

# **Full Stack Module-II**

**Manual V8.3**

**ANUDIP FOUNDATION**

---



## ICONS AND THEIR MEANING



### HINTS:

*Get ready for helpful insites on difficult topics and questions.*



### STUDENTS:

*This icon symbolize important instrcutions and guides for the students.*



### TEACHERS/TRAINERS:

*This icon symbolize important instrcutions and guides for the trainers.*

Lesson No	Lesson Name	Practical Duration (Minutes)	Theory Duration (Minutes)	Page No
1	Introduction to JavaScript	60	60	03
2	The Essential JavaScript Statements	60	60	07
3	JavaScript Language	60	60	14
4	Working with Predefined Core Objects	60	60	19
5	Working with arrays	60	60	23
6	Document Object Model	60	60	29
7	Working With Document Object	60	60	34
8	Working with Form Object	60	60	39
9	Work with Regular Expressions	60	60	45
10	Closures, Callbacks, and Recursion	60	60	50
11	Advance JavaScript	60	60	54
12	Ajax Development	60	60	58
13	JavaScript Best Practices	60	60	63
14	Test and Debug a JavaScript Application	60	60	67
15	ES6 Features	60	60	71
16	Bootstrap Components with JavaScript	60	60	77

**Total Duration:** \_\_\_\_Hours

**Lesson 1: Introduction to JavaScript (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Basics</li><li>• Syntax</li><li>• Data types</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**Basics**

JavaScript is a scripting programming language that allows you to implement complex things on web pages.

JavaScript makes it easy to display timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, and lot more.

**Common features of JavaScript**

- Store useful values inside variables.
- Operations on pieces of text (known as "strings" in programming).
- Running code in response to certain events occurring on a web page.
- JavaScript is an object-based scripting language.
- Giving the user more control over the browser.
- Handling dates and time.
- Detecting the user's browser and OS,
- It is light weighted.
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter based scripting language.
- JavaScript is case sensitive.
- JavaScript is object based language as it provides predefined objects.
- Every statement in javascript must be terminated with semicolon (;).
- Most of the javascript control statements syntax is same as syntax of control statements in C language.
- An important part of JavaScript is the ability to create new functions within scripts. Declare a function in JavaScript using function keyword.

**Limitations**

- Client-side JavaScript does not allow the reading or writing of files.

- It cannot be used for networking applications because there is no such support available.
- It doesn't have any multithreading or multiprocessor capabilities.

## Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed. Have a look below:

```
var x, y, z;           // How to declare variables
x = 5; y = 6;          // How to assign values
z = x + y;              // How to compute values
```

The JavaScript syntax defines two types of values:

1. Fixed values
2. Variable values

Fixed values are called literals. Variable values are called variables.

## Data Types

Data types are used to classify one particular type of data in programming languages. For instance, a number and a string of characters are different types of data that will be treated differently by JavaScript.

This is important because the specific data type you use will determine what values you can assign to it and what you can do to it. This is to say, to be able to do operations with variables in JavaScript, it is important to understand the data type of any given variable.

There are basically 7 data types in JavaScript:

1. Number
2. Strong
3. Boolean
4. Null
5. Undefined
6. Objects
7. Symbols

## A sample JavaScript program (output on the right hand side)

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

<h2>My First JavaScript</h2>

<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>

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

</body>
</html>
```

### My First JavaScript

Click me to display Date and Time.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a program that embeds JavaScript code within to produce an output “Hello World”

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Embedding JavaScript</title>
</head>
<body>
<script>
var greet = "Hello World!";
document.write(greet); // Prints: Hello World!
</script>
</body>
</html>
```

### 2. Write a program that displays your name as the output.

### 3. Write a program that displays “The Capital Of India Is New Delhi” as the output.

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

This chapter introduces the readers to the basic concept of JavaScript, its features and limitation, and the data types being used.

**Testing your skills**

1. JavaScript is a \_\_\_\_\_programming language
  - a) Hyper Text Marking Language, b) scripting, c) OOP, d) None of the above
  
2. Which of the following can be displayed through JavaScript
  - a) Interactive maps, b) Timely content updates, c) scrolling video jukeboxes, d) all of the above
  
3. Which of these features are true for JavaScript
  - a) Case sensitive, b) interpreter based scripting language, c) predefined objects, d) All of the above
  
4. Which of these are data types of JavaScript
  - a) Boolean, b) Null, c) Both (a) and (b), d) None of the above
  
5. How many types of values are available with JavaScript
  - a) 2, b) 3, c) 4, d) 5

**Lesson 2: The Essential JavaScript Statements (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Different javascript statements, the syntaxs</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

Statements are used in JavaScript to control its program flow. Unlike properties, methods, and events, which are fundamentally tied to the object that owns them, statements are designed to work independently of any JavaScript object. That means you can use a statement whether you're working with the document object, the window object, the history object, or some other object.

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

As a language, JavaScript supports relatively few statements -- just enough to get by. It does, however, offer the bare minimum of statements that are needed to construct a fully-functional application.

JavaScript currently supports the following 12 statements. Have a look:

1. comment
2. break
3. continue
4. for
5. for...in
6. function
7. if...else
8. new
9. return
10. var
11. while
12. with

**Comment**

The comment character tells JavaScript that you want to include explanatory comments in your program. Comment is denoted by // symbol. You can place the // comment characters anywhere on a line. JavaScript will treat all the text on that line after the // as a comment.

Have a look below:

```
MyVariable="This is a test" // assigns text variable MyVariable
```



## Break

The break statement tells JavaScript to exit a "controlled structure" and resume execution at a point after the structure. The break statement is used with structures built using the following commands:

- for
- for...in
- while

The break statement is most commonly used to prematurely end a 'for' loop.

Let's understand this through a sample example:

```
for (Count=1; Count<=10; Count++) {  
    if (Count == 6)  
        break;  
    document.write ("<P>Loop: " + Count + "</P>");  
}
```

## Continue

The continue statement tells JavaScript to skip any instructions that may follow in a for, for...in, or while loop, and proceed with the next iteration. The most common use of the continue statement is to conditionally skip instructions in the loop but not exit the loop (as the break statement does).

Let's understand this through a sample example:

```
for (Count=1; Count<=10; Count++) {  
    if (Count == 6)  
        continue;  
    document.write ("<P>Loop: " + Count + "</P>");  
}
```

## For

The for statement repeats a block of instructions one or more times. The number of iterations is controlled by values supplied as arguments.

The syntax of the for statement is:

*for (InitVal; Test; Increment)*

InitVal is an expression that establishes the initial value and assigns that value to a variable.

Test is the expression used by the for statement to control the number of iterations of the loop.

Increment indicates how you want the for loop to count -- by ones, twos, fives, tens, and so on.

## For...in

The for...in statement is a special version of the for statement and is used to display the property names and/or property contents of objects. It is mostly handy as a debugging and testing tool.

The syntax for the for...in statement is:

```
for (var in object) {  
    statements  
}
```

var is the name of a variable

object is the object you wish to examine

statements are one or more JavaScript instructions you wish to execute for each property returned by the for...in loop

## Function

The function statement lets you create your own user-defined functions (as well as user-defined objects and methods for those objects).

Let's understand this through a sample example:

```
<HTML>  
<HEAD>  
<TITLE>Function Test</TITLE>  
<SCRIPT LANGUAGE="JavaScript">  
function writeMyName () {  
    MyName="John Doe"  
    alert (MyName)  
}  
</SCRIPT>  
</HEAD>  
<BODY>  
<FORM>  
<INPUT TYPE="button" NAME="button1" VALUE="Click Me!" onClick="writeMyName()"><P>  
</FORM>  
</BODY>  
</HTML>
```

In the above example, you can notice that the writeMyName function is defined within `<SCRIPT>...</SCRIPT>` tags. It is activated (otherwise known as "called") when the form button is pushed. This calling action is accomplished using the onClick event handler, defined in the `<INPUT>` tag for the form button.

## If...else

The if, along with its optional else, statement is used to build an "if conditional" expression. It is called a conditional expression because it tests for a specific condition. If the expression is true, the script performs the instructions following the if statement. If the expression is false, the script jumps to the instructions that follow the else statement. If there is no else statement, the script jumps past the if statement entirely and continues from there.

The syntax for if is:

*if (expression)*

## New

The new statement creates a new copy of an object. It is used in either of two ways:

1. To define a new Date object (Date is a built-in JavaScript object)
2. To define a new user-defined object

The syntax for new is:

*varname = new objectName(params);*

varname is the name of the new object.

objectName is the name of the object.

params are one or more parameters that you pass to the object function, if needed.

## Return

The return statement is used to mark the end of a function. The return statement may not be used outside of a function.

A sample example:

```
function myFunc() {  
    var OutString = "This is a test";  
    return (OutString);  
}  
  
function myFunc() {  
    OutString = "This is a test";  
    return;  
}
```

## This

The `this` keyword (it's not really a statement) refers to the current object and is shorthand for using the formal name of the object. It is typically used in one of two ways:

1. To refer to the current form or control in an event handler (such as `onClick` or `onSubmit`)
2. To define a new property in a user-defined object

A sample example:

```
<INPUT TYPE="button" NAME="button" VALUE="Click" onClick="test(this.form)" >
```

## Var

The `var` statement is used to explicitly declare a variable. The `var` statement also sets the "scope" of a variable when the variable is defined inside a function.

The syntax for `var` is:

```
var VariableName;
```

## While

The `while` statement sets up a unique repeating loop that causes the script to repeat a given set of instructions.

The syntax of the `while` statement is:

```
while (Expression) {  
    // stuff to repeat  
}
```

## With

The `with` statement is designed as a time and space saver. The `with` statement is typically used with the built-in `Math` object, as it requires you to specify the name of the object when accessing any of its properties.

The syntax for `with` is:

```
with (object) {  
    statements  
}
```

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create a Javascript program that will display "Let's Party!" if the current day is Friday**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript If Statement</title>
</head>
<body>
<script>
var now = new Date();
var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6
if (dayOfWeek == 5) {
document.write("Let's Party!");
}
</script>
<p><strong>Note:</strong> This example will print "Let's Party!" if the current day is Friday.</p>
</body>
</html>
```

**2. Create a Javascript program that will display "Time For Office" if the current day is Monday****3. Create a Javascript program that will display "It's Holiday" if the current day is Christmas**

**Note:** After saving the code, click the created HTML file to view the output.

**Reviewing the chapter**

In this chapter we learned about different JavaScript statements along with their syntaxes and also focused on some sample programming codes.

**Testing your skills**

1. JavaScript can be grouped together in code blocks, inside\_\_\_\_\_

a) {}, b) <>, c) [], d) ()

2. The break statement is used with structures built using which of the following commands

a) About, b) While, c) In To, d) If Else

3. \_\_\_\_\_ statement is used to conditionally skip instructions

a) Continue, b) Break, c) For, d) None of the above

4. The \_\_\_\_\_ statement is used to mark the end of a function

a) break, b) end, c) return, d) exit

5. The \_\_\_\_\_ statement is used to explicitly declare a variable

a) This, b) While, c) With, d) Var

**Lesson 3: JavaScript Language (120 minutes)****Objective:** After completing this lesson you will be able to :

- Variables
- Operators
- Control structures

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes**Practical Duration:** 60 minutes**Total Duration:** 120 minutes**Variables**

A JavaScript variable is simply a name of storage location. In other words, they are containers for storing data values.

There are two types of variables in JavaScript:

1. Local variable
2. Global variable

There are some rules while declaring a JavaScript variable (also known as identifiers). Have a look:

- Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( \$ ) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.

**A sample example**

```
<html>
<body>
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
</body>
</html>
```

30

**Operators**

JavaScript operators are symbols that are used to perform operations on operands.

There are 6 types of operators used in JavaScript:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Comparison operator compares the two operands.

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10===20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Bitwise operators perform bitwise operations on operands.

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
	Bitwise OR	(10==20   20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

Logical operators are used to test for true or false. They are also known as Comparison operators.



Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
	Logical OR	(10==20    20==33) = false
!	Logical Not	!(10==20) = true

Assignment operator assigns a value to its left operand based on the value of its right operand.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Special operators are a hodge-podge of miscellaneous other symbols and words that perform other and important functions.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

## Control structures

JavaScript has number of statements that control flow of the program. There are:

- conditionals (if-else, switch) that perform different actions depending on the value of an expression,
- loops (while, do-while, for, for-in, for-of), that execute other statements repetitively,
- jumps (break, continue, labeled statement) that cause a jump to another part of the program.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension. After saving the code, click the created HTML file to view the output.

**1. Create a javascript program that displays the result of different arithmetic functions like addition, subtraction, multiplication, and division**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript Arithmetic Operators</title>
</head>
<body>
<script>
var x = 10;
var y = 4;
document.write(x + y); // Prints: 14
document.write("<br>");
document.write(x - y); // Prints: 6
document.write("<br>");
document.write(x * y); // Prints: 40
document.write("<br>");
document.write(x / y); // Prints: 2.5
document.write("<br>");
</script>
</body>
</html>
```

**2. Create a javascript program that displays the sum of two digits, 24 and 58**

**3. Create a javascript program that displays the difference of two numbers, 42 and 33**

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

In this chapter, we learned about different operators, control structures, and variables.

## Testing your skills

1. There are \_\_\_\_ types of variables in JavaScript

a) 2, b) 3, c) 4, d) 5

2. Which of the following are variables in JavaScript?

a) logical, b) local, c) static, d) none of the above

3. \_\_\_\_\_ are symbols that are used to perform operations on operands

a) operators, b) variables, c) identifiers, d) none of the above

4. \_\_\_\_\_ operators are used to test for true or false

a) bitwise, b) special, c) assignment, d) logical

5) Which among these control structures perform different actions depending on the value of an expression

a) while, b) break, c) if else, d) continue

## Lesson 4: Working with Predefined Core Objects (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"> <li>• Objects and methods</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>• Computer With Windows XP and above</li> <li>• Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

Do remember that JavaScript is template based and not class based. Here, we don't create class to get the object. But, we direct create objects.

### Syntax to create an object

*object={property1:value1,property2:value2.....propertyN:valueN}*

A sample example (observe the outcome at the right hand side)

<pre> &lt;html&gt; &lt;body&gt; &lt;script&gt; emp={id:102,name:"Shyam Kumar",salary:40000} document.write(emp.id+" "+emp.name+" "+emp.salary); &lt;/script&gt; &lt;/body&gt; &lt;/html&gt; </pre>	102 Shyam Kumar 40000
--	-----------------------

### Object Properties

The *name:values* pairs in JavaScript objects are called properties. Accessing object properties can be done in either of the two ways:

1. `objectName.propertyName`
2. `objectName["propertyName"]`

### The role of methods in JavaScript objects

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method. Methods are actions that can be performed on objects. They are stored in properties as function definitions.

You access an object method with the following syntax:

*objectName.methodName()*

Let's focus on a sample example:

<pre> &lt;html&gt; &lt;body&gt; &lt;script&gt; function emp(id,name,salary){ this.id=id; this.name=name; this.salary=salary;  this.changeSalary=changeSalary; function changeSalary(otherSalary){ this.salary=otherSalary; } } e=new emp(103,"Sonoo Jaiswal",30000); document.write(e.id+" "+e.name+" "+e.salary); e.changeSalary(45000); document.write("&lt;br&gt;" +e.id+" "+e.name+" "+e.salary); &lt;/script&gt; &lt;/body&gt; &lt;/html&gt; </pre>	<pre> 103 Sonoo Jaiswal 30000 103 Sonoo Jaiswal 45000 </pre>
--	--

## A look at different object methods used in JavaScript

S.No	Methods	Description
1	<code>Object.assign()</code>	This method is used to copy enumerable and own properties from a source object to a target object
2	<code>Object.create()</code>	This method is used to create a new object with the specified prototype object and properties.
3	<code>Object.defineProperty()</code>	This method is used to describe some behavioral attributes of the property.
4	<code>Object.defineProperties()</code>	This method is used to create or configure multiple object properties.
5	<code>Object.entries()</code>	This method returns an array with arrays of the key, value pairs.
6	<code>Object.freeze()</code>	This method prevents existing properties from being removed.
7	<code>Object.getOwnPropertyDescriptor()</code>	This method returns a property descriptor for the specified property of the specified object.
8	<code>Object.getOwnPropertyDescriptors()</code>	This method returns all own property descriptors of a given object.
9	<code>Object.getOwnPropertyNames()</code>	This method returns an array of all properties (enumerable or not) found.
10	<code>Object.getOwnPropertySymbols()</code>	This method returns an array of all own symbol key properties.
11	<code>Object.getPrototypeOf()</code>	This method returns the prototype of the specified object.
12	<code>Object.is()</code>	This method determines whether two values are the same value.
13	<code>Object.isExtensible()</code>	This method determines if an object is extensible
14	<code>Object.isFrozen()</code>	This method determines if an object was frozen.
15	<code>Object.isSealed()</code>	This method determines if an object is sealed.
16	<code>Object.keys()</code>	This method returns an array of a given object's own property names.
17	<code>Object.preventExtensions()</code>	This method is used to prevent any extensions of an object.
18	<code>Object.seal()</code>	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	<code>Object.setPrototypeOf()</code>	This method sets the prototype of a specified object to another object.
20	<code>Object.values()</code>	This method returns an array of values.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create a Javascript Object that will display the name "Mark Strong"**

```
<!DOCTYPE html>

<html>

<body>

<p>Creating a JavaScript Object.</p>

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

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

document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;
</script>

</body>
</html>
```

**2. Create a program that uses a method to access the object**

**3. Create a program that will display the country name "Latvia"**

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

In this chapter, we learned about the use of objects and methods in JavaScript along with sample examples to have a better understanding of the same.

## Testing your skills

1. JavaScript is \_\_\_\_\_ based

a) template, b class, c) object, d) method

2. This method prevents existing properties from being removed.

a) Object.is(), b) Object.assign(), c) Object.freeze(), d) Object.keys()

3. This method determines whether two values are the same value.

a) Object.is(), b) Object.assign(), c) Object.freeze(), d) Object.keys()

4. This method returns an array of values.

a) Object.arrays(), b) Object.values(), c) Object.array(), d) Object.value()

5. This method is used to describe some behavioral attributes of the property.

a) Object.defineProperty(), b) Object.defineProperties(), c) Object.getOwnPropertyNames(), d) None

**Lesson 5: Working with arrays (120 minutes)****Objective:** After completing this lesson you will be able to :

- Array, object, property, method

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes**Practical Duration:** 60 minutes**Total Duration:** 120 minutes

An array is a special variable, which can hold more than one value at a time. Arrays in JavaScript are used for storing multiple values in a single variable.

Say for example, you have a list of names. Storing these names in single variables could look like this:

```
var name1 = "Jack";
```

```
var name2 = "Amit";
```

```
var name3 = "Kriti";
```

Now, what if there is not three but three hundred names to work with and you need to loop through the names to search for a specific one?

This is when Array makes the entry to handle the situation in an easier manner. An array can hold many values under a single name, and you can access the values by referring to an index number.

**Array syntax**

```
var array_name = [item1, item2, ...];
```

**A sample example**

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

<h2>JavaScript Arrays</h2>

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

<script>
var name = ["Jack", " Amit", " Kriti"];
document.getElementById("demo").innerHTML = name;
</script>

</body>
</html>
```

**JavaScript Arrays**

Jack, Amit, Kriti



## Constructing array

There are basically 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

### The syntax of creating array using array literal

```
var arrayname=[value1,value2.....valueN];
```

A sample example

<pre>&lt;html&gt; &lt;body&gt; &lt;script&gt; var emp=["Sonoo","Vimal","Ratan"]; for (i=0;i&lt;emp.length;i++){ document.write(emp[i] + "&lt;br/&gt;"); } &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	Sonoo Vimal Ratan
---	-------------------------

### The syntax of creating instance of an array directly using new keyword

```
var arrayname=new Array();
```

A sample example

<pre>&lt;html&gt; &lt;body&gt; &lt;script&gt; var i; var emp = new Array(); emp[0] = "Arun"; emp[1] = "Varun"; emp[2] = "John";  for (i=0;i&lt;emp.length;i++){ document.write(emp[i] + "&lt;br/&gt;"); } &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	Arun Varun John
---	-----------------------

### By using an Array constructor (using new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

<pre>&lt;html&gt; &lt;body&gt; &lt;script&gt; var emp=new Array("Jai","Vijay","Smith"); for (i=0;i&lt;emp.length;i++){ document.write(emp[i] + "&lt;br/&gt;"); } &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	Jai Vijay Smith
--	-----------------------

## Array methods used in JavaScript

Methods	Description
<code>concat()</code>	It returns a new array object that contains two or more merged arrays.
<code>copyWithin()</code>	It copies the part of the given array with its own elements and returns the modified array.
<code>every()</code>	It determines whether all the elements of an array are satisfying the provided function conditions.
<code>fill()</code>	It fills elements into an array with static values.
<code>filter()</code>	It returns the new array containing the elements that pass the provided function conditions.
<code>find()</code>	It returns the value of the first element in the given array that satisfies the specified condition.
<code>findIndex()</code>	It returns the index value of the first element in the given array that satisfies the specified condition.
<code>forEach()</code>	It invokes the provided function once for each element of an array.
<code>includes()</code>	It checks whether the given array contains the specified element.
<code>indexOf()</code>	It searches the specified element in the given array and returns the index of the first match.
<code>join()</code>	It joins the elements of an array as a string.
<code>lastIndexOf()</code>	It searches the specified element in the given array and returns the index of the last match.
<code>map()</code>	It calls the specified function for every array element and returns the new array
<code>pop()</code>	It removes and returns the last element of an array.
<code>push()</code>	It adds one or more elements to the end of an array.
<code>reverse()</code>	It reverses the elements of given array.
<code>shift()</code>	It removes and returns the first element of an array.
<code>slice()</code>	It returns a new array containing the copy of the part of the given array.
<code>sort()</code>	It returns the element of the given array in a sorted order.
<code>splice()</code>	It add/remove elements to/from the given array.
<code>unshift()</code>	It adds one or more elements in the beginning of the given array.

## Some essential Array methods in JavaScript

### toString()

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

<pre> &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;h2&gt;JavaScript Array Methods&lt;/h2&gt;  &lt;h2&gt;toString()&lt;/h2&gt;  &lt;p&gt;The toString() method returns an array as a comma separated string:&lt;/p&gt; &lt;p id="demo"&gt;&lt;/p&gt;  &lt;script&gt; var fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo").innerHTML = fruits.toString(); &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt; </pre>	<p><b>JavaScript Array Methods</b></p> <p><b>toString()</b></p> <p>The <code>toString()</code> method returns an array as a comma separated string:</p> <p>Banana,Orange,Apple,Mango</p>
---	--

### join()

The `join()` method joins all array elements into a string. It behaves just like `toString()`, but in addition you can specify the separator.

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

<h2>JavaScript Array Methods</h2>

<h2>join()</h2>

<p>The join() method joins array elements into a string.</p>
<p>It this example we have used " * " as a separator between the elements:</p>
<p id="demo"></p>

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

</body>
</html>
```

### JavaScript Array Methods

#### join()

The join() method joins array elements into a string.

It this example we have used " \* " as a separator between the elements:

Banana \* Orange \* Apple \* Mango

## Pop()

The pop() method removes the last element from an array

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

<h2>JavaScript Array Methods</h2>

<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>
<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

### JavaScript Array Methods

#### pop()

The pop() method removes the last element from an array.

Banana,Orange,Apple,Mango

Banana,Orange,Apple

## Push()

The push() method adds a new element to an array (at the end)

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

<h2>JavaScript Array Methods</h2>

<h2>push()</h2>

<p>The push() method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

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

function myFunction() {
  fruits.push("Kiwi");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

### JavaScript Array Methods

#### push()

The push() method appends a new element to an array.

[Try it](#)

Banana,Orange,Apple,Mango

## Properties

Properties are the values associated with a JavaScript object. They can usually be changed, added, and deleted, but some are read only.

A JavaScript object is a collection of unordered properties.

There are three Syntax styles to define a property. Have a look:

1. `objectName.property`      `// person.age`
2. `objectName["property"]`      `// person["age"]`
3. `objectName[expression]`      `// x = "age"; person[x]`

### A sample example

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;h2&gt;JavaScript Object Properties&lt;/h2&gt;  &lt;p&gt;There are two different ways to access an object property:&lt;/p&gt; &lt;p&gt;You can use .property or ["property"].&lt;/p&gt;  &lt;p id="demo"&gt;&lt;/p&gt;  &lt;script&gt; var person = {   firstname:"John",   lastname:"Doe",   age:50,   eyecolor:"blue" };  document.getElementById("demo").innerHTML = person.firstname + " is " + person.age + " years old."; &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<h4>JavaScript Object Properties</h4> <p>There are two different ways to access an object property:</p> <p>You can use .property or ["property"].</p> <p>John is 50 years old.</p>
---	--

### Property Attributes

All properties have a name. In addition they also have a value which is one of the property's attributes.

Some of the other attributes are: enumerable, configurable, and writable. These attributes define how the property can be accessed (is it readable?, is it writable?).

In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).

### Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension. After saving the code, click the created HTML file to view the output.

**1.** Create a program that displays an array of vehicle manufacturing brands like Volvo, BMW, and Audi

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>JavaScript Arrays</h2>
```

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

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

</body>
</html>
```

### Reviewing the chapter

- An array is a special variable, which can hold more than one value at a time. Arrays in JavaScript are used for storing multiple values in a single variable.
- All properties have a name. In addition they also have a value which is one of the property's attributes.
- A JavaScript object is a collection of unordered properties.

### Testing your skills

1. Which of these ways are used to construct array in JavaScript

a) array literal, b) creating instance of Array directly, c) using an Array constructor, d) all of the above

2. \_\_\_\_\_ checks whether the given array contains the specified element

a) includes(), b) indexOf(), c) filter(), d) fill()

3. \_\_\_\_\_ adds one or more elements to the end of an array

a) pop, b) push, c) join, d) map

4. \_\_\_\_\_ returns a new array object that contains two or more merged arrays

a) findIndex, b) every, c) concat, d) splice

5) \_\_\_\_\_ removes and returns the first element of an array

a) sort, b) shift, c) splice, d) unshift

**Lesson 6: Document Object Model (120 minutes)****Objective:** After completing this lesson you will be able to :

- Overview
- A look at window and navigator object

**Materials Required:**

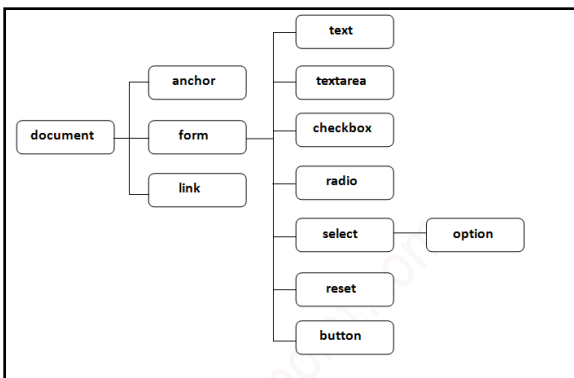
- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes**Practical Duration:** 60 minutes**Total Duration:** 120 minutes

The document object represents the whole html document. Commonly referred to as DOM, it is a programming interface for HTML and XML(Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Please keep in mind that DOM is often referred to as a Logical structure because it doesn't specify any relationship between objects.

DOM is a way to represent the webpage in the structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes or Elements using commands or methods provided by Document object.

**Common Methods of DOM**

- write("string") - writes the given string on the document.
- writeln("string") - writes the given string on the document with newline character at the end.
- getElementById() - returns the element having the given id value.
- getElementsByName() - returns all the elements having the given name value.
- getElementsByTagName() - returns all the elements having the given tag name.
- getElementsByClassName() - returns all the elements having the given class name.

## DOM Properties

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed of like tag elements with attributes in HTML.

Have a look at the following properties of DOM

- Window Object: Window Object is at always at top of hierarchy.
- Document object: When HTML document is loaded into a window, it becomes a document object.
- Form Object: It is represented by form tags.
- Link Objects: It is represented by link tags.
- Anchor Objects: It is represented by a href tags.
- Form Control Elements:: Form can have many control elements such as text fields, buttons, radio buttons, and checkboxes, etc.

## DOM Levels

### Level 0:

Provides low-level set of interfaces.

### Level 1:

DOM level 1 can be described in two parts: CORE and HTML.

CORE provides a low-level interfaces that can be used to represent any structured document.

HTML provides high-level interfaces that can be used to represent HTML document.

### Level 2:

Consist of six specifications:

1. CORE2: extends functionality of CORE specified by DOM level 1.
2. VIEWS: views allows programs to dynamically access and manipulate content of document.
3. EVENTS: Events are scripts that are either executed by browser when user reacts to web page.
4. STYLE: allows programs to dynamically access and manipulate content of style sheets.
5. TRAVERSAL: allows programs to dynamically traverse the document.
6. RANGE: allows programs to dynamically identify a range of content in document.

**Level 3:**

Consists of five different specifications:

1. CORE3: extends functionality of CORE specified by DOM level 2.
2. LOAD and SAVE: allows program to dynamically load the content of XML document into DOM document and save the DOM Document into XML document by serialization.
3. VALIDATION: allows program to dynamically update the content and structure of document while ensuring document remains valid.
4. EVENTS: extends functionality of Events specified by DOM Level 2.
5. XPATH: XPATH is a path language that can be used to access DOM tree.

**A sample example of DOM**

Using a sample example, we will try to access field value by document object

In this example, we are going to get the value of input text by user. Here, we are using `document.form1.name.value` to get the value of name field. Here, document is the root element that represents the html document, form1 is the name of the form, name is the attribute name of the input text, value is the property, that returns the value of the input text.

```
<script type="text/javascript">  
  
function printvalue(){  
  
var name=document.form1.name.value;  
  
alert("Welcome: "+name);  
  
}  
  
</script>  
  
<form name="form1">  
  
Enter Name: <input type="text" name="name"/>  
  
<input type="button" onclick="printvalue()" value="print name"/>  
  
</form>
```

Output of the above example:

The screenshot shows a web form with a label "Enter Name:" followed by a text input field. To the right of the input field is a button labeled "print name". The entire form is enclosed in a thin black border.



## Window Object

The window object represents an open window in a browser. If a document contains frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

## Navigator Object

The navigator object contains information about the browser. Do keep a note of the fact that there is no public standard that applies to the navigator object, but all major browsers support it.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a program that changes the content (the innerHTML) of the <p> element with id="demo"

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Page</h2>

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

<script>

document.getElementById("demo").innerHTML = "Hello World!";

</script>

</body>

</html>
```

### 2. Create a program that retrieves the text of an <h1> element and copies it into a <p> element

### 3. Create a program that changes the content (the innerHTML) of the <p> element with id="archive"

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- The document object represents the whole html document. Commonly referred to as DOM, it is a programming interface for HTML and XML(Extensible markup language) documents.
- The window object represents an open window in a browser
- The navigator object contains information about the browser.
- With DOM, we can easily access and manipulate tags, IDs, classes, Attributes or Elements using commands or methods provided by Document object.

## Testing your skills

1. Which of the followings can be manipulated through DOM?

a) Attribute, b) ID, c) Tag, d) All of the above

2. \_\_\_\_\_ writes the given string on the document with newline character at the end

a) writein, b) writenew, c) writenl, d) write

3. \_\_\_\_\_ object is always at top of hierarchy

a) Anchor, b) Link, c) Window, d) Form

4. In DOM Level 2, \_\_\_\_\_ are scripts that are either executed by browser when user reacts to web page

a) Views, b) Events, c) Range, d) Core2

5. \_\_\_\_\_ is represented by href tags

a) Anchor, b) Link, c) Form, d) Form Control

## Lesson 7: Working With Document Object (120 minutes)

**Objective:** After completing this lesson you will be able to :

- Link
- Anchor, and cookies object

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

**Total Duration:** 120 minutes

When an HTML document is loaded into a web browser, it becomes a document object.

The document object is the root node of the HTML document.

### Link

The links collection returns a collection of all links in the document.

The links in the collection represents `<a>` elements and/or `<area>` elements with a HREF attribute.

However, do take a note of the fact that if the element is missing the HREF attribute, nothing is returned.

Also, it is to be noted that the elements in the collection are sorted as they appear in the source code.

### Syntax

*document.links*

### A sample example

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

<img src = "planets.gif" width="145" height="126" alt="Planets" usemap = "#planetmap">
<map name="planetmap">
  <area shape="rect" coords="0,0,82,126" href="sun.htm" alt="Sun">
  <area shape="circle" coords="90,58,3" href="mercur.htm" alt="Mercury">
  <area shape="circle" coords="124,58,8" href="venus.htm" alt="Venus">
</map>

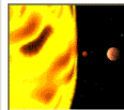
<p>
  <a href="/html/default.asp">HTML</a><br>
  <a href="/css/default.asp">CSS</a>
</p>

<p>Click the button to display the number of links in the document.</p>
<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var x = document.links.length;
  document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```



[HTML](#)  
[CSS](#)

Click the button to display the number of links in the document.

5

## Anchor

The anchors collection returns a collection of all `<a>` elements in the document that have a name attribute. However, keep in mind that the name attribute of the `<a>` element is not supported in HTML5. The elements in the collection are sorted as they appear in the source code.

## Syntax

`document.anchors`

## A sample example

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;a name="html"&gt;HTML Tutorial&lt;/a&gt;&lt;br&gt; &lt;a name="css"&gt;CSS Tutorial&lt;/a&gt;&lt;br&gt; &lt;a name="xml"&gt;XML Tutorial&lt;/a&gt;&lt;br&gt;  &lt;p&gt;Click the button to display the number of anchors in the document.&lt;/p&gt;  &lt;button onclick="myFunction()"&gt;Try it&lt;/button&gt;  &lt;p id="demo"&gt;&lt;/p&gt;  &lt;script&gt; function myFunction() {   var x = document.anchors.length;   document.getElementById("demo").innerHTML = x; } &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<p>HTML Tutorial CSS Tutorial XML Tutorial</p> <p>Click the button to display the number of anchors in the document.</p> <p><input type="button" value="Try it"/></p> <p>3</p>
---	--

## Anchor Object

The Anchor object represents an HTML `<a>` element. You can access an `<a>` element by using `getElementById()`.

## A sample example

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;h3&gt;A demonstration of how to access an A element&lt;/h3&gt;  &lt;a id="myAnchor" href="https://www.sampletutorial.com"&gt;Tutorials&lt;/a&gt;  &lt;p&gt;Click the button to get the URL of the link.&lt;/p&gt;  &lt;button onclick="myFunction()"&gt;Try it&lt;/button&gt;  &lt;p id="demo"&gt;&lt;/p&gt;  &lt;script&gt; function myFunction() {   var x = document.getElementById("myAnchor").href;   document.getElementById("demo").innerHTML = x; } &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<p><b>A demonstration of how to access an A element</b></p> <p><a href="https://www.sampletutorial.com">Tutorials</a></p> <p>Click the button to get the URL of the link.</p> <p><input type="button" value="Try it"/></p> <p>https://www.sampletutorial.com/</p>
--	---

## Creating an Anchor Object

You can create an `<a>` element by using the `document.createElement()` method.

A sample example

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;p&gt;Click the button to create an A element with a link to sampletutorial.com.&lt;/p&gt;  &lt;button onclick="myFunction()"&gt;Try it&lt;/button&gt;  &lt;script&gt; function myFunction() {   var x = document.createElement("A");   var t = document.createTextNode("Tutorials");   x.setAttribute("href", "https://www.w3schools.com");   x.appendChild(t);   document.body.appendChild(x); } &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>Click the button to create an A element with a link to sampletutorial.com.</p> <p><input type="button" value="Try it"/> <a href="https://www.w3schools.com">Tutorials</a></p>
---	--

## Cookies

Cookies are small items of data, each consisting of a name and a value, stored on behalf of a website by visitors' web browsers.

In JavaScript, cookies can be accessed through the `document.cookie` object, but the interface provided by this object is very primitive. Cookies.js is a JavaScript object that allows cookies to be created, retrieved, and deleted through a simple and intuitive interface.

Cookies can be created using the `set` function, which takes the cookie name and value as parameters:

```
Cookies.set('theme', 'green');
```

Every cookie is accessible only within a particular path, which defaults to the current directory.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a program that gets the HTML content of the first `<a>` element in the document

```
<!DOCTYPE html>

<html>

<body>

<a name="html">HTML Tutorial</a><br>
<a name="css">CSS Tutorial</a><br>
<a name="xml">XML Tutorial</a><br>

<p>Click the button to display the HTML content of the first anchor in the document.</p>
```

```
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
var x = document.anchors[0].innerHTML;
document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

**2. Create a program that retrieves the cookie associated with the present document**

**3. Create a program that gets the HTML content of the second <a> element in the document**

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- When an HTML document is loaded into a web browser, it becomes a document object.
- The document object is the root node of the HTML document.
- The links collection returns a collection of all links in the document.
- The anchors collection returns a collection of all <a> elements in the document that have a name attribute.
- Cookies are small items of data, each consisting of a name and a value, stored on behalf of a website by visitors' web browsers.

### Testing your skills

1. The\_\_\_\_\_ is the root node of the HTML document
  - a) Form object, b) document object, c) Label object, d) <none of the above

2. The \_\_\_\_\_ collection returns a collection of all <a> elements in the document that have a name attribute

a) Anchors, b) Links, c) Name, d) Attribute

3. \_\_\_\_\_ represents an HTML <a> element

a) Link, b) Anchor collection, c) Anchor object, d) none of the above

4. Cookies can be accessed through the \_\_\_\_\_ object

a) cookie.doc, b) object.cookies.doc, c) document.cookie, d) document.object

5. Cookies can be created using the \_\_\_\_\_ function

a) create, b) set, c) get, d) createcookie

**Lesson 8: Working with Form Object (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>Form Object Properties, Methods &amp; Event Handlers</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>Computer With Windows XP and above</li><li>Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

Form object is a Browser object of JavaScript used to access an HTML form.

If a user wants to access all forms within a document then he can use the forms array. The form object is actually a property of document object that is uniquely created by the browser for each form present in a document.

The properties and methods associated with form object are used to access the form fields, attributes and controls associated with forms.

**Properties of Form Object**

- action
- elements[]
- encoding
- length
- method
- name
- target
- button
- checkbox
- FileUpload
- hidden
- password
- radio
- reset
- select
- submit
- text
- textarea

**action:**

The action property of form object is used to access the action attribute present in HTML associated with the <form> tag. This property is a read or write property and its value is a string.



**elements[]:**

The elements property of form object is an array used to access any element of the form. It contains all fields and controls present in the form. The user can access any element associated with the form by using the looping concept on the elements array.

**encoding:**

The encoding property of a form object is used to access the enctype attribute present in HTML associated with the <form> tag. This property is a read or write property and its value is a string. This property helps determine the way of encoding the form data.

**length:**

The length property of form object is used to specify the number of elements in the form. This denotes the length of the elements array associated with the form.

**method:**

The method property of form object is used to access the method attribute present in HTML associated with the <form> tag. This property is a read or write property and its value is a string. This property helps determine the method by which the form is submitted.

**name:**

The name property of form object denotes the form name.

**target:**

The target property of form object is used to access the target attribute present in HTML associated with the <form> tag. This property denotes the name of the target window to which form it is to be submitted into.

**button:**

The button property of form object denotes the button GUI control placed in the form.

**checkbox:**

The checkbox property of form object denotes the checkbox field placed in the form.

**FileUpload:**

The FileUpload property of form object denotes the file upload field placed in the form..

**hidden:**

The hidden property of form object denotes the hidden field placed in the form.

**password:**

The password property of form object denotes the object that is placed as a password field in the form.

**radio:**

The radio property of form object denotes the radio button field placed in the form.

**reset:**

As the name implies, the reset property of form object denotes the object placed as reset button in the form.

**select:**

The select property of form object denotes the selection list object placed in the form.

**submit:**

The submit property of form object denotes the submit button field that is placed in the form.

**text:**

The text property of form object denotes the text field placed in the form.

**textarea:**

The textarea property of form object denotes the text area field placed in the form.

A sample example to understand the form properties:

```
<FORM NAME="exforsys" ACTION="" METHOD="GET">
```

```
Input Values:<BR>
```

```
<INPUT TYPE="text" NAME="test" VALUE=""><P>
```

```
<INPUT TYPE="button" NAME="example" VALUE="Click" onClick="testfunc(this.form)">
```

```
</FORM>
```

In the above example, FORM NAME="exforsys" creates and denotes the name of the form. The user can refer to the form in his or her code of JavaScript by the name exforsys. The form name given must follow the naming conventions or rules of JavaScript's variable or function naming rules.

The next word, ACTION="" tells the way the user wants the browser to handle the form when it is submitted to a CGI program running on the server. It can also be left as not specified with any of the above properties. This means the URL for the CGI program is then omitted.

The METHOD="GET" defines that the method data is passed to the server when the form is submitted.

INPUT TYPE="text" defines the object used as a text box with the name of the text box as a test with no initial value specified.

INPUT TYPE="button" defines the object used as a button with the name of the button object as an example. The value is specified as Click and when the button is clicked, the onClick event handler fires and the function associated with it (testfunc(this.form)) is called.

### Methods of Form Object

- reset()
- submit()
- handleEvent()

#### reset():

The reset() method of form object is used to reset a form.

#### submit():

The submit() method of form object is used to submit a form.

#### handleEvent():

The handleEvent() method of form object is used to start or invoke a form's event handler for a specified event.

### Event Handlers

Event handlers are extremely powerful and useful JavaScript codes that are not added inside the <script> tags, but rather, inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc.

The basic syntax of these event handlers is:

*name\_of\_handler="JavaScript code here"*

**Sample example:**

```
<a href="http://google.com" onClick="alert('hello!')">Google</a>
```

As you can, this is certainly unlike a regular JavaScript code in that here we're inserting it directly inside a HTML tag, via the onClick event handler. When the above link is clicked, the user will first see an alert message before being taken to Google.

Below are some of the most commonly used event handlers supported by JavaScript:

- **onclick:** Use this to invoke JavaScript upon clicking (a link, or form boxes)
- **onload:** Use this to invoke JavaScript after the page or an image has finished loading.
- **onmouseover:** Use this to invoke JavaScript if the mouse passes by some link
- **onmouseout:** Use this to invoke JavaScript if the mouse goes pass some link
- **onunload:** Use this to invoke JavaScript right after someone leaves this page.

**Practical Session**

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create a validation form using Javascript**

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
var x = document.forms["myForm"]["fname"].value;
if (x == "") {
alert("Name must be filled out");
return false;
}
}
</script>
</head>
<body>
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
  Name: <input type="text" name="fname">
```

```
<input type="submit" value="Submit">
</form>
</body>
</html>
```

## 2. Create a validation form with alert "input the numbers " using Javascript

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Form object is a Browser object of JavaScript used to access an HTML form.
- The properties and methods associated with form object are used to access the form fields, attributes and controls associated with forms.
- Event handlers are extremely powerful and useful JavaScript codes that are not added inside the `<script>` tags, but rather, inside the html tags, that execute JavaScript when something happens

### Testing your skills

1. \_\_\_\_\_ property of form object denotes the text area field placed in the form  
a) text, b) textarea, c) target, d) name
2. \_\_\_\_\_ property of form object is an array used to access any element of the form  
a) text, b) encoding, c) target, d) elements
3. \_\_\_\_\_ defines that the method data is passed to the server when the form is submitted  
a) GET, b) ACTION, c) SET, d) SUBMIT
4. Which of the following are methods of Form Object?  
a) reset, b) submit, c) handleEvent, d) all of the above
5. Use this to invoke JavaScript if the mouse goes pass some link  
a) onclick, b) onmouseover, c) onmouseout, d) none of the above

**Lesson 9: Work with Regular Expressions (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• RegExp objects, special character searching</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**Regular Expression (RegExp)**

A regular expression is an object that describes a pattern of characters.

The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

**Syntax**

A regular expression could be defined with the RegExp () constructor in either of the two ways:

1. `var pattern = new RegExp(pattern, attributes);`
2. `var pattern = /pattern/attributes;`

Here is the description of the parameters –

*pattern* – A string that specifies the pattern of the regular expression or another regular expression.

*attributes* – An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multi-line matches, respectively.

**How to write a regular expression?**

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab\*c/ or /Chapter (\d+)\.d\*/. So henceforth, there are two ways to write regular expression pattern. Have a look:

**Using simple patterns**

Simple patterns are constructed of characters for which you want to find a direct match.

For example, the pattern /abc/ matches character combinations in strings only when exactly the characters 'abc' occur together and in that order. Such a match would succeed in the strings "Hi, do you know your abc's?" and "The latest airplane designs evolved from slabcraft." In both cases the match is

with the substring 'abc'. There is no match in the string 'Grab crab' because while it contains the substring 'ab c', it does not contain the exact substring 'abc'.

## Using special characters

When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, you can include special characters in the pattern.

For example, to match a single 'a' followed by zero or more 'b's followed by 'c', you'd use the pattern `/ab*c/`; the `*` after 'b' means "0 or more occurrences of the preceding item." In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'.

## The Special Characters in RegExp

- Assertions - Indicates in some way that a match is possible. Assertions include look-ahead, look-behind, and conditional expressions.
- Boundaries - Indicate the beginnings and endings of lines and words.
- Character Classes - Distinguishes kinds of characters such as, for example, distinguishing between letters and digits.
- Groups and Ranges - Indicates groups and ranges of expression characters.
- Quantifiers - Indicates numbers of characters or expressions to match.
- Unicode Property Escapes - Distinguishes based on unicode character properties, for example, upper and lower case letters, math symbols, and punctuation.

## Escaping

If you need to use any of the special characters literally (actually searching for a '\*', for instance), you must escape it by putting a backslash in front of it. For instance, to search for 'a' followed by '\*' followed by 'b', you'd use `/a*b/`—the backslash "escapes" the '\*', making it literal instead of special.

Similarly, if you're writing a regular expression literal and need to match a slash ('/'), you need to escape that (otherwise, it terminates the pattern). For instance, to search for the string `/example/` followed by one or more alphabetic characters, you'd use `/\/example\[a-z]+\i`—the backslashes before each slash make them literal.

To match a literal backslash, you need to escape the backslash.

For instance, to match the string `C:\` where 'C' can be any letter, you'd use `/[A-Z]:\\`—the first backslash escapes the one after it, so the expression searches for a single literal backslash.

If using the RegExp constructor with a string literal, remember that the backslash is an escape in string literals, so to use it in the regular expression, you need to escape it at the string literal level. `/a\*b/` and `new RegExp("a\\*b")` create the same expression, which searches for 'a' followed by a literal '\*' followed by 'b'.

If escape strings are not already part of your pattern you can add them using `String.replace`. Have a look at a sample example below:

```
function escapeRegExp(string) {  
  
    return string.replace(/[\.*+?^${}()|[\]\]/g, '\\$&'); // $& means the whole matched string  
  
}
```

The `g` after the regular expression is an option or flag that performs a global search, looking in the whole string and returning all matches.

### Methods used by Regular Expressions

- `exec` A RegExp method that executes a search for a match in a string. It returns an array of information or null on a mismatch.
- `test` A RegExp method that tests for a match in a string. It returns true or false.
- `match` A String method that returns an array containing all of the matches, including capturing groups, or null if no match is found.
- `matchAll` A String method that returns an iterator containing all of the matches, including capturing groups.
- `search` A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
- `replace` A String method that executes a search for a match in a string, and replaces the matched substring with a replacement substring.
- `split` A String method that uses a regular expression or a fixed string to break a string into an array of substrings.

### Reviewing the chapter

- A regular expression is an object that describes a pattern of characters.
- A regular expression could be defined with the `RegExp ()` constructor
- When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, you can include special characters in the pattern.



## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a program that uses a regular expression to do a case-insensitive search for "Anudip" in a string

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Regular Expressions</h2>

<p>Search a string for "Anudip", and display the position of the match: </p>

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

<script>

var str = "Visit Anudip!";

var n = str.search(/Anudip/i);

document.getElementById("demo").innerHTML = n;

</script>

</body>

</html>
```

### 2. Create a program that uses a regular expression to do a case-insensitive search for "Numerical" in a string

**Note:** After saving the code, click the created HTML file to view the output.

## Testing your skills

1. A RegExp method that executes a search for a match in a string.

a) search, b) exec, c) test, d) match

2. \_\_\_\_\_ indicate the beginnings and endings of lines and words

a) Assertions, b) boundaries, c) Quantifiers, d) none of the above

3. A string that specifies the pattern of the regular expression or another regular expression

a) Pattern, b) Attribute, c) Expression, d) Array

4. \_\_\_\_\_ indicates numbers of characters or expressions to match

a) Unicode property escape, b) Assertions, c) Quantifiers, d) none of the above

5. To break a string into an array of substrings

a) Exec, b) Test, c) replace, d) Split

**Lesson 10: Closures, Callbacks, and Recursion (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Introduction, callback, recursion</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function.

In JavaScript, closures are created every time a function is created, at function creation time.

In JavaScript, variables can belong to either of the two scopes: local or global.

Closures can be used to make global variable local or private.

A sample example:

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

<p>A function can access variables defined inside the function:</p>
<button type="button" onclick="myFunction()">Click Me!</button>

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

<script>
function myFunction() {
  var a = 4;
  document.getElementById("demo").innerHTML = a * a;
}
</script>
</body>
</html>
```

A function can access variables defined inside the function:

Click Me!

16

**The importance of closures**

Closures are important because they control what is and aren't in scope in a particular function, along with which variables are shared between sibling functions in the same containing scope. Understanding how variables and functions relate to each other is critical to understanding what's going on in your code, in both functional and object oriented programming styles.

## Using closures

To use a closure, define a function inside another function and expose it. To expose a function, return it or pass it to another function.

The inner function will have access to the variables in the outer function scope, even after the outer function has returned.

### A sample example:

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

<h2>JavaScript Closures</h2>

<p>Counting with a local variable.</p>

<button type="button" onclick="myFunction()">Count!</button>

<p id="demo">0</p>

<script>
var add = (function () {
  var counter = 0;
  return function () {counter += 1; return counter;}
})();

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

</body>
</html>
```

#### JavaScript Closures

Counting with a local variable.

Count!

1

In the above example, the variable "add" is assigned the return value of a self-invoking function. The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way "add" becomes a function. The "wonderful" part is that it can access the counter in the parent scope. This is called a JavaScript closure. It makes it possible for a function to have "private" variables.

Do keep a note that the counter is protected by the scope of the anonymous function, and can only be changed using the add function.

## Callback

In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called higher-order functions. Any function that is passed as an argument is called a callback function.

In simpler words, a callback is a function that is to be executed after another function has finished executing — hence the name 'call back'.

## Recursion

Recursion is a programming pattern that is useful in situations when a task can be naturally split into several tasks of the same kind, but simpler. In other words, when a task can be simplified into an easy action plus a simpler variant of the same task, it can be termed as recursion.

When a function solves a task, in the process it can call many other functions. A partial case of this is when a function calls itself. That's called recursion.

### Why recursion is favored?

One reason that recursion is favored in functional programming languages is that it allows for the construction of code that doesn't require setting and maintaining state with local variables.

Recursive functions are also naturally easy to test because they are easy to write in a pure manner, with a specific and consistent return value for any given input, and no side effects on external variable states.

## Practical Session

**Instructions:** Create the code.....using an editor.....and save it with HTML extension.

### 1. Create a program using a Java function where variables can be declared within the function

```
<!DOCTYPE html>

<html>

<body>

<p>A function can access variables defined inside the function:</p>

<button type="button" onclick="myFunction()">Click Me!</button>

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

<script>

function myFunction() {

var a = 4;

document.getElementById("demo").innerHTML = a * a;
```

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

## 2. Create a program using a Java function where two variables can be declared within the function

Note: After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- A closure gives you access to an outer function's scope from an inner function.
- In JavaScript, closures are created every time a function is created, at function creation time.
- Closures are important because they control what is and aren't in scope in a particular function, along with which variables are shared between sibling functions in the same containing scope.
- Any function that is passed as an argument is called a callback function.
- Recursion is a programming pattern that is useful in situations when a task can be naturally split into several tasks of the same kind, but simpler.

### Testing your skills

1. In JavaScript, \_\_\_\_\_ are created every time a function is created, at function creation time

a) Callbacks, b) Recursions, c) Array, d) Closures

2. Lexical environment refers to

a) Surrounding state, b) Concurring state, c) Dragging state, d) None of the above

3. In JavaScript, variables can belong which of the following scopes:

a) Local, b) Global, c) both (a) and (b), d) None of the above

4. \_\_\_\_\_ can be used to make global variable local or private

a) Callbacks, b) Closures, c) Recursion, d) None of the above

5. Any function that is passed as an argument is called a \_\_\_\_\_ function

a) Callback, b) Rollback, c) Closure, d) Recursive

## Lesson 11: Advance Javascript (120 minutes)

**Objective:** After completing this lesson you will be able to :

- Prototype paradigm
- Prototype inheritance

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

**Total Duration:** 120 minutes

### The Prototype property in JavaScript

The JavaScript prototype property allows you to add new properties/methods to an existing prototype. A prototype is an object from which other objects inherit properties. Every object has a prototype by default. Since prototypes are themselves objects, every prototype has a prototype as well.

### Advantages of prototype

- No matter how many objects you create, functions are loaded only once into memory.
- It allows you to override functions if required.
- When we put something on the prototype, every instance of the object shares the same code for the method. They are all using the same function instance.

### A sample code

```
function Pokemon(pokemonName, pokemonType, pokemonAttackList){
    this.name = pokemonName;
    this.type = pokemonType;
    this.attackList = pokemonAttackList;
}

var Arcanine = new Pokemon("Arcanine", "Fire", ["Ember", "Bite", "FlameThrower"]);
var Pikachu = new Pokemon("Pikachu", "Electric", ["Tail Whip", "ThunderBolt", "Agility"]);
var Blastoise = new Pokemon("Blastoise", "Water", ["Water Gun", "Hydro Cannon", "Hyper Beam"]);

Pokemon.prototype.callPokemon = function(){
    console.log(`I choose you, ${this.name}!`);
};

Pokemon.prototype.attack = function(attack_number){
    console.log(`${this.name}! use ${this.attackList[attack_number]}.`);
};

Pikachu.callPokemon();
Arcanine.callPokemon();
Blastoise.callPokemon();

Pikachu.attack(1);
Arcanine.attack(2);
Blastoise.attack(2);
```

OUTPUT

```
I choose you, Pikachu!
I choose you, Arcanine!
I choose you, Blastoise!
Pikachu! use ThunderBolt.
Arcanine! use FlameThrower.
Blastoise! use Hyper Beam.
```

If you focus on the above example, the inclusion of prototypes makes the scenario a lot more efficient. In the above example, the methods are placed on the prototype since we typically want all instances to use the same method. However, the properties are placed on the instance itself because we usually don't want all instances to share the same properties.

### **Finding a relation between Objects, Functions, Prototypes, and Inheritance**

Objects at the most basic level can be thought of a list of key/value pairs with the key always being a string and the value being... well anything else. Everything that you type in JavaScript that is not a primitive is an object. Objects make it easy to package and move data around and creating new ones is more trivial than in other object-oriented languages like Java/C#.

Functions are objects since they are not primitive. Since functions are objects they are often referred to as function-objects. A function-object is a special group of key/value pairs with specific properties for executing code and passing down values.

Prototypes are a special type of object and exist as a property on function-objects. When we try to access a key on a function-object, JavaScript will look at its prototype property to see if it's there. If not it will go up the prototype-chain to try to find it. To understand the prototype-chain, we need to learn about functions and inheritance.

At the heart of all JavaScript inheritance is the `Object` keyword, which is a function-object. All instance-objects inherit from it. When we create an object-literal, JavaScript under the hood is actually calling `new Object()`. The new object's `__proto__` will point to `Object`'s prototype. So all objects made from object-literals are actually instance-objects of `Object`. This provides us with a lot of useful methods like `hasOwnProperty`, which can tell us if a property exists on an object. If we try to access a property directly on a function-object, JavaScript will look at prototype first and then move up the chain using `__proto__` on the prototype.

Finally, there is also the option of multilevel inheritance. When we say inheritance we typically think of instance-objects returned from functions. With prototype you can also do multiple levels of inheritance and have function-objects inherit from other function-objects. All you have to do is set the child function-object's prototype to another instance of the parent function-object's prototype. Then all properties of the parent will be copied.

### **Practical Session**

**Instructions:** Create the code....using an editor....and save it with HTML extension.



### 1. Create a program that adds a new property to the constructor

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

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

<script>

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

this.nationality = "English";

}

var myFather = new Person("John", "Doe", 50, "blue");

var myMother = new Person("Sally", "Rally", 48, "green");

document.getElementById("demo").innerHTML =

"The nationality of my father is " + myFather.nationality + ". The nationality of my mother is " +

myMother.nationality;

</script>

</body>

</html>
```

### 2. Create a program using the JavaScript prototype property that will allow you to add new properties to the object constructors

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- The JavaScript prototype property allows you to add new properties/methods to an existing prototype. A prototype is an object from which other objects inherit properties.
- Prototypes are a special type of object and exist as a property on function-objects.

## Testing your skills

1. \_\_\_\_\_ is an object from which other objects inherit properties  
a) JavaScript, b) HTML, c) Prototype, d) None of the above
2. Every object has a \_\_\_\_\_ by default  
a) Prototype, b) function, c) structure, d) array
3. Which of these are features of prototype in JavaScript  
a) Overriding functions, b) functions are loaded only once in the memory, c) Both (a) and (b), d) None of the above
4. All objects made from object-literals are actually instance-objects of \_\_\_\_\_  
a) Function, b) Object, c) Property, d) Array

**Lesson 12: Ajax Development (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Basics</li><li>• Xmlhttp</li><li>• Application</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**AJAX Basics**

AJAX stands for Asynchronous JavaScript and XML.

AJAX can be defined as a group of interrelated web development techniques used on the client side and not on the server side, to create asynchronous web applications.

Using AJAX, web applications can send data to and achieve data from a server, and they can do that asynchronously or in the background. It can be done without interfering with the display or behavior of the existing page.

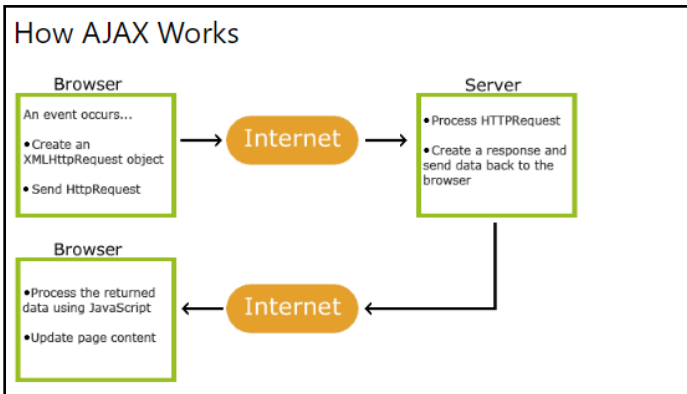
Data can be retrieved using the XMLHttpRequest object, and despite the name, the use of XML is not actually required.

**Features**

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.

- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction.
- Data-driven as opposed to page-driven.

### The working procedure of AJAX



### XMLHttpRequest

The keystone of AJAX is the *XMLHttpRequest* object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari Opera) have a built-in XMLHttpRequest object.

Syntax:

```
variable = new XMLHttpRequest();
```

### A sample application

```

<!DOCTYPE html>
<html>
<body>

<h1>The XMLHttpRequest Object</h1>

<p id="demo">Let AJAX change this text.</p>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
</body>
</html>
  
```

#### The XMLHttpRequest Object

##### AJAX

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

[Change Content](#)

## XMLHttpRequest Object Methods

Method	Description
<code>new XMLHttpRequest()</code>	Creates a new XMLHttpRequest object
<code>abort()</code>	Cancels the current request
<code>getAllResponseHeaders()</code>	Returns header information
<code>getResponseHeader()</code>	Returns specific header information
<code>open(method,url,async,user,psw)</code>	Specifies the request  <i>method</i> : the request type GET or POST <i>url</i> : the file location <i>async</i> : true (asynchronous) or false (synchronous) <i>user</i> : optional user name <i>psw</i> : optional password
<code>send()</code>	Sends the request to the server Used for GET requests
<code>send(string)</code>	Sends the request to the server. Used for POST requests
<code>setRequestHeader()</code>	Adds a label/value pair to the header to be sent

## Why AJAX is gaining so focus?

- AJAX is the most viable Rich Internet Application (RIA) technology so far. Numerous tool kit and frameworks are emerging for this technology.
- AJAX is Based on Open Standards
- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create an XMLHttpRequest object that will allow the changing of a text

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>The XMLHttpRequest Object</h2>
```

```
<p id="demo">Let AJAX change this text.</p>
```

```
<button type="button" onclick="loadDoc()">Change Content</button>
```

```
<script>
```

```
function loadDoc() {
```

```
    var xhttp = new XMLHttpRequest();
```

```
    xhttp.onreadystatechange = function() {
```

```
        if (this.readyState == 4 && this.status == 200) {
```

```
            document.getElementById("demo").innerHTML = this.responseText;
```

```
        }
```

```
    };
```

```
    xhttp.open("GET", "ajax_info.txt", true);
```

```
    xhttp.send();
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

## 2. Create an XMLHttpRequest object that will allow the changing of the text "Anudip" to "Anudip Foundation"

Note: After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- AJAX can be defined as a group of interrelated web development techniques used on the client side and not on the server side, to create asynchronous web applications.
- Ajax uses XMLHttpRequest for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.

- The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

**Testing your skills**

1. Ajax uses \_\_\_\_\_ for content

a) CSS, b) XHTML, c) DOM, d) None of the above

2. Ajax uses \_\_\_\_\_ for presentation

a) CSS, b) XHTML, c) DOM, d) None of the above

3. Which of these are features of AJAX

a) Natural user interaction b) data driven c) independent of web server software, d) All of the above

4. The keystone of AJAX is the \_\_\_\_\_ object

a) XMLHttpRequest, b) XMLHttpRequest, c) XMLHttpRequest, d) None of the above

5. \_\_\_\_\_ is used for POST requests

a) send(string), b) send, c) setRequestHeader(), d) None of the above

## Lesson 13: JavaScript Best Practices (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"> <li>• Unobtrusive JavaScript</li> <li>• Progressive Enhancement</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>• Computer With Windows XP and above</li> <li>• Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

### Unobtrusive JavaScript

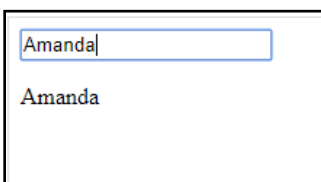
Unobtrusive JavaScript is an approach to JavaScript programming intended to facilitate progressive enhancement, separation of functionality from content/presentation, and overall best practice.

The basic concept behind unobtrusive JavaScript (also known as "unobtrusive DOM scripting") is to provide a best practice approach to JavaScript programming that maintains usability and accessibility, and degrades gracefully. Code maintainability from a programming perspective can also be considered a goal of unobtrusive JavaScript.

A sample example

<pre>&lt;!DOCTYPE html&gt; &lt;title&gt;My Example&lt;/title&gt;  &lt;!-- JavaScript --&gt; &lt;script src="/javascript/tutorial/unobtrusive_javascript_example.js"&gt;&lt;/script&gt;  &lt;!-- HTML --&gt; &lt;form action="/html/tags/html_form_tag_action.cfm"&gt;   &lt;input type="input" id="first_name" name="first_name" placeholder="First Name..."&gt;   &lt;input type="submit" id="submit" value="Submit"&gt; &lt;/form&gt;  &lt;p id="msg"&gt;&lt;/p&gt;</pre>	
---	--

In the above example, if you enter your name into the following input field, your name will appear under the input field. It will update with each letter you type. You don't need to click any buttons to make this happen. It just works. See below:





Now, if you remove the JavaScript from that example, it will still work, but in a slightly different way. Have a look below:

<pre>&lt;!DOCTYPE html&gt; &lt;title&gt;My Example&lt;/title&gt;  &lt;!-- We've removed the JavaScript, so this example relies on the action page. --&gt; &lt;form action="/html/tags/html_form_tag_action.cfm"&gt;   &lt;input type="input" id="first_name" name="first_name" placeholder="First Name..."&gt;   &lt;input type="submit" id="submit" value="Submit"&gt; &lt;/form&gt;  &lt;p id="msg"&gt;&lt;/p&gt;</pre>	<input type="text" value="Amanda"/> <input type="submit" value="Submit"/>
---	---

## Unobtrusive JavaScript features

These are the following features of Unobtrusive JavaScript

- Usability
- Graceful Degradation or progressive enhancement to support user agents that may not support advanced JavaScript functionality
- Accessibility
- Separation of functionality from content and presentation

## Progressive Enhancement

Progressive Enhancement is a powerful methodology that allows Web developers to concentrate on building the best possible websites while balancing the issues inherent in those websites being accessed by multiple unknown user-agents.

Progressive Enhancement (PE) is the principle of starting with a rock-solid foundation and then adding enhancements to it if you know certain visiting user-agents can handle the improved experience.

### The benefits of PE

Progressive Enhancement allows everyone to access the basic content and functionality of a web page, using any browser or Internet connection, while also providing an enhanced version of the page to those with more advanced browser software or greater bandwidth.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create a Progressively Enhanced list of website navigation items whose order is saved by the user.**

```
<form action="save_order.php" method="post">
```

```
<fieldset>
```

<legend>Order of Navigation</legend>

<ol>

<li id="homepage-12">Homepage <label for="menu-id-12">Change the order for Homepage</label><input type="text" name="homepage-12" id="menu-id-12" value="1" /></li>

<li id="contact-23">Contact Us <label for="menu-id-23">Change the order for Contact Us</label><input type="text" name="contact-23" id="menu-id-23" value="2" /></li>

<li id="about-16">About Us <label for="menu-id-16">Change the order for About Us</label><input type="text" name="about-16" id="menu-id-16" value="3" /></li>

<li id="latest-14">Latest News <label for="menu-id-14">Change the order for Latest News</label><input type="text" name="latest-14" id="menu-id-14" value="4" /></li>

</ol>

</fieldset>

<p><input type="submit" value="Save new order" /></p>

</form>

**2. Create another Progressively Enhanced list of website navigation items whose order is saved by the user.**

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- Unobtrusive JavaScript is an approach to JavaScript programming intended to facilitate progressive enhancement, separation of functionality from content/presentation, and overall best practice.
- Progressive Enhancement (PE) is the principle of starting with a rock-solid foundation and then adding enhancements to it if you know certain visiting user-agents can handle the improved experience.

**Testing your skills**

1. \_\_\_\_\_ is an approach to JavaScript programming intended to facilitate progressive enhancement
  - a) Unobtrusive JavaScript, b) Conditional JavaScript, c) CSS JS, d) None of the above
2. The full form of PE is
  - a) Positional Element, b) Progressive Enhancement, c) Programming Extension, d) None
3. These are the following features of Unobtrusive JavaScript
  - a) Usability, b) Accessibility, c) Progressive enhancement, d) all of the above
4. Unobtrusive JavaScript is also known as
  - a) Unobtrusive DOM scripting, b) Class scripting, c) Universal Scripting, d) Conditional scripting

**Lesson 14: Test and Debug a JavaScript Application (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Introduction to testing and debugging</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

When developing codes in JavaScript, you can witness occurrence of errors. It is a common scenario. Errors can be syntactical or logical. However, the occurrence of such errors creates a lot of ambiguity in the logic and understanding of both users and programmers. There can also be errors in the code which can remain invisible to the programmer's eye and can create havoc. To identify these errors we need suitable debugging methods. The debuggers test the program, identify the errors, and then fix the same.

**The Debugger Keyword**

The debugger keyword is used in the code to force stop the execution of the code at a breaking point and calls the debugging function. The debugger function is executed if any debugging is needed at all else no action is performed.

Let's have a look at a sample example:

*<h4>Debugging demonstrations using Debugger keyword</h4>*

*The solution of  $20 * 5$  is:*

*<p id="test"></p>*

*<p>If the debugger is turned on the code stops*

*execution at the start of the debugger</p>*

*var x = 20;*

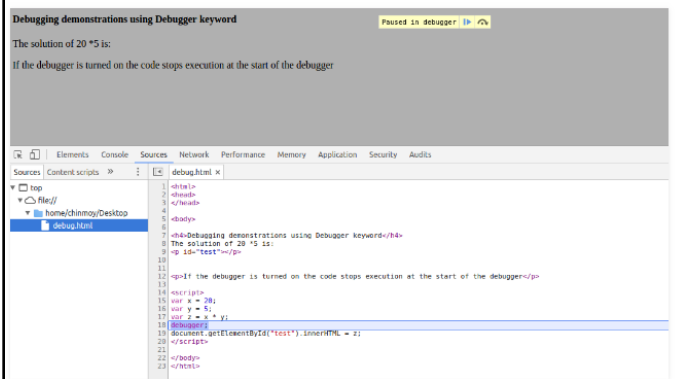
*var y = 5;*

*var z = x \* y;*

*debugger;*

*document.getElementById("test").innerHTML = z;*

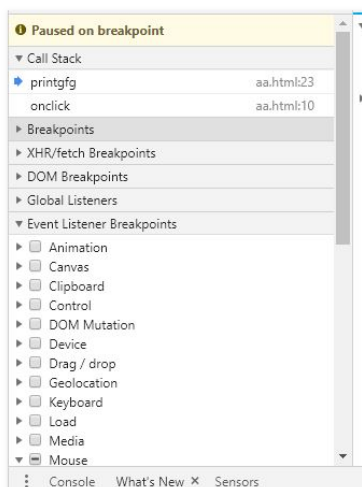
**Output:**



Use of `console.log()` method is also carried out in certain cases. However, setting breakpoint is a faster, efficient and better method. In this method, Breakpoints are set in code which stops the execution of code at that point so that the values of variables can be examined at that time.

Let's focus on the benefits associated with the use of Breakpoints over `console.log()` method:

- In `console.log()` method user has to understand the code and manually put `console.log()` at points in the code. But in breakpoints method, we can set breakpoints through Developers tool anywhere in code without even knowing it.
- In `console.log()` method we have to manually print values of different variables but at a certain breakpoint, all the values of all variables are displayed automatically in Developers tool.
- Enter Developers tool section by pressing F12 key and go to sources.
- In the source section, select a javascript file and set breakpoints by either selecting from the provided list like DOM breakpoints or Event listener breakpoints which stops the execution of code whenever an event occurs



Let's focus on another sample example that demonstrates the use of debugger:

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;/head&gt;  &lt;body&gt;  &lt;p id="demo"&gt;&lt;/p&gt;  &lt;p&gt;With the debugger turned on, the code below should stop executing before it executes the third line.&lt;/p&gt;  &lt;script&gt; var x = 15 * 5; debugger; document.getElementById("demo").innerHTML = x; &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<p>75</p> <p>With the debugger turned on, the code below should stop executing before it executes the third line.</p>
--	---

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a JavaScript program that converts a Fahrenheit into Celsius

```
<!DOCTYPE html>

<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>
```

**2. Create a JavaScript program that converts a Celsius into Fahrenheit**

**3. Create a program that uses the toString() number method to convert a number to a string**

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

In this chapter, we learned about the basics of debugging in JavaScript through some sample applications.

### Testing your skills

1. \_\_\_\_\_ keyword is used in the code to force stop the execution of the code at a breaking point

a) Debugger, b) Debugging, c) Debug, d) None of the above

2. Which key can be pressed to visit the Developers Tool section?

a) F5, b) F8, c) F12, d) F10

3. Errors can be

a) Syntax based, b) Logical, c) both (a) and (b), d) None of the above

4. \_\_\_\_\_ can stops the execution of code whenever an event occurs

a) Debugging breakpoint, b) Event Listener breakpoint, c) Direct breakpoint, d) None of the above

**Lesson 15: ES6 Features (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"><li>• Let, Var and Const</li><li>• Keywords,</li><li>• Default function arguments</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**Var**

Var is the JavaScript variables statement and used for declaring a variable and, optionally, we can initialize the value of that variable.

*Example: var a =10;*

**Mark the following facts about JavaScript Variables:**

- Variable declarations are processed before the execution of the code.
- The scope of a JavaScript variable declared with var is its current execution context.
- The scope of a JavaScript variable declared outside the function is global.

Take a sample coding example:

```
function nodeSimplified(){  
  
    var a =10;  
  
    console.log(a); // output 10  
  
    if(true){  
  
        var a=20;  
  
        console.log(a); // output 20  
    }  
  
    console.log(a); // output 20  
}
```



Study the above code. What can you find? Well, you can witness that when the variable is updated inside the if loop, that the value of variable "a" updated 20 globally, hence outside the if loop the value persists. It is similar to the Global variable present in other languages. But, be sure to use this functionality with great care because there is the possibility of overriding an existing value.

## Let

The let statement declares a local variable in a block scope. It is similar to Var, in that we can optionally initialize the variable.

*Example: let a =10;*

### Mark the following facts about Let statement:

The let statement allows you to create a variable with the scope limited to the block on which it is used.

It is similar to the variable we declare in other languages like Java, .NET, etc.

Take a sample coding example:

```
function nodeSimplified(){  
  let a =10;  
  
  console.log(a); // output 10  
  
  if(true){  
    let a=20;  
  
    console.log(a); // output 20  
  }  
  
  console.log(a); // output 10  
}
```

It is almost the same behavior we see in most language.

```
function nodeSimplified(){  
  let a =10;
```

```
let a =20; //throws syntax error

console.log(a);

}
```

Error Message: Uncaught SyntaxError: Identifier 'a' has already been declared.

However, when you use Var, it works fine.

```
function nodeSimplified(){

  var a =10;

  var a =20;

  console.log(a); //output 20

}
```

The scope will be well maintained with a let statement and when using an inner function the Let statement makes your code clean and clear.

By now, you must have understood the basic difference between Var and Let commands in JavaScript. Now let's focus on Const.

## Const

The Const statement values can be assigned once and they cannot be reassigned. The scope of Const statement works similar to Let statements.

Take a sample coding example:

```
const a =10;

function nodeSimplified(){

  const MY_VARIABLE =10;

  console.log(MY_VARIABLE); //output 10

}
```

As per usual, naming standards dictated that we declare the const variable in capital letters. `const a = 10` will work the same way as the code given above. Naming standards should be followed to maintain the code for the long run.

## Function Scope

Variables declared with `Var` and `Let` are quite similar when declared inside a function.

They will both have Function Scope.

Take a sample coding example:

```
function myFunction() {  
  
    var carName = "Volvo"; // Function Scope  
  
}
```

```
function myFunction() {  
  
    let carName = "Volvo"; // Function Scope  
  
}
```

## Function Arguments

Function arguments are the real values passed to (and received by) the function. A JavaScript function does not perform any checking on parameter values (arguments). JavaScript functions have a built-in object called the arguments object. The argument object contains an array of the arguments used when the function was called (invoked).

Let's focus on an example where you can simply use a function to find (for instance) the highest value in a list of numbers:

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;p&gt;Finding the largest number.&lt;/p&gt; &lt;p id="demo"&gt;&lt;/p&gt;  &lt;script&gt; function findMax() {     var i;     var max = -Infinity;     for(i = 0; i &lt; arguments.length; i++) {         if (arguments[i] &gt; max) {             max = arguments[i];         }     }     return max; } document.getElementById("demo").innerHTML = findMax(4, 5, 6); &lt;/script&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<p>Finding the largest number.</p> <p>6</p>
--	---

## Essential facts about arguments

- The parameters, in a function call, are the function's arguments.
- JavaScript arguments are passed by value: The function only gets to know the values, not the argument's locations.
- If a function changes an argument's value, it does not change the parameter's original value.
- Changes to arguments are not visible (reflected) outside the function.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Using the LET statement declare a variable with block scope

```
<!DOCTYPE html>

<html>

<body>

<h2>Declaring a Variable Using let</h2>

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

<script>

var x = 10;

// Here x is 10

{

let x = 2;

// Here x is 2

}

// Here x is 10

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>
```

## 2. Using the LET statement declare a variable with value 10 with block scope

Note: After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Var is the JavaScript variables statement and used for declaring a variable
- The let statement declares a local variable in a block scope
- The Const statement values can be assigned once and they cannot be reassigned
- Function arguments are the real values passed to (and received by) the function
- JavaScript functions have a built-in object called the arguments object

### Testing your skills

1. \_\_\_\_\_ is the JavaScript variables statement and used for declaring a variable

a) Var, b) Let, c) Const, d) All of the above

2. \_\_\_\_\_ declares a local variable in a block scope

a) Var, b) Let, c) Const, d) None of the above

3. \_\_\_\_\_ statement values can be assigned once and they cannot be reassigned

a) Var, b) Let, c) Const, d) None of the above

4. JavaScript functions have a built-in object called the

a) static object, b) arguments object, c) function object, d) content object

## Lesson 16: Bootstrap Components with JavaScript (120 minutes)

**Objective:** After completing this lesson you will be able to :

- Pagination, panel, jumbotron

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

**Total Duration:** 120 minutes

### Pagination

Pagination.js is a robust, highly customizable & stylish, jQuery based pagination system for your long content to improve webpage readability.

Features:

- Supports local data or remote data via Ajax request.
- Customizable pagination text and numbers.
- Fully styleable via CSS.
- Tons of options/methods/events to meet your actual needs.

A sample example to create pagination

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
.pagination a {
  color: black;
  float: left;
  padding: 8px 16px;
  text-decoration: none;
  transition: background-color .3s;
}

.pagination a.active {
  background-color: dodgerblue;
  color: white;
}

.pagination a:hover:not(.active) {background-color: #ddd;}
</style>
</head>
<body>

<h2>Pagination</h2>
<p>Responsive pagination with hover effects:</p>

<div class="pagination">
  <a href="#">&laquo;</a>
  <a href="#">1</a>
  <a class="active" href="#">2</a>
  <a href="#">3</a>
  <a href="#">4</a>
  <a href="#">5</a>
  <a href="#">6</a>
  <a href="#">&raquo;</a>
</div>
</body>
</html>
```

### Pagination

Responsive pagination with hover effects:

« 1 2 3 4 5 6 »

## Panel

A panel in bootstrap is a bordered box with some padding around its content. Panels are created with the `.panel` class, and content inside the panel has a `.panel-body` class. The `.panel-default` class is used to style the color of the panel.

Let's focus on a sample example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
</head>
<body>
  <div class="container">
    <h2>Basic Panel</h2>
    <div class="panel panel-default">
      <div class="panel-body">A Basic Panel</div>
    </div>
  </div>
</body></html>
```

### Basic Panel

A Basic Panel

## Jumbotron

A jumbotron indicates a big grey box for calling extra attention to some special content or information. Inside the jumbotron, you can put nearly any valid HTML, including other Bootstrap elements/classes.

In the example below, we will be using a `<div>` element with class `.jumbotron` to create a jumbotron:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</head><body>
  <div class="container">
    <div class="jumbotron">
      <h1>Bootstrap Tutorial</h1>
      <p>Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile-first projects on the web.</p>
    </div>
    <p>This is some text.</p>
    <p>This is another text.</p>
  </div>
</body></html>
```

### Bootstrap Tutorial

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile-first projects on the web.

This is some text.

This is another text.

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create a Basic Panel in JavaScript Bootstrap

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>

<title>Bootstrap Example</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

</head>

<body>

<div class="container">

<h2>Basic Panel</h2>

<div class="panel panel-default">

<div class="panel-body">A Basic Panel</div>

</div>

</div>
```

## 2. Create another Basic Panel with different source in JavaScript Bootstrap

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

In this chapter, we learned about different bootstrap concepts like pagination, panel, and jumbotron. We also focused on sample examples for each case.



**Testing your skills**

1. Which of these are features of pagination

a) Supports remote data via Ajax, b) Fully stylized via CSS, c) Tons of methods, d) All of the above

2. \_\_\_\_\_ is a bordered box with some padding around its content

a) Panel, b) Outline, c) Offset, d) none of the above

3. \_\_\_\_\_ indicates a big grey box for calling extra attention to some special content

a) Panel, b) Jumbotron, c) Offset, d) none of the above

4. The \_\_\_\_\_ class is used to style the color of the panel

a) .panel-style, b) .panel-css, c) .panel-default, d) .panel-color