Web Technology

Kumar Lamichhane
Email:kumar.lamichhane@deerwalk.edu.np
Contact No.: 9841389377

Syllabus

- UNIT 4: Client Side Scripting with JavaScript (9 Hrs.)
  - Structure of JavaScript Program
  - Variables and Data Types
  - Statements: Expression, Keyword, Block
  - Operators
  - Flow Controls, Looping, Functions
  - Popup Boxes: Alert, Confirm, Prompt
  - Objects and properties
  - Constructors
  - Arrays
  - Built-in Objects: Window, String, Number, Boolean, Data, Math, RegExp,
    Form, DOM
  - User Defined Objects
  - Event Handling and Form Validation, Error Handling, Handling Cookies, jQuery Syntax
  - jQuery Selectors, Events and Effects
  - Introduction to JSON

UNIT 4: CLIENT SIDE SCRIPTING WITH JAVASCRIPT

JavaScript

- HTML to define the content of web pages; CSS to specify the layout of web pages; JavaScript to program the behavior of web pages.
- JavaScript is a case sensitive scripting language.
- A scripting language is a lightweight programming language.
- JavaScript is usually embedded directly into HTML pages.
- JavaScript is an interpreted language (that scripts execute without preliminary compilation).
- Everyone can use JavaScript without purchasing a license.
- Java and JavaScript are two completely different languages in both concept & design.
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language – in the same category as C and C++.

What can JavaScript do?

- **JavaScript gives HTML designers a programming tool** – HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax. Almost anyone can put small "snippets" of code into their HTML pages.
- **JavaScript can put dynamic text into an HTML page** – a JavaScript statement like this: *document.write("<h1>"+name+"</h1>")* can write a variable text into an HTML page.
- **JavaScript can react to events** – a JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.
- **JavaScript can read and write HTML elements** – a JavaScript can read and change the content of an HTML element.
- **JavaScript can be used to validate data** – a JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing.
- **JavaScript can be used to detect the visitor's browser** – a JavaScript can be used to detect the visitor's browser, and – depending on the browser – load another page specifically designed for that browser.
- **JavaScript can be used to create cookies** – a JavaScript can be used to store and retrieve information on the visitor's computer.

JavaScript – Syntax

- Use JavaScript to write text on a web page:

```
<body>
<script type="text/javascript"> document.write("Hello World!");
</script>
</body>
```

- Add HTML tags to the JavaScript

```
<body>
<script type="text/javascript">
document.write("<h1>HelloWorld!</h1>");
</script>
</body>
```

JavaScript – Comments

- Single line comments

```
<body>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
// Change heading:
document.getElementById("myH").innerHTML = "JavaScript Comments";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
```

- Multiple lines comments

```
<body>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
/*
The code below will change the heading with id = "myH"
and the paragraph with id = "myP"
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
```

JavaScript – Comments

- Single line comments to prevent execution

```
<body>
<h2>JavaScript Comments</h2>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
<p>The line starting with // is not executed.</p>
</body>
```

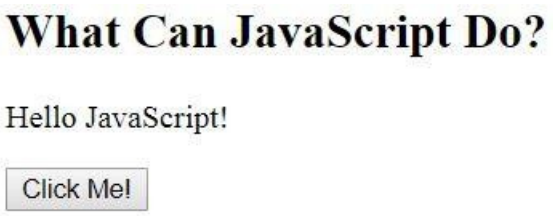
- Multiple lines comments to prevent execution

```
<body>
<h2>JavaScript Comments</h2>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
/*
document.getElementById("myH").innerHTML = "Welcome to my Homepage";
document.getElementById("myP").innerHTML = "This is my first paragraph.";
*/
document.getElementById("myP").innerHTML = "The comment-block is not executed.";
</script>
</body>
```

What can JavaScript do? – Example 1

```
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick='document.getElementById("demo").innerHTML
= "Hello JavaScript!"'>Click Me!</button>
</body>
</html>
```

Output:



What can JavaScript do? – Example 2

```
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p>JavaScript can change HTML attribute values.</p>
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>
<button onclick="document.getElementById('myImage').src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/93927/pic_bulbon.gif'">Turn on the light</button>
<img id="myImage" src="https://s3-us-west-2.amazonaws.com/s.cdpn.io/93927/pic_bulboff.gif" style="width:100px">
<button onclick="document.getElementById('myImage').src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/93927/pic_bulboff.gif'">Turn off the light</button>
</body>
</html>
```

What can JavaScript do? – Example 2

Output:

What Can JavaScript Do?

JavaScript can change HTML attribute values.

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

Turn on the light Turn off the light

What Can JavaScript Do?

JavaScript can change HTML attribute values.

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

Turn on the light Turn off the light

Where to Insert JavaScript?

- ***JavaScript in Head***

```
<html>
<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h2>JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

Where to Insert JavaScript?

Output:

Where to Insert JavaScript?

- **JavaScript in Body**

```
<html>
<body>
<h2>JavaScript in Body</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
 document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</body>
</html>
```

Where to Insert JavaScript?

Output:

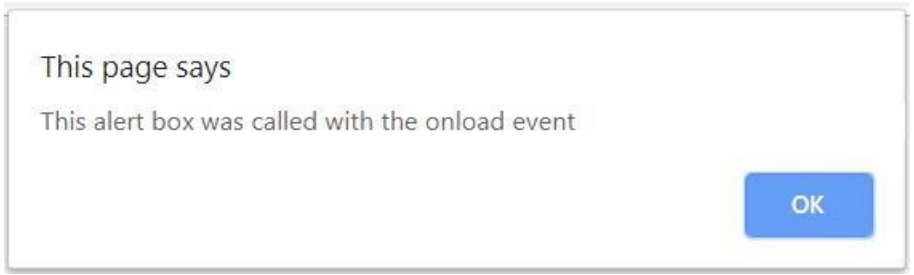
Where to Insert JavaScript?

- **JavaScript in both Head & Body**

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>
<body onload="message()">
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>
</html>
```

Where to Insert JavaScript?

Output:



This message is written by JavaScript

Using an External JavaScript

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file and save the file with a *.js* file extension.
- The external script cannot contain the *<script></script>* tags.

```
function myFunction()
{document.getElementById("demo").innerHTML = "Paragraph changed.";}
```

- Using an External JavaScript

```
<body>
<h2>External JavaScript</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
<p>(myFunction is stored in an external file called "myScript.js")</p>
<script src="myScript.js"></script>
</body>
```

JavaScript Statements

- JavaScript statements are commands to the browser.

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello Dolly.";
</script>
```

- JavaScript code is a sequence of statements.

```
<p id="demo"></p>
<script>
var x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;  // Statement 4
document.getElementById("demo").innerHTML ="The value of z is " + z + ".";
</script>
```

- JavaScript statements are separated with semicolon.

```
<p id="demo1"></p>
<script>
var a, b, c;  a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
```

JavaScript Statements

- Multiple statement on one line is allowed.

```
<p id="demo1"></p>
<script>
var a, b, c; a = 5; b = 6; c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
```

- JavaScript statements can be grouped together in code blocks.

```
<button type="button" onclick="myFunction()">Click Me!</button>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
function myFunction()
{
document.getElementById("demo1").innerHTML=
"Hello Students!";
document.getElementById("demo2").innerHTML = "How are you?";
}
</script>
```

JavaScript Keywords

- In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

JavaScript Data Types

- JavaScript data types are dynamic. i.e. the same variable can be used to hold different data types.

```
var x;                  // Now x is undefined
x = 5;                  // Now x is a Number
x = "John";             // Now x is a String
```

- JavaScript evaluates expressions from left to right.
- Different sequences can produce different results.

```
var x = 16+"Volvo"; //results:16Volvo
var x = "Volvo" + 16;     //results:Volvo16
var x = 16 + 4 + "Volvo"; //results: 20Volvo
```

- In the last example, JavaScript treats 16 and 4 as numbers, until it reaches “Volvo”. As a result, it adds 16 & 4 and then concatenates Volvo with the result.

JavaScript Variables

- JavaScript variables are containers for storing data values.
- A variable can have a short name, like x, or a more descriptive name, like carname.
- Rules for JavaScript variable names:
  - Variable names are case sensitive (y & Y are two different variables).
  - Variable names must begin with a letter or the underscore character.
- You can declare JavaScript variables with the *var* statement.

  var x; var x=5;

  var carname; var carname="volvo";
- After the execution of the statements above, the variable x will hold the value 5, and carname will hold the value volvo.

JavaScript Operators

- The *assignment* operator (=) assigns a value to a variable.

  var x = 10;

- The *addition* operator (+) adds numbers.

  var x=5; var y=2; var z = x+y;

- The *multiplication* operator (*) multiplies numbers.

  *var x=5; var y=2; var z = x*y;*

JavaScript Arithmetic Operators

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

JavaScript Arithmetic Operators

- The addition (+) operator

```
<body>
<p id="demo"></p>
<script>
var x = 5; var y = 2; var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
```

- The increment (++) operator

```
<body>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
var y = 6;
document.getElementById("demo1").innerHTML = y;
var x = ++y;
document.getElementById("demo2").innerHTML = x;
</script>
</body>
```

JavaScript Assignment Operators

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

JavaScript Assignment Operators

- The += and **= operators

```
<body>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
var x = 10; x += 5;
document.getElementById("demo1").innerHTML = x;
var y = 5; y **= 3;
document.getElementById("demo2").innerHTML = y;
</script>
</body>
```

JavaScript String Operators

- The + operator can also be used to add (concatenate) strings.
  var txt1="John"; var txt2="Doe"; var txt3=txt1+" "+txt2;
- The result of txt3 will be:
  John Doe
- The+= assignment operator can also be used to add (concatenate) strings.
  var txt1="What a "; txt1+="nice day";
- The result of txt1 will be:
  What a nice day
- Adding two numbers, will return the sum, but adding a number and a string will return a string.
  var x=5+5; var y="5"+5; var z = "Hello"+5;
- The result of x, y, & z will be:
  10 55 Hello5

JavaScript Comparison Operators

Operator	Description
==	Equal to
===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator

JavaScript Comparison Operators

- The === and !== operators

```
<body>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
 var x = 5;
document.getElementById("demo1").innerHTML = (x === 6);
var y = 5;
document.getElementById("demo2").innerHTML = (y !== 6);
</script>
</body>
```

JavaScript Logical Operators

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

Popup Boxes – Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.

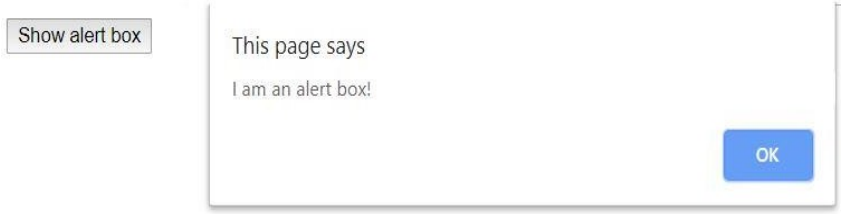
  alert("sometext");

- For example:

```
<head>
<script type="text/javascript">
function show_alert()
{alert("I am an alert box!"); }
</script>
</head>
<body>
<input type="button" onclick="show_alert()"
value="Show Alert box" />

</body>
```

Output:



Popup Boxes – Confirmation Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either “OK” or “Cancel” to proceed.
- If the user clicks “OK”, the box returns true; and if the user clicks “Cancel”, the box returns false.

confirm(“sometext”);

- For example:

```
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
  if (r==true) {document.write("You pressed OK!"); }
  else {document.write("You pressed Cancel!");}
}
</script>
</head>

  <body>

  <input type="button" onclick="show_confirm()" value="Show confirm box" />

  </body>
```

Popup Boxes – Confirmation Box

Output:

You pressed OK!

You pressed Cancel!

Popup Boxes – Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK", the box returns the input value; and if the user clicks "Cancel", the box returns null.

prompt("sometext", "defaultvalue");

- For example:

```
<body>

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

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

<script>
function myFunction() {
    var txt;
    var person = prompt("Please enter your name:");
    if (person == null || person == "") {txt = "User cancelled the prompt.";}
      else {txt = "Hello " + person + "! How are you today?";}
      document.getElementById("demo").innerHTML = txt;

}

</script>

</body>
```

Popup Boxes – Prompt Box

Output:

Try it

Try it

Hello Harry Potter! How are you today?

Try it

User cancelled the prompt.

Functions

- A function is simply a block of code with a name, which allows the block of code to be called by other components in the scripts to perform certain tasks.
- Functions can also accept parameters that they use to complete their task.
- JavaScript actually comes with a number of built-in functions to accomplish a variety of tasks.
- Creating Custom Functions:

```
<script type = "text/javascript">
function name_of_function(argument1,argument2,…,arguments)
{
                    ……..……..
                    //Block of code
                    ……..……..
}
</script>
```

Calling Functions:

There are two common ways to call a function; from an event handler and from another function.

Calling a function is simple. You have to specify its name followed by the pair of parenthesis

```
name_of_function(argument1,argument2,…,arguments)
```

A Simple Function

```
<html>
<head>
<script>
function myFunction()
{
 document.getElementById("demo").innerHTML = "Hello
 World!";
}
</script>
</head>
<body>
<p>When you click "Try it", a function will be called.</p>
<p>The function will display a message.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
</body>
</html>
```

A Simple Function

Output:

When you click "Try it", a function will be called.

The function will display a message.

Try it

When you click "Try it", a function will be called.

The function will display a message.

Try it

Hello World!

Function that Returns a Value

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

Output:

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

12

JavaScript Objects

- In JavaScript, almost everything is an object
- Booleans, Numbers, and Strings (if defined with the new keyword) can be objects.
- Dates, Maths, Arrays, Functions, Regular expressions, and Objects are always objects.
- All JavaScript values, except primitives, are objects.
- A primitive value is a value that has no properties or methods.
- A primitive data type is data that has a primitive value.
- JavaScriptdefines 5 types of primitive data types: string, number, boolean, null, & undefined.
- Primitive values are immutable (they are hardcoded and therefore cannot be changed).

JavaScript Objects Properties

- JavaScript variables can contain single values.

  var person = "John Doe";
- Objects are variables too; but objects can contain many values.
- The values are written as name:value pairs (name and value separated by a colon). – creating an object using object literal

  var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
- A JavaScript object is a collection of named values.
- The named values, in JavaScript objects, are called properties.
- In above example, "firstName", "lastName", "age", and "eyeColor" are properties; whereas "John", "Doe", "50", and "blue" are their respective values.

Math Object in JavaScript

- The Math object allows you to perform mathematical tasks.
- The Math object includes several mathematical constants and methods.

```
var pi_value = Math.PI;
var sqrt_value = Math.sqrt(16);
```

- All properties and methods of Math can be called by using Math as an object without creating it.

```
<body>
<h2>JavaScript Math.PI</h2>
<p>Math.PI returns the ratio of a circle's circumference to its
diameter:</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Math.PI;
</script>
</body>
```

Math Object in JavaScript

Properties

- Math.E – returns Euler's number (approx. 2.718)
- Math.LN2 – returns the natural logarithm of 2 (approx. 0.693)
- Math.LN10 – returns the natural logarithm of 10 (approx. 2.302)
- Math.LOG2E – returns the base-2 logarithm of E (approx. 1.442)
- Math.LOG10E – returns the base- 10 logarithm of E (approx. 0.434)
- Math.PI – returns PI (approx. 3.14159)

Methods

- abs(x) – returns the absolute value of x
- ceil(x) – returns x, rounded upwards to the nearest integer
- floor(x) – returns x, rounded downwards to the nearest integer
- log(x) – returns the natural logarithm (base E) of x
- max(x,y,z,…,n) – returns the number with the highest value
- min(x,y,z,…,n) – returns the number with the lowest value
- pow(x,y) – returns the value of x to the power of y
- sqrt(x) – returns the square root of x
- random(x) – returns a random number between 0 & 1
- round(x) – rounds x to the nearest integer

Window Object in JavaScript

- The window object represents the browser's window.
- Two properties *window.innerHeight* and *window.innerWidth* can be used to determine the size (in pixels) of browser window.

```
<body>
<p id="demo"></p>
<script>
var w = window.innerWidth;
var h = window.innerHeight;
var x = document.getElementById("demo");
 x.innerHTML = "Browser inner window width: " + w + ", height: " + h+ ".";
</script>
</body>
```

Date Object in JavaScript

- The Date object is used to work with dates and times.
- Date objects are created with the Date() constructor.
- Date object methods in JavaScript:
  - *getDate()* – returns the day of the month (from 1 – 31)
  - *getDay()* – returns the day of the week (from 0 – 6)
  - *getFullYear()* – returns the year (four digits)
  - *getHours()* – returns the hour (from 0 – 23)
  - *getMilliseconds()* – returns the milliseconds (from 0 – 999)
  - *getMinutes()* – returns the minutes (from 0 – 59)
  - *getMonth()* – returns the month (from 0 – 11)
  - *getSeconds()* – returns the seconds (from 0 – 59)

Date Object in JavaScript

```
<html>
<body>
<h2>JavaScript new Date()</h2>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
//document.getElementById("demo").innerHTML = d.getDate();
//document.getElementById("demo").innerHTML = d.getDay();
//document.getElementById("demo").innerHTML = d.getFullYear();
//document.getElementById("demo").innerHTML = d.getHours();
//document.getElementById("demo").innerHTML = d.getMilliseconds();
//document.getElementById("demo").innerHTML = d.getMinutes();
//document.getElementById("demo").innerHTML = d.getMonths();
//document.getElementById("demo").innerHTML = d.getSeconds();
</script>
</body>
</html>
```

String Methods in JavaScript

- String methods helps us to work with strings.
- The *length* property returns the length of a string.
- The *indexOf()* method returns the index of the *first* occurrence of a specified text in a string.
- The *search()* method searches a string for a specified value and returns the position of the match.
- The *slice()* method extracts a part of a string and returns the extracted part in a new string.
- The *replace()* method replaces a specified value with another value in a string.
- The *toUpperCase()* method converts a string to upper case.
- The *toLowerCase()* method converts a string to lower case.

String Methods in JavaScript

```
<html>
<body>
<h2>JavaScript String Properties</h2>
<p id="demo"></p>
<script>
var txt = "The quick brown fox jumps over the lazy dog.";
document.getElementById("demo").innerHTML = txt.length;
/*
var pos = txt.indexOf("fox");
document.getElementById("demo").innerHTML = pos;

var c = txt.search("jumps");
document.getElementById("demo").innerHTML = c;

var res = txt.slice(4,9);
document.getElementById("demo").innerHTML = res;

var txt = txt.replace("quick","slow");
document.getElementById("demo").innerHTML = txt;

var txt = txt.toUpperCase();
*/
</script>
</body>

</html>
```

Number Methods in JavaScript

- The *Number()*, can be used to convertJavaScript variables to numbers.
- The *parseInt()* parses a string and returns a whole number. Spaces are allowed but only the first number is returned.
- The *parseFloat()* parses a string and returns a number. Spaces are allowed but only the first number is returned.
- The *toExponential()* method returns a string representing the number object in exponential notation.
- The *toString()* method returns a string representing the specified object. The *toString()* method parses its first argument, and attempts to return a string representation in the specified radix.
- The *toFixed()* returns a string, with the number written with a specified number of decimals.
- The *toPrecision()* returns a string, with a number written with a specified length.

Number Methods in JavaScript

```
<script type="text/ javascript">
x = true;
document.write(Number(x) +"<br>");
x = false;
document.write(Number(x) +"<br>");
x = "10"
document.write(Number(x) +"<br>");
x = "10 20"
document.write(Number(x) +"<br>");
document.write(parseInt("12") + "<br>");
document.write(parseInt("12.33") + "<br>");
document.write(parseInt("12 6") + "<br>" );
document.write(parseInt("12 years") + "<br>");
document.write(parseInt("month 12"));
</script>
```

Number Methods in JavaScript

```
<body>
<p id="demo"></p>
<script>
var x = 9.656;
document.getElementById("demo").innerHTML =
  x.toExponential() + "<br>" +  x.toExponential(2) +
  "<br>" +  x.toExponential(4) + "<br>" +
  x.toExponential(6);
</script>
</body>
```

Number Methods in JavaScript

```
<script type="text/javascript">
var num = new Number(15);
document.write("num.toString() is: " + num.toString() +"<br />");
document.write("num.toString(2) is: " + num.toString(2)
+"<br />");
document.write("num.toString(4) is: " + num.toString(4) +"<br
/>");
</script>
```

Number Methods in JavaScript

```
<body>
<p id="demo"></p>
<script>
var x = 9.656;
document.getElementById("demo").innerHTML =  x.toFixed(0) + "<br>" +
 x.toPrecision() + "<br>" +  x.toFixed(2) + "<br>" +  x.toPrecision(2)
 + "<br>" +
 x.toFixed(4) + "<br>" +  x.toPrecision(4) + "<br>" +  x.toFixed(6) +
 "<br>" +
 x.toPrecision(6);
</script>
</body>
```

Boolean Methods in JavaScript

- A JavaScript Boolean represents one of two values: **true** or **false**.
- *Boolean(5>3)* and 5>3 returns same output as true.

```
<body>
<p>Display the value of Boolean(10 > 9):</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
  document.getElementById("demo").innerHTML = Boolean(5 > 3);
}
</script>
</body>
```

Boolean Methods in JavaScript

- Everything with a "*value*" is **TRUE**.

```
<body>
<p id="demo"></p>
<script>
var b1 = Boolean(100);
var b2 = Boolean(3.14);
var b3 = Boolean(-15);
var b4 = Boolean("Hello");
var b5 = Boolean('false');
var b6 = Boolean(1 + 7 + 3.14);
document.getElementById("demo").innerHTML = "100 is "+ b1 + "<br>" +
"3.14 is " + b2 + "<br>" +  "-15 is " + b3 + "<br>" +
"Any (not empty) string is " + b4 + "<br>" +  "Even the string 'false' is " +
b5 + "<br>" +  "Any expression (except zero) is " + b6;
</script>
</body>
```

Boolean Methods in JavaScript

- Everything without a "*value*" is **FALSE**.
- The Boolean value of *0 (zero)* is **FALSE**.
- The Boolean value of *-0 (minus zero)* is **FALSE**.
- The Boolean value of " " *(empty string)* is **FALSE**.
- The Boolean value of *undefined* is **FALSE**.
- The Boolean value of *null* is **FALSE**.
- The Boolean value of *false* is **FALSE**.
- The Boolean value of *NaN* is **FALSE**.

Regular Expression in JavaScript

- Regular expressions are patterns used to match character combinations in strings.
- In JavaScript, regular expressions are also objects.
- Regular expressions can be used to perform all types of *text search* and *text replace*.
- There are two main string methods: *search()* and *replace()*.
- ***search()*** method uses an expression to search for a match, and returns the position of the match.
- ***replace()*** method uses an expression to return a modified string where the pattern is replaced.

Regular Expression in JavaScript

```
<body>
<p id="demo"></p>
<script>
var str = "The quick brown fox jumps over the lazy dog";
var n = str.search(/brown/);
document.getElementById("demo").innerHTML = n;
</script>
</body>
```

Regular Expression in JavaScript

```
<body>
<button onclick="myFunction()">Click to replace</button>
<p id="demo">The quick brown fox jumps over the lazy dog</
p>
<script>
function myFunction()
{
  var str = document.getElementById("demo").innerHTML;
  var txt = str.replace(/quick/,"slow");
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
```

Form Method Property

- It is used to change the method for sending form data.
- The method property sets or returnsthe value of method attribute in a form.
- The syntax for setting the method property is:

  formObject.method = get | post
- **get**– appends the form-data to the URL: URL? name=value&name=value (this is default).
- **post** – sends the form-data as an HTTP post transaction.

Form Method Property

```
<body>
<h2>Contact Us</h2>
<p>Please fill in this form and send us.</p>
<form action="form2-get.php" method="get">
<p>
<label for="inputName">Name:<sup>*</sup></label>
<input type="text" name="name" id="inputName">
</p>
<p>
<label for="inputEmail">Email:<sup>*</sup></label>
<input type="text" name="email" id="inputEmail">
</p>
<p>
<label for="inputSubject">Subject:</label>
<input type="text" name="subject" id="inputSubject">
</p>
<p>
<label for="inputComment">Message:<sup>*</sup></label>
<textarea name="message" id="inputComment" rows="5" cols="30"></textarea>
</p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
</body>
```

Form Method Property

```
<body>
<h1>ThankYou</h1>
<p>Here is the information you have submitted:</p>
<ol>
<li><em>Name:</em> <?php echo $_GET["name"]?></li>
<li><em>Email:</em> <?php echo $_GET["email"]?></li>
<li><em>Subject:</em> <?php echo
$_GET["subject"]?></li>
<li><em>Message:</em><?php echo
$_GET["message"]?></li>
</ol>
</body>
```

JavaScript Arrays

- An array is a special variable that can hold more than one value at a time.
- A list of items, storing the fruits in a single variable could be like:
- *var fruit1 = "Apple"; var fruit2 = "Banana";*
- This might look possible for few items but isn't feasible for hundreds of item.
- The solution is an array which can replace multiple declaration in a single line like:
- *var fruits = ["Apple","Banana"];*
- To access an array element, we can refer to the index number like:
- *document.getElementById("demo").innerHTML = fruits[0];*
- The *length* property of an array returns the length of an array.
- The *push()* method is used to add a new element to an array.

JavaScript Arrays

```
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Apple", "Banana","Cherry","Orange","Mango"];
document.getElementById("demo").innerHTML = "The first element of an array is " + fruits[0];
//document.getElementById("demo").innerHTML = "The last element of an array is " +
fruits[fruits.length-1];
//document.getElementById("demo").innerHTML = "The elements of an array are " + fruits;
//document.getElementById("demo").innerHTML = "The length of an array is " + fruits.length;
/* fruits.push("Lemon");
document.getElementById("demo").innerHTML = "The elements of an array after adding Lemon are "
+ fruits;
*/
</script>
</body>
</html>
```

JavaScript Object Constructors

- Sometimes, we need a “blueprint” for creating many objects of same “type”.
- In such situation, the better option to create an “object type” is to use an object constructor function.
- All the objectsof same type are created by calling the constructor function with the ***new*** keyword.

JavaScript Object Constructors

```
<html>
<body>
<h2>JavaScript Object Constructors</h2>
<p id="demo"></p>
<script>
// Constructor function for Person objects
function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyeColor
= eye;
}
// Create two Person objects
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
// Display age
document.getElementById("demo").innerHTML = "My father is " + myFather.age + ". My mother is " +
myMother.age + ".";
/*
document.getElementById("demo").innerHTML = "My father name is " + myFather.firstName + " " +
myFather.lastName + ". He is " + myFather.age + " years old with " + myFather.eyeColor + " eyes. My mother
name is " + myMother.firstName + " " + myMother.lastName + ". She is " + myMother.age + " years old with "
+ myMother.eyeColor + " eyes.";
*/
</script>
</body>
</html>
```

JavaScript Conditionals

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
  - Use if to specify a block of code to be executed, if a specified condition is true.
  - Use else to specify a block of code to be executed, if the same condition is false.
  - Use else if to specify a new condition to test, if the first condition is false.
  - Use switch to specify many alternative blocks of code to be executed.

JavaScript Conditionals (if)

- Syntax:

```
if (condition) {
//b lock of code to be executed if the condition is true
}
```

- Example:

```
<body>
<p>Display "Good day!" if the hour is less than 18:00:
</p>
<p id="demo">Good Evening!</p>
<script>
if (new Date().getHours() < 18)
  {document.getElementById("demo").innerHTML = = "Good day!";}
  </script>
</body>
```

JavaScript Conditionals (else)

- *Syntax:*

```
if (condition) {
//b lock of code to be executed if the condition is true
} else {
//block of code to be executed if the condition is false
}
```

- *Example:*

```
<body>
<p>Click the button to display a time-based greeting:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
 var hour = new Date().getHours();
 var greeting;
 if (hour < 18) {
  greeting = "Good day";
 } else {
  greeting = "Good evening";
 }
 document.getElementById("demo").innerHTML = greeting;
}
</script>
</body>
```

JavaScript Conditionals (else if)

- *Syntax:*

```
if (condition1) {
//block of code to be executed if the condition1 is true
} else if (condition2) {
//block of code to be executed if the condition1 is false and condtion2 is true
} else {
//block of code to be executed if the condition1 is false and condition2 is false
}
```

- *Example:*

```
<body>
<p>Click the button to get a time-based greeting:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
var greeting;
var time = new Date().getHours();
if (time < 12) {
greeting = "Good morning";
} else if (time < 18) {  greeting = "Good day";
} else {
greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
}
</script>
```

- `</body>`

JavaScript Conditionals (switch)

Syntax:

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

Example

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

JavaScript Loops

- Loops can execute a block of code a number of times.
- JavaScript supports different kinds of loops:
  - for – loops through a block of code a number of times
  - for/in – loops through the properties of an object
  - for/of – loops through the values of an iterable object
  - while – loops through a block of code while a specified condition is true
  - do/while – also loops through a block of code while a specified condition is true

JavaScript Loop (for)

- Syntax:

```
for (statement1; statement2; statement3) {
//b lock of code to be executed
}
```

- Statement 1 is executed (one time) before the execution of the code block.
- Statement 2 defines the condition for executing the code block.
- Statement 3 is executed (every time) after the code block has been executed.
- Example:

```
<body>
<p id="demo"></p>
<script>
var num = ""; var i;
for (i = 0; i < 5; i++)
{num += "The number is " + i + "<br>"; }
document.getElementById("demo").innerHTML = num;
</script>
```

- `</body>`

JavaScript Loop (for/in)

- *Example:*
- *<body>*
- * *
- *<script>*
- *var txt = "";*
- *var person = {fname:"John", lname:"Doe", age:25}; var x;*
- *for (x in person)*
- *{*
- *txt + = person[x] + " ";*
- *}*
- *document.getElementById("demo").innerHTML = txt;*
- *</script>*
  - *</body>*

JavaScript Loop (for/of)

- Example:

```
<body>
<script>
var cars = ['BMW', 'Volvo', 'Mini'];
var x;
for (x of cars)
{
document.write(x + "<br >");
}
</script>
</body>
```

JavaScript Loop (while)

- Syntax:

```
while (condition) {
  //b lock of code to be executed
}
```

Example:

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

</script>

</body>
```

JavaScript Loop (do/while)

- Syntax:

```
do {
//b lock of code to be executed
} while (condition)
```

- Example:

```
<body>
<p id="demo"></p>
<script>
var text = "" var i = 0;
do {
  text += "<br>The number is " + i;  i++;
}
while (i < 10);
document.getElementById("demo").innerHTML = text;
</script>
</body>
```

JavaScript Error Handling

- When executing JavaScript code, different errors can occur.
- Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.
- There are four statements to handle errors:
  - The try statement lets you test a block of code for errors.
  - The catch statement lets you handle the error.
  - The throw statement lets you create custom errors.
  - The finally statement lets you execute code, after try and catch,
    regardless of the result.

JavaScript Error Handling – try, catch, & throw

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The throw statement allows you to create a custom error.
- If you use throw together with try and catch, you can control program flow and generate custom error messages.
- The JavaScript statements try and catch come in pairs:

```
try{
// block of code to try
}
catch(err) {
// block of code to handle errors
}
```

- Example:

```
<body>
<p id="demo"></p>
<script>
try { adddlert("Welcome guest!"); }
catch(err) { document.getElementById("demo").innerHTML = err.message; }
</script>
</body>
```

JavaScript Error Handling – try, catch, & throw (Example)

```
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction()
{  var message, x;
message = document.getElementById("p01");
message.innerHTML = "";
x = document.getElementById("demo").value;
 try {
  if(x == "")  throw "is empty";  if(isNaN(x)) throw "is not a number";  x = Number(x);
  if(x > 10)   throw "is too high";  if(x < 5)  throw "is too low";
}
catch(err) {   message.innerHTML = "Input " + err;  }
finally {  document.getElementById("demo").value = "";  }
}

      </script>

      </body>
```

JavaScript Error Handling – finally

- The finally statement letsyou execute code, after try and catch, regardless of the result:
- Syntax:

```
try{
//b lock of code to try
}
catch(err) {
//b lock of code to handle errors
}
finally{
//block of code to be executed regardless of the try/catch result
}
```

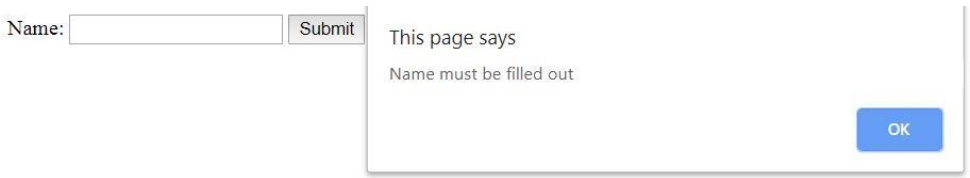
Form Validation

- It is the process of checking that a form has been filled in correctly before it is processed or not.
- For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form.
- There are two main methods for validating forms: server-side (using CGI scripts, ASP, etc.), and client-side (usually done using JavaScript).
- Server-side validation is more secure but often more tricky to code and it also increases load of server computer, whereas client- side validation is easier to do and quicker too because the browser doesn't have to connect to the server to validate the form, so the user finds out instantly if they've missed out that required field).
- Client-side validation also decreases the load of server computer and hence server computer can focus on business logic processing.

Form Validation: Checking Non-Empty

```
<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="validateForm()"
method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Output:



Form Validation: Checking Numbers

```
<script type='text/javascript'>  function
validate()
{
var patt= /^[0-9]+$/ ;
var v=
document.getElementById('elem').value;
if(v.match(patt))
 {       alert("valid entry");     }  else
 {
 alert("Invalid entry:");
 document.getElementById('elem').value="";
 document.getElementById('elem').focus();
 }
 }
 </script>
 <form>
 Required Field: <input type='text' id='elem'/>
 <input type='button' onclick="validate()" value='Check'/
 >
 </form>
```

Output:

Required Field: Check

Form Validation: Checking All Letters

```
<script type='text/javascript'>  function
validate()
{
var patt= /^[a-zA-Z]+$/ ;
var v=
document.getElementById('elem').value;
if(v.match(patt))
{       alert("valid entry");     }  else
{
alert("Invalid entry:");
document.getElementById('elem').value="";
document.getElementById('elem').focus();
}
}
</script>
<form>
Required Field: <input type='text' id='elem'/>
<input type='button' onclick="validate()" value='Check'/
>
</form>
```

Output:

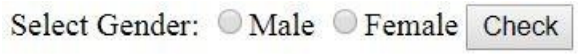
Required Field: Check

Form Validation: Radio Buttons

```
<script type='text/javascript'>  function
validate()
{
var sex=document.getElementsByName("gen");
if(sex[0].checked==false &&
sex[1].checked==false)
{   alert("You must choose Gender");   }

else {  if(sex[0].checked==true)  alert("Male");
else  alert("Female");
}
}
</script>
<form>  Select Gender:
<input type=radio name='gen'>Male
<input type=radio name='gen'>Female
<input type='button' onclick='validate()' value='Check'/>
</form>
```

Output:



Form Validation: Selection Made

```
<script type='text/ javascript'>  function validate()
{
var si=document.getElementById('con').selectedIndex;
var v=
document.getElementById('con').options[si].text;

if(v=="Please Choose")
{     alert("You must choose the country");       }
else
{     alert("Your Country is:"+v);     }
}
</script>
<form>
Select Country: <select id='con'>
<option>Please Choose</option>
<option>Nepal</option>
<option>India</option> <option>China</option>
</select>
<input type='button' onclick='validate()' value='Check'/>
</form>
```

Output:

Form Validation: Email

- Every email is made up of 5 parts:
  - A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores.
  - The @ symbol.
  - A combination of letters, numbers, hyphens, and/or periods.
  - A period (.).
  - The top level domain (com, net, org, gov, …).

Form Validation: Email

```
<script type='text/ javascript'>  function validate()
{
var patt=/^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]
{2,4}$/;  var v= document.getElementById('elem').value;
if(v.match(patt))


{      alert("valid
Email");   }
else
{
alert("Invalid Email"); document.getElementById('elem').value="";
document.getElementById('elem').focus();
}
}
</script>
<form>
Email ID: <input type='text' id='elem'/>
<input type='button' onclick="validate()" value='Check'/>
</form>
```

Output:

Handling Cookies in JavaScript

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user?"
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie remembers his/her name.
- Cookies are saved in name-value pairs like: username = John Doe
- When a browser requests a web page from a server, cookies belonging to the page are added to the request; this way the server gets the necessary data to remember information about users.
- JavaScript can create, read, and delete cookies with the document.cookie property.

Handling Cookies in JavaScript

- Examples of cookies are:
  - Name Cookie – the first time a visitor arrives to your web page, he/she must fill in his/her name. The name is then stored in a cookie. Next time the visitor arrives at your page, he/she could get a welcome message like “Welcome John Doe!”. The name is retrieved from the stored cookie.
  - Password Cookie – the first time a visitor arrives to your web page, he/she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie.
  - Date Cookie – the fist time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he/she could get a message like “Your last visit was on Tuesday July 30, 2019!”. The date is retrieved from the stored cookie.

Create a Cookie with JavaScript

- With JavaScript, a cookie can be created like this:

  document.cookie = "username=John Doe";
- You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

  document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
- With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

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

Read a Cookie with JavaScript

- With JavaScript, a cookie can be read like this:

  var x = document.cookie;

Change a Cookie with JavaScript

- With JavaScript, you can change a cookie the same way as you create it:
  document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
- The old cookie is overwritten

Delete a Cookie with JavaScript

- Deleting a cookie is very simple as you don't have to specify a cookie value when you delete it.
- Just set the expires parameter to a passed date:
  document.cookie = "username; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

jQuery

- jQuery is a powerful and widely used JavaScript library to simplify common web scripting task.
- It is based on the principle “write less, do more”.
- The purpose of jQueryis to make it much easier to use JavaScript.
- The jQuery library contains the following features:
  - HTML manipulation
  - CSS manipulation
  - HTML event methods
  - Effects and Animations
  - AJAX
  - Utilities

What you can do with jQuery?

- Easily select elements to perform manipulations.
- Easily create effect like show or hide elements, sliding transition, and so on.
- Easily create complex CSS animation with fewer lines of code.
- Easily manipulate DOM elements and their attributes.
- Easily implement Ajax to enable asynchronous data exchange between client and server.
- Easily traverse all aroundthe DOM tree to locate any element.
- Easily perform multiple actions on an element with a single line of code.
- Easily get or set dimensions of the HTML elements.

Advantages of jQuery?

- Save lots of time
- Simplify common JavaScript tasks
- Easy to use
- Compatible with browsers
- Absolutely free

jQuery Syntax

- A jQuery statement starts with the dollar sign (**$**) and ends with a semicolon (**;**).
- The dollar sign (**$**) is just an alias for jQuery.
- Basic syntax for jQuery is: ***$(selector).action();***

```
<script>
$(document).ready(function(){
// Some code to be executed... alert("HelloWorld!"); });
</script>
```

- **<script>** element – since jQuery is just a JavaScript library, the jQuery code can be placed inside the *<script>* element. However, if you want to place it in an external JavaScript file, which is preferred, you just remove this part.
- **$(document).ready(handler);** – this statement is known as ready event where the *handler* is basically a function that is passed to the *ready()* method to be executed safely as soon as the document is ready to be manipulated.

jQuery Selectors

- jQuery selectors allow us to select and manipulate HTML elements.
- Current Element Selector:
  - ***$(this).hide()*** – hides the current element.
- Element Selector:
  - ***$("p").hide()*** – hides all <p> elements.
- .class Selector:
  - ***$(".test").hide()*** – hides the elements with class = "test".
- #id Selector:
  - ***$("#test").hide()*** – hides the element with id = "test".

jQuery Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(this).hide();
    //$("p").hide();
    //$("#test").hide();
    //$(".test").hide();
  });
});
</script>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<p id="test">This is another paragraph with id.</p>
<p class="test">This is another paragraph with class.</p>
<button>Click me to hide paragraphs</button>
</body>
</html>
```

jQuery

Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first element of the first
$("ul li:first-child")	Selects the first element of every
$("[href]")	Selects all elements with an href attribute
$ ("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target! ='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":button")	Selects all <button> elements and <input> elements of type="button"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

jQuery Events

- Events are often triggered by the user's interaction with the web page, such as:
  - when a link or button is clicked
  - text is entered into an input box or textarea
  - selection is made in a select box
  - key is pressed on the keyboard
  - the mouse pointer is moved, etc.
- In some cases, the browser itself triggers the event, such as the page load and unload events.

Mouse Events	Keyboard Events	Form Events	Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

jQuery Mouse Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
   p{  padding: 20px; font: 20px sans-serif;  background: aqua;  }
</style>
<script>
$(document).ready(function(){
$("p").click(function(){
$(this).slideUp();
});
});
</script>
</head>
<body>
   <p>Click on me and I'll disappear.</p>
   <p>Click on me and I'll disappear.</p>
   <p>Click on me and I'll disappear.</p>
</body>
</html>
```

jQuery Keyboard Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{  padding: 10px; background: lightgreen; display: none; }  div{  margin: 20px 0;  }
</style>
<script>
$(document).ready(function(){
  var i = 0;
  $('input[type="text"]').keypress(function(){
    $("span").text(i += 1);
    $("p").show().fadeOut();
  });
});
</script>
</head>
<body>
  <input type="text">
  <div>Keypress: <span>0</span></div>
                    <div><strong>Note:</strong> Enter something inside the input box and see the result.</div>
  <p>Keypress is triggered.</p>
</body>
</html>
```

jQuery Form Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<script>
$(document).ready(function(){
  $("select").change(function(){
    var selectedOption = $(this).find(":selected").val();  alert("You have selected - " + selectedOption);
  });
});
</script>
</head>
<body>
  <form>
    <label>City:</label>
    <select>
      <option>Kathmandu</option>
      <option>Pokhara</option>
      <option>Chitwan</option>
    </select>
  </form>
<p><strong>Note:</strong> Select any value from the dropdown select and see the result.</p>
</body>
</html>
```

jQuery Form Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  label{  display: block; margin: 5px 0;  }  label span{  display: none;  }
</style>
<script>
$(document).ready(function(){
  $("input").focus(function(){
    $(this).next("span").show().fadeOut("slow");
  });
});
</script>
</head>
<body>
  <form>
    <label>Email: <input type="text"> <span>focus fire</span></label>
    <label>Password: <input type="password"> <span>focus fire</span></label>
    <label><input type="submit" value="Sign In"> <span>focus fire</span></label>
  </form>
  <p><strong>Note:</strong> Click on the form control or press the "Tab" key to set focus.</p>
</body>
</html>
```

jQuery Window Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{  padding: 20px; font: 20px sans-serif; background: #f0e68c; }
</style>
<script>
$(document).ready(function(){
  $(window).resize(function() {
    $(window).bind("resize", function(){
      $("p").text("Window width: " + $(window).width() + ", " + "Window height: " + $
      (window).height());
    });
  });
});
</script>
</head>
<body>
  <p>Open the output in a new tab and resize the browser window by dragging the corners.</p>
</body>
</html>
```

jQuery Window Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
   p{ width: 100%; padding: 50px 0; text-align: center; font: bold 34px sans-serif;
   background:

#f0e68c; position: fixed; top: 50px; display: none; }
   .dummy-content{ height: 600px; font: 34px sans-serif; text-align: center; }
</style>
<script>
$(document).ready(function(){
   $(window).scroll(function() {
      $("p").show().fadeOut("slow");
   });
});
</script>
</head>
<body>
   <p>Scroll Happened!</p>
   <div class="dummy-content">Scroll the viewport.</div>
   <div class="dummy-content">Scroll the viewport.</div>
   <div class="dummy-content">Scroll the viewport.</div>
   <div class="dummy-content">Scroll the viewport.</div>
   <div class="dummy-content">Scroll the viewport.</div>
</body>
</html>
```

Introduction to JSON

- JSON is a syntax for storing and exchanging data.
- JSON stands for JavaScript Object Notation.
- It is a lightweight text based data-interchange format.
- When exchanging data between a browser and a server, the data can only be text.
- It is easy to understand, manipulate, and generate.
- It is based on a subset of the JavaScript Programming Language.

Why use JSON?

- Straightforward syntax
- Easy to create and manipulate
- Supported by all major JavaScript frameworks
- Supported by most backend technologies

JSON Object Syntax

- Unordered sets of name/value pairs
- Begins with **{** (left brace)
- Ends with **}** (right brace)
- Each name is followed by **:** (colon)
- Name/Value pairs are separated by **,** (comma)

JSON Example

```
<html>
<body>
<p>Access a JavaScript object:</p>
<p id="demo"></p>
<script>  var myObj, x;
myObj = { name: "John", age: 30, city: "NewYork" };
x = myObj.name;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

END OF UNIT FOUR