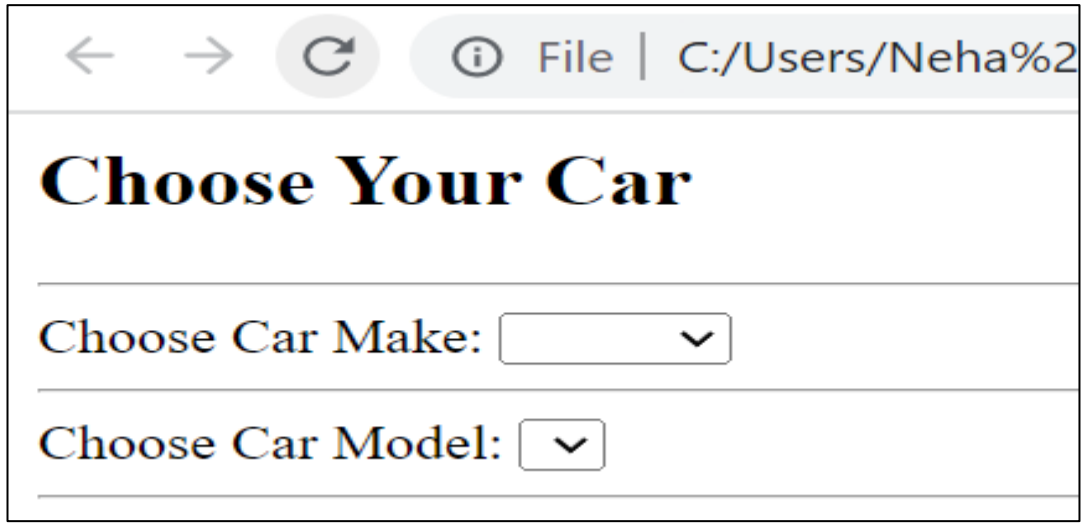
The Document Object
The createElement() Method
Create a p element with some text:

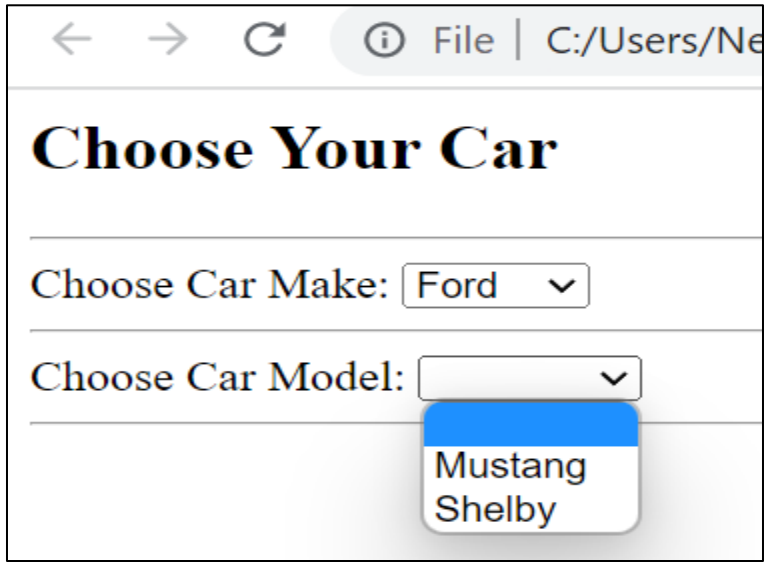
This is a paragraph.

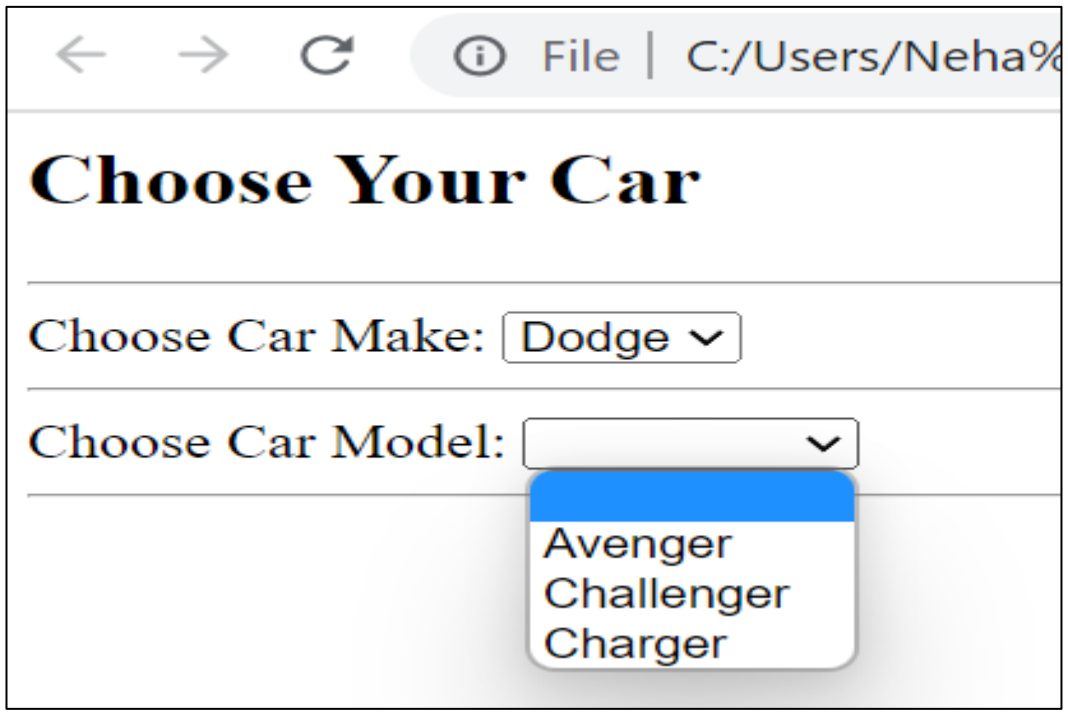
dynamically changing a menu

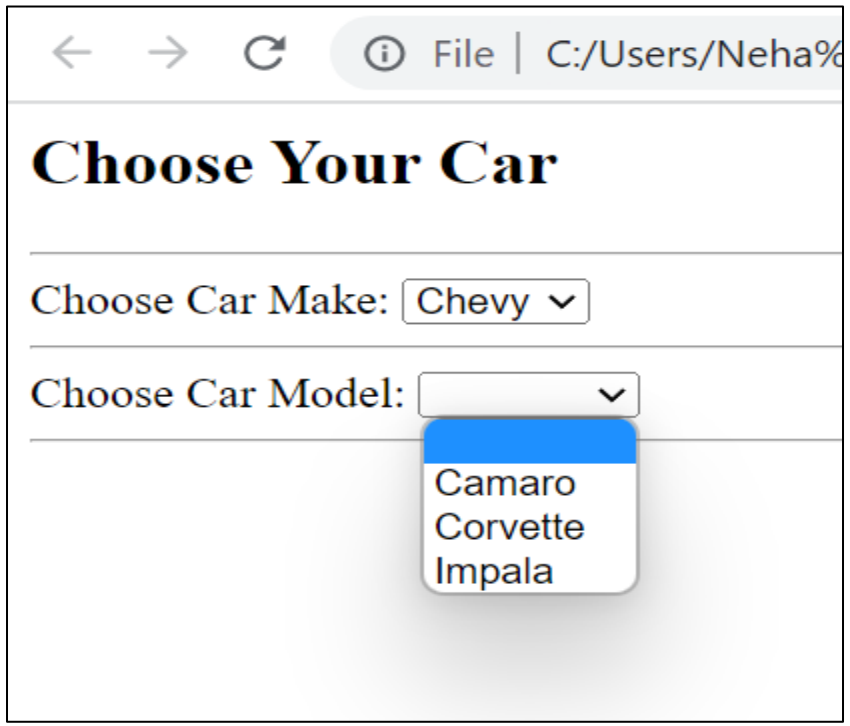
```
<html>
<script>
function populate(s1,s2){
        var sel1 = document.getElementById(s1);
        var sel2 = document.getElementById(s2);
        sel2.innerHTML = "";
        if(sel1.value == "Chevy"){
                var optionArray = ["|","camaro|Camaro","corvette|Corvette","impala|Impala"];
        } else if(sel1.value == "Dodge"){
                var optionArray = ["|","avenger|Avenger","challenger|Challenger","charger|Charger"];
        } else if(sel1.value == "Ford"){
                var optionArray = ["|","mustang|Mustang","shelby|Shelby"];
        }
        for(var option in optionArray){
                var pair = optionArray[option].split("|");
                var newOption = document.createElement("option");
                newOption.value = pair[0];
                newOption.innerHTML = pair[1];
                sel2.options.add(newOption);
        }
}
</script>
```

```
<body>
<h2>Choose Your Car</h2>
<hr />
Choose Car Make:
<select id="slct1" name="slct1" onchange="populate(this.id,'slct2')">
  <option value=""></option>
  <option value="Chevy">Chevy</option>
  <option value="Dodge">Dodge</option>
  <option value="Ford">Ford</option>
</select>
<hr />
Choose Car Model:
<select id="slct2" name="slct2"></select>
<hr />
</body>
</html>
```


File | C:/Users/Neha%2
Choose Your Car
Choose Car Make:
Choose Car Model:


File | C:/Users/Ne
Choose Your Car
Choose Car Make: Ford
Choose Car Model:
Mustang
Shelby


File | C:/Users/Neha%
Choose Your Car
Choose Car Make: Dodge
Choose Car Model:
Avenger
Challenger
Charger


File | C:/Users/Neha%
Choose Your Car
Choose Car Make: Chevy
Choose Car Model:
Camaro
Corvette
Impala

Status bar in javascript:

The window.statusbar property returns a Statusbar object representing the status bar of browser. It appears at the bottom of browser. However, it is almost impossible for you to interact with the Statusbar via Javascript

```
window.statusbar
// Or simple:
Statusbar
```

The tendency of modern browsers is to make the Viewport window as wide as possible, therefore, they remove other components like Statusbar, or make Menubar smaller

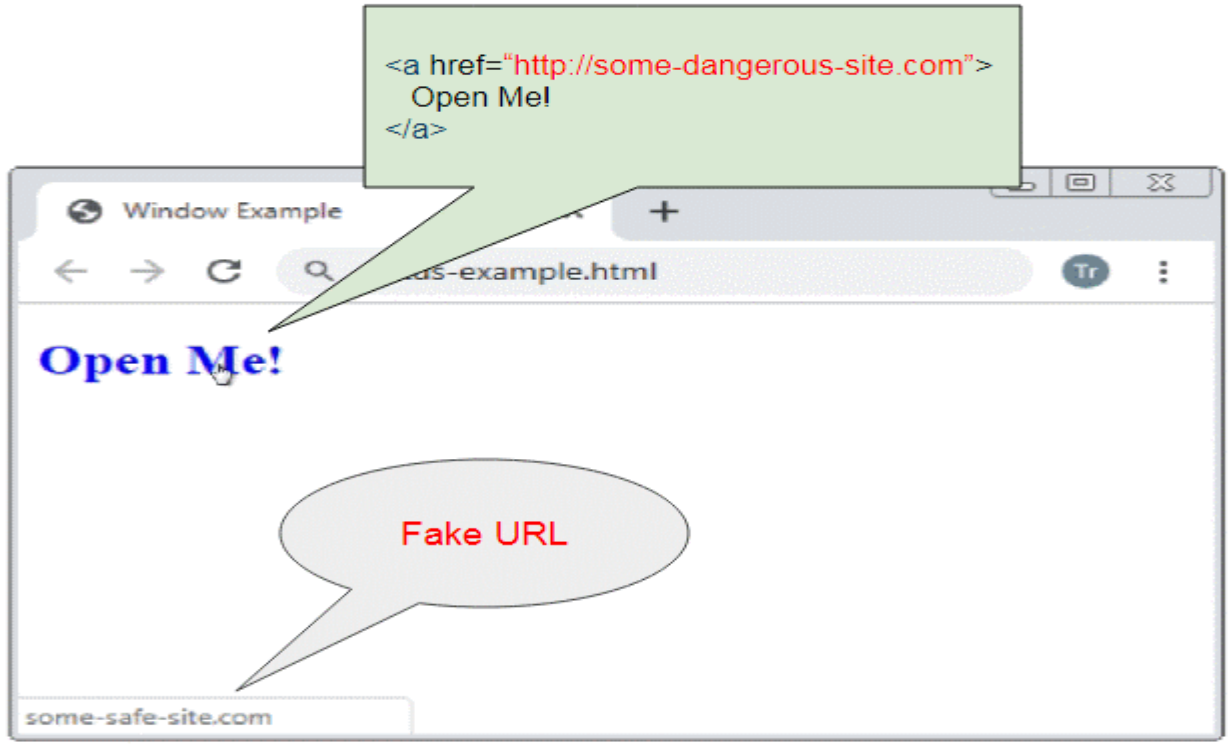
For modern browsers, you see the status bar appear only when a user moves the mouse on the surface of a link.

statusbar.visible

The statusbar.visible property returns true if the status bar is displayed on browser. However, this is unreliable property. You get a true value, which does not mean that you are seeing the status bar.

```
<html>
<head>
   <title>Statusbar</title>
   <meta charset="UTF-8">

</head>
<body>
   <h3>statusbar.visible</h3>

   <br/><br/>
   <button onclick="alert(statusbar.visible)">
      alert(statusbar.visible)
   </button>

</body>
</html>
```

The status property of the window object help you establish a content of text shown on he status bar. By default, for security reasons, most browsers disable this feature for JavaScript. However, if an user wants, they are able to enable this feature for JavaScript by entering the "Options" of the browser.



Before clicking on a link, users often move the mouse over the link's surface to preview its address displayed in the status bar, and only click this link when they feel safe. Some websites may take advantage of window.status to display a fake content.

The status property is deprecated.
It should be avoided to prevent RUN-TIME ERRORS in the future.

Status Bar Example:1

```
<html>
   <head>
     <title>JavaScript Status Bar</title></head>
     <body onLoad="window.status='Welcome!';return true">
   </body>
 </html>
```

onLoad tells the browser that as soon as your page finished loading, it will display in your current window's status bar (window.status) the message "Welcome!". The return true is necessary because with out, it won't work.

Status Bar Example: 2

```
<html>
 <head>
  <title>JavaScript Status Bar</title>
 </head>
 <body>
  <a href="http://www.htmlcenter.com"
  onMouseOver="window.status='HTMLcenter';return true"
  onMouseOut="window.status= '';return true">

    HTMLcenter

  </a>
 </body>
</html>
```

When the user moves his mouse over the link, it will display “HTMLcenter” in the status bar. When he moves his mouse away from the link the status bar will display nothing.

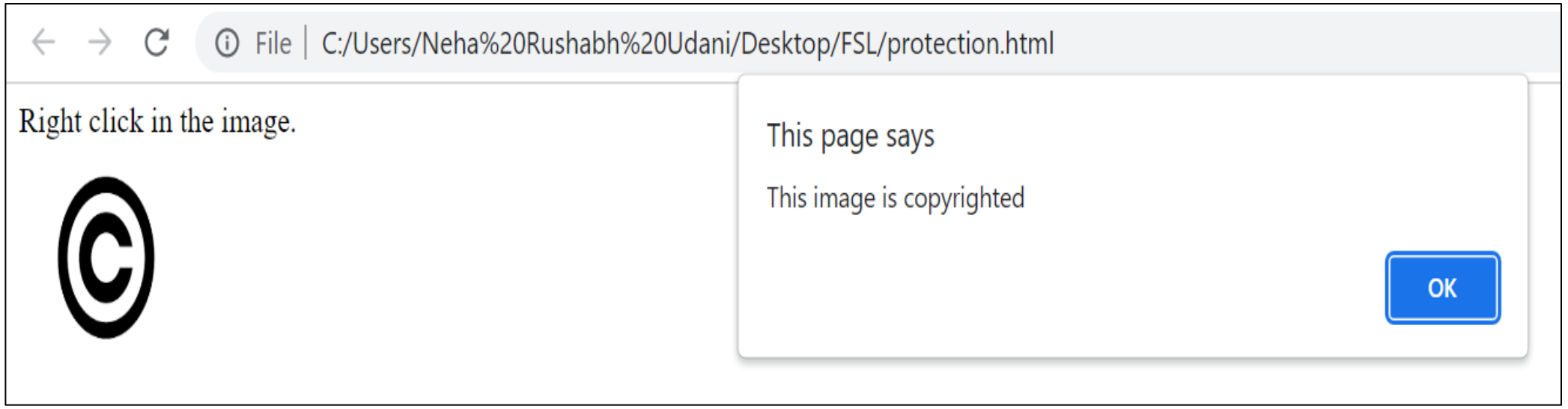
Moving the message along the status bar

```
<html>    <head>    <title>Javascript ScrollText</title>
      <script language="javascript">
         msg="This is an example of scrolling message";
         spacer="..... .....";
         pos=0;
         function ScrollMessage()
         {
            window.status=msg.substring(pos,msg.length)+spacer;
            pos++;
            if(pos>msg.length)
               pos=0;
            window.setTimeout("ScrollMessage()",100);
         }
         ScrollMessage();
      </script>
   </head>
   <body>
      <p>Scrolling Message Example</p>
      <p>Look at the status line at the bottom of the page</p>
   </body>
</html>
```

Protection web page

```
<html>
<head>
<script language="JavaScript">
function function2()
{
alert("This image is copyrighted")
}
</script>
</head>
<body oncontextmenu="function2()">
<p>Right click in the image.</p>
<img oncontextmenu="function2()" src="copy.png" alt="www.java2s.com" width="99" height="76">
</body>
</html>
```


File | C:/Users/Ne
Right click in the image.


File | C:/Users/Neha%20Rushabh%20Udani/Desktop/FSL/protection.html
Right click in the image.
This page says
This image is copyrighted
OK

Folding tree menu:

```
<html><head><style>
ul, #myUL {
  list-style-type: none;
}

.caret::before {
  content: "\25B6";
  color: black;
  display: inline-block;
  margin-right: 6px;
}

.caret-down::before {
    transform: rotate(90deg);
}

.nested {
  display: none;
}

.active {
  display: block;
}
</style></head>
```

For UL

::before selector inserts something before the content of each selected element(s).

Black right pointing triangle.

The rotate() CSS function defines a transformation that rotates an element around a fixed point on the 2D plane

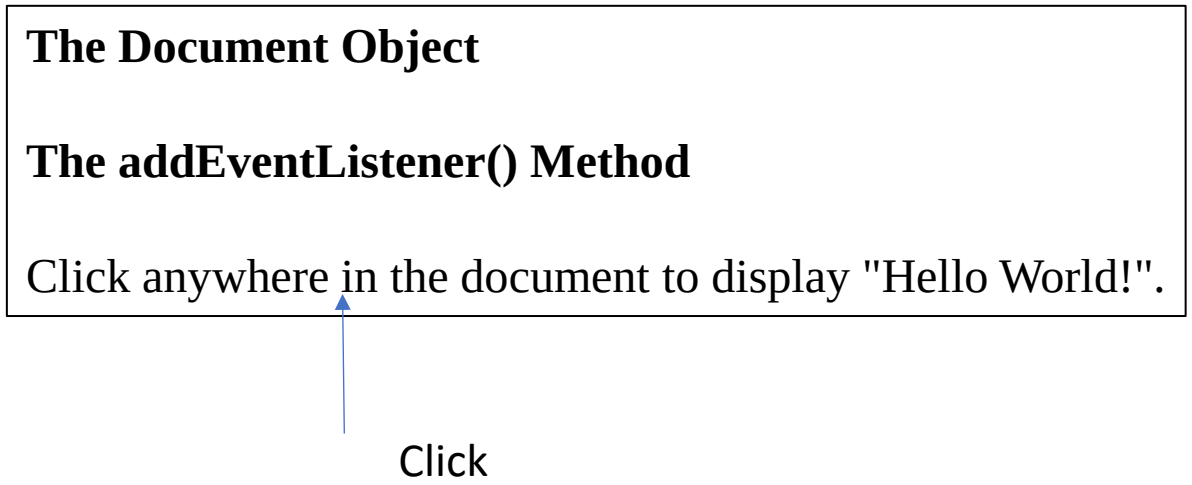
```
<h2>Tree View</h2>
<p>A tree view represents a hierarchical view of information, where each item can have a number of subitems.</p>
<p>Click on the arrow(s) to open or close the tree branches.</p>

<ul id="myUL">
  <li><span class="caret">Beverages</span>
    <ul class="nested">
      <li>Water</li>
      <li>Coffee</li>
      <li><span class="caret">Tea</span>
        <ul class="nested">
          <li>Black Tea</li>
          <li>White Tea</li>
          <li><span class="caret">Green Tea</span>
            <ul class="nested">
              <li>Sencha</li>
              <li>Gyokuro</li>
              <li>Matcha</li>
              <li>Pi Lo Chun</li>
            </ul>
          </li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

```
<script>
var toggler = document.getElementsByClassName("caret");
var i;

for (i = 0; i < toggler.length; i++) {
  toggler[i].addEventListener("click", function() {
    this.parentElement.querySelector(".nested").classList.toggle("active");
      });
}
</script>

</body>
</html>
```

The addEventListener() method attaches an event handler to a document.
The querySelector() method returns the first element that matches a CSS selector.

HTML DOM Document addEventListener()

```
<html>
<body>

<h1>The Document Object</h1>
<h2>The addEventListener() Method</h2>

<p>Click anywhere in the document to display "Hello World!".</p>

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

<script>
document.addEventListener("click", function(){
  document.getElementById("demo").innerHTML = "Hello World!";
});
</script>

</body>
</html>
```



The Document Object

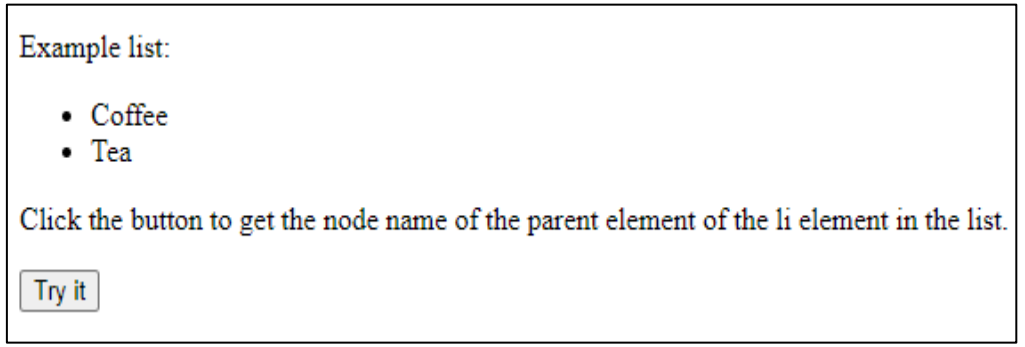
The addEventListener() Method

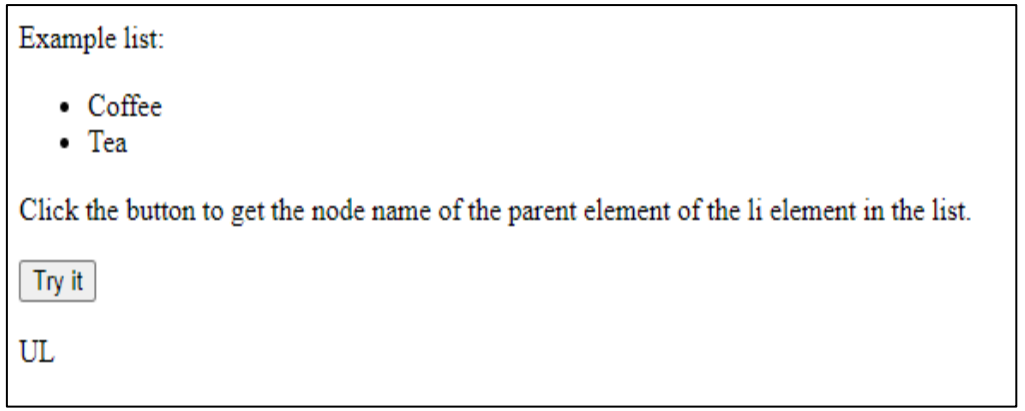
Click anywhere in the document to display "Hello World!".

Hello World!

HTML DOM parentElement Property

```
<html>
<body>
<p>Example list:</p>
<ul>
  <li id="myLI">Coffee</li>
  <li>Tea</li>
</ul>
<p>Click the button to get the node
name of the parent element of the li element in the list.</p>

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

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

<script>
function myFunction() {
  var x = document.getElementById("myLI").parentElement.nodeName;
  document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

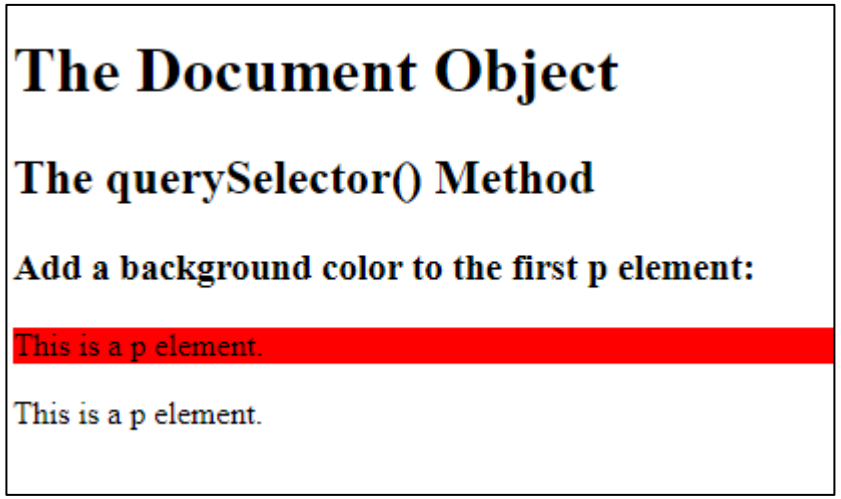




HTML DOM Document querySelector()

The querySelector() method returns the first element that matches a CSS selector.

```
<html>
<body>
<h1>The Document Object</h1>
<h2>The querySelector() Method</h2>

<h3>Add a background color to the first p element:</h3>
<p>This is a p element.</p>
<p>This is a p element.</p>

<script>
document.querySelector("p").style.backgroundColor = "red";
</script>

</body>
</html>
```



HTML DOM Element classList: The classList property returns the CSS classnames of an element.

```
<style>
.myStyle {
  background-color: coral;
  padding: 16px;
}
</style>

<body>
<h1>The DOMToken Object</h1>
<h2>The toggle() Method</h2>

<button onclick="myFunction()">Toggle</button>
<p>Click "Toggle" to toggle "myStyle" on and off.</p>

<div id="myDIV">
<p>I am myDIV.</p>
</div>

<script>
function myFunction() {
  document.getElementById("myDIV").classList.toggle("myStyle");
}
</script>
```

Tab Menu:

Tabs are used to navigate and display different content around the website. We use tabs to manage space and to make the website more attractive.

```
<html><head><style>
/* Style the tab */
.tab {
  overflow: hidden;----------------------------     With the hidden value, the overflow is clipped, and the rest of the content is hidden:
  border: 1px solid #ccc;
  background-color: #f1f1f1;
}

/* Style the buttons inside the tab */
.tab button {
  background-color: inherit;----------The inherit keyword specifies that a property should inherit its value from its parent element.
  float: left;
  border: none;
  outline: none;
  cursor: pointer;----------------------The cursor is a pointer and indicates a link
  padding: 14px 16px;
  font-size: 17px;
}
```

```
/* Style the tab content */
.tabcontent {
  display: none;
  padding: 6px 12px;
  border: 1px solid #ccc;
  border-top: none;
}
</style>
</head>
<body>
```

```
<h2>Tabs</h2>
<p>Click on the buttons inside the tabbed menu:</p>

<div class="tab">
  <button class="tablinks" onclick="openCity(event, 'London')">London</button>
  <button class="tablinks" onclick="openCity(event, 'Paris')">Paris</button>
  <button class="tablinks" onclick="openCity(event, 'Tokyo')">Tokyo</button>
</div>

<div id="London" class="tabcontent">
  <h3>London</h3>
  <p>London is the capital city of England.</p>
</div>

<div id="Paris" class="tabcontent">
  <h3>Paris</h3>
  <p>Paris is the capital of France.</p>
</div>

<div id="Tokyo" class="tabcontent">
  <h3>Tokyo</h3>
  <p>Tokyo is the capital of Japan.</p>
</div>

<script>
```

```
function openCity(event, cityName) {
  var i, tabcontent;
  tabcontent = document.getElementsByClassName("tabcontent");
  for (i = 0; i < tabcontent.length; i++)
 {
    tabcontent[i].style.display = "none";
  }
    document.getElementById(cityName).style.display = "block";

}
</script>

</body>
</html>
```

Protecting web page

- ✓ JavaScript is designed as an open scripting language. It is not intended to replace proper security measures, and should never be used in place of proper encryption.

- ✓ JavaScript has its own security model, but this is not designed to protect the Web site owner or the data passed between the browser and the server.

- ✓ The security model is designed to protect the user from malicious Web sites, and as a result, it enforces strict limits on what the page author is allowed to do.

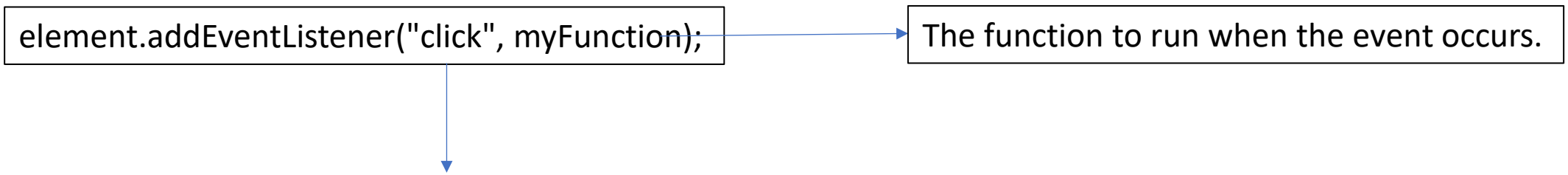
- ✓ They may have control over their own page inside the browser, but that is where their abilities end.

- JavaScripts cannot read or write files on users' computers (File API - not currently supported by many browsers - allows files to be read by scripts if the user specifically chooses to allow it), though they can cause the browser to load remote pages and resources like scripts or images, which the browser may choose to cache locally.
- They are allowed to interact with other pages in a frameset, if those frames originate from the same Web site, but not if they originate from another Web site
- JavaScript cannot be used to set the value attribute of a file input, and will not be allowed to use them to upload files without permission.
- JavaScript cannot read what locations a user has visited by reading them from the location object, although it can tell the browser to jump back or forward any number of steps through the browser history. It cannot see what other Web pages the user has open.
- JavaScript cannot access the cookies or variables from other sites.
- It cannot see when the user interacts with other programs, or other parts of the browser window.
- It cannot open windows out of sight from the user or too small for the user to see, and in most browsers, it cannot close windows that it did not open.

addEventListener()

The addEventListener() method attaches an event handler to an element.

```
<h1>The Document Object</h1>
<h2>The addEventListener() Method</h2>

<p>Click anywhere in the document to display "Hello World!".</p>

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

<script>
document.addEventListener("click", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>
```



preventDefault() Event Method

- The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

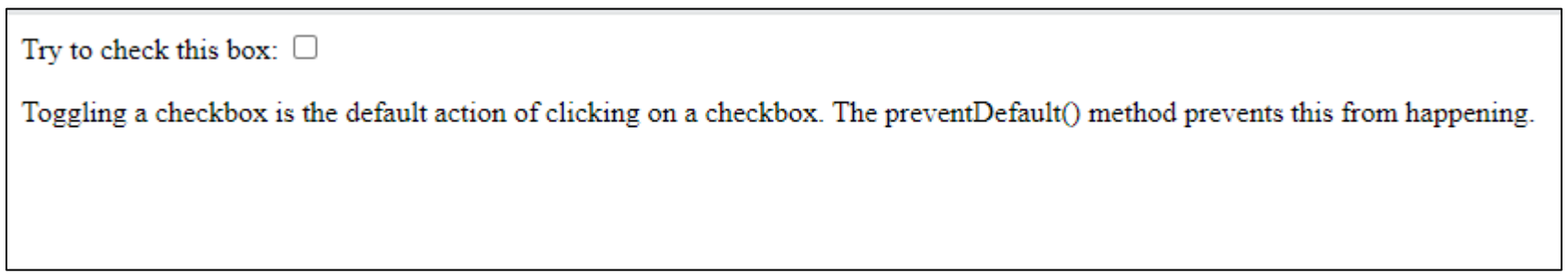
For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

```
<a id="myAnchor" href="https://w3schools.com/">Go to W3Schools.com</a>

<p>The preventDefault() method will prevent the link above from following the URL.</p>

<script>
document.getElementById("myAnchor").addEventListener("click", function(event){
  event.preventDefault()
});
</script>
```

Example 2

```
<html>
<body>
Try to check this box: <input type="checkbox" id="myCheckbox">
<p>Toggling a checkbox is the default action of clicking on a checkbox. The preventDefault() method
prevents this from happening.</p>
<script>
document.getElementById("myCheckbox").addEventListener("click", function(event){
  event.preventDefault()
});
</script>
</body>
</html>
```



Disabling the right mouse button:

```
<html>
<head>
  <title>Example</title>
</head>
<body>
  <h2>Disabling right-clicking on a webpage using JavaScript</h2>
  <p> Right Click is disabled</p>
  <script>
    document.addEventListener("contextmenu", (event) => {
      event.preventDefault();
    });
  </script>
</body>
</html>
```

Step 1 − Create an HTML page with a script element.

Step 2 − In the script element, use the addEventListener method to attach an event listener function to the contextmenu event.

Step 3 − In the event listener function, use the preventDefault method to cancel the default action of the event.

The above JavaScript code uses the addEventListener method to attach a contextmenu event listener to the document object. The event listener function cancels the default behavior of the right-click by calling the preventDefault method of the event object. This will disable right−clicking on the page and prevent the context menu from appearing.

concealing email address Hide the Email Id in a form using HTML and JavaScript

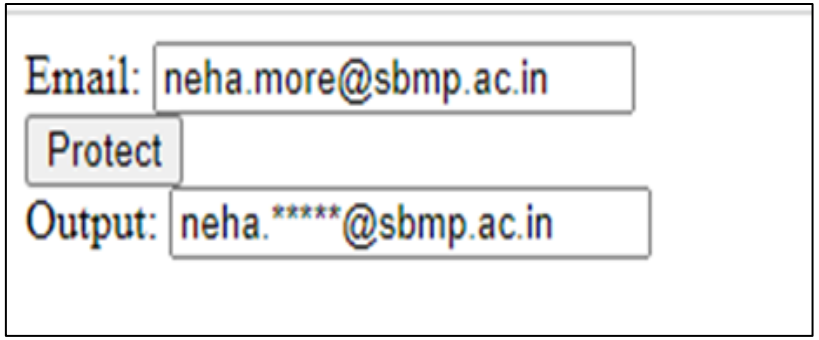
1.Input field: We are going to enter the email id in this field.
2.Button: On clicking this button hidden email id will be displayed.
3.Output field: Field to display hidden email id.

Examples:
Input :robin_singh@gmail.com
Output :robin*****@gmail.com

Input :geeksforgeeks@gmail.com
Output :geeks*****@gmail.com

Concealingemail

```
<html>
    <head>
      <title></title>
      <script type="text/javascript">
        function test_str(){
          var str = document.getElementById("t1").value ;
          var idx = str.indexOf('@');
          var res = str.replace(str.slice(5, idx), "*".repeat(5));
          document.getElementById("t2").value = res;
        }
      </script>
    </head>
    <body>
      <p>
        Email: <input type="text" placeholder="abc" id="t1"/><br/>
        <input type="button" value="Protect" onclick="test_str()"/><br/>
        Output: <input type="text" id="t2" readonly/>
      </p>
    </body>
 </html>
```



JavaScript String repeat()

The repeat() method returns a string with a number of copies of a string.

```
let text = "Hello world!";
let result = text.repeat(4);
```

Hello world!Hello world!Hello world!Hello world!

JavaScript frameworks

- JavaScript frameworks are a collection of libraries containing code written in JavaScript, making life a lot easier for software developers.
- Each JavaScript framework offers pre-built codes for different areas and different purposes in software development, saving time for the developer.

JavaScript frameworks

JavaScript frameworks provide structure for your code, so in that sense, they're pretty essential to your programming. There are many useful JavaScript frameworks that developers regularly use, and we will cover some of these in this article. A JavaScript framework provides a blueprint, so you have a guide to follow rather than having to start coding your website from scratch.

Most frameworks are open-source, meaning they are constantly being improved by the community that uses them, so they are always up to date. They are by no means set in stone either; you are free to tweak the framework you choose to suit your own website or application.

Why use a JavaScript framework?

- Developers created JavaScript frameworks to make life easier for themselves. They allow programmers to use the most up-to-date JavaScript features and tools without having to go through the arduous task of coding them from scratch by themselves.
- These frameworks are templates that provide a foundation for software applications. It collects shared resources like libraries, reference documents, images and more and packages them for developers to use. With these frameworks, programmers can add better functionality and more to a web page and website.

Popular JavaScript Framework libraries

AngularJS

AngularJS is an open-source framework that came into being in October 2010 and is the oldest available. It's a good one to choose when you're thinking about which framework would be best to go for; brilliantly, it is supported by Google! There are even apps built into cars made by General Motors that have been developed in Angular – it has emerged as a bit of a market leader in JavaScript frameworks. Google's lead Angular developer, Igor Minar, believes that Angular is the most widely used framework because it, more so than others, encourages regular updates and developments.

FRONT-END FRAMEWORKS

REACT

React.js is an efficient and flexible JavaScript library for building user interfaces created by Facebook. Technically, React is a JS library, but it is often discussed as a web framework and is compared to any other open source JavaScript framework.

React makes it easy to create interactive user interfaces because it has predictable JavaScript code that is easy to debug. Furthermore, it provides a REACT component system where blocks of JavaScript code can be written once and reused repeatedly in different parts of the application or even other applications.

ANGULAR

AngularJS is a popular enterprise-level JavaScript framework used for developing large and complex business applications. It is an open-source web framework created by Google and supported by both Google and Microsoft.

VUE

Vue.js is a progressive framework for building user interfaces. It is an up-and-coming framework that helps developers in integrating with other libraries and existing projects. It has an ecosystem of libraries that allow developers to create complex and solid single-page applications.

BACK-END FRAMEWORKS

EXPRESS

Express.js is a flexible, minimalistic, lightweight, and well-supported framework for Node.js applications. It is likely the most popular framework for server-side Node.js applications. Express provides a wide range of HTTP utilities, as well as high-performance speed. It is great for developing a simple, single-page application that can handle multiple requests at the same time.

NEXT.JS

Next.js is a minimalistic framework that allows a a JavaScript developer to create a server-side rendering and static web applications using React.js. It is one of the newest and hottest frameworks that takes pride in its ease of use. Many of the problems developers experience while building applications using React.js are solved using Next.js. It has many important features included “out of the box,” and makes development a JavaScript breeze.