Web Programming

YJ – 2016

JavaScript

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

```
<html>
   <body>

      <script language="javascript" type="text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>

   </body>
</html>
```

Advantages

- **Less server interaction −** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors −** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity −** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces −** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities

Type

- ❖ Internal

```
<html>
   <head>
      <script type="text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>
   </head>

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

      <input type="button" onclick="sayHello()" value="Say Hello" />

   </body>
</html>
```

Type

❖ External

```
<html>


   <head>
      <script type="text/javascript" src="filename.js" ></script>
   </head>


   <body>
      .......
   </body>
</html>
```

Variable

```
var surname;
var age;
var name = "Tom";
age = 43;
var apples = 5, pears = 10;

var piecesOfFruit = apples + pears;

(10 + 2) / 2 + 4 * 2
```

Variable Scope

```
<html>
   <body onload = checkscope();>
      <script type = "text/javascript">
         <!--
            var myVar = "global"; // Declare a global variable
            function checkscope( ) {
               var myVar = "local";  // Declare a local variable
               document.write(myVar);
            }
         //-->
      </script>
   </body>
</html>
```

This produces the following result −

```
local
```

Reserved Words

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

Operator

- + (Addition)

Adds two operands
Ex: A + B will give 30

- - (Subtraction)

Subtracts the second operand from the first
Ex: A - B will give -10

- * (Multiplication)

Multiply both operands
Ex: A * B will give 200

- / (Division)

Divide the numerator by the denominator
Ex: B / A will give 2

- % (Modulus)

Outputs the remainder of an integer division
Ex: B % A will give 0

- ++ (Increment)

Increases an integer value by one
Ex: A++ will give 11

- -- (Decrement)

Decreases an integer value by one
Ex: A-- will give 9

Comparison Operators

Sr.No	Operator and Description
1	**= = (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.

Logical Operators

Sr.No	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

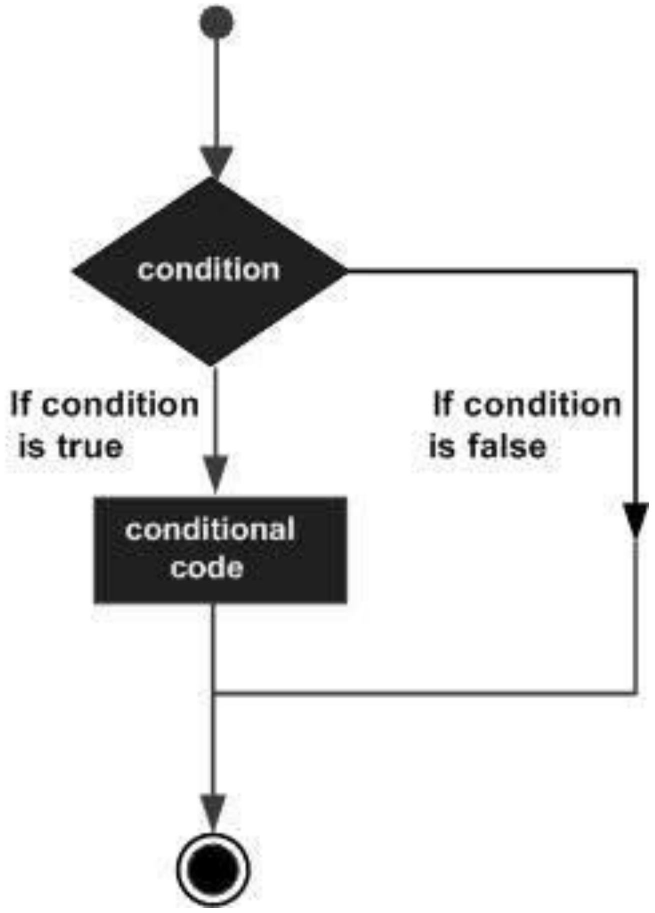
Bitwise Operators

Sr.No	Operator and Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand.
5	**<< (Left Shift)** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.
7	**>>> (Right shift with Zero)** This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

Assignment Operators

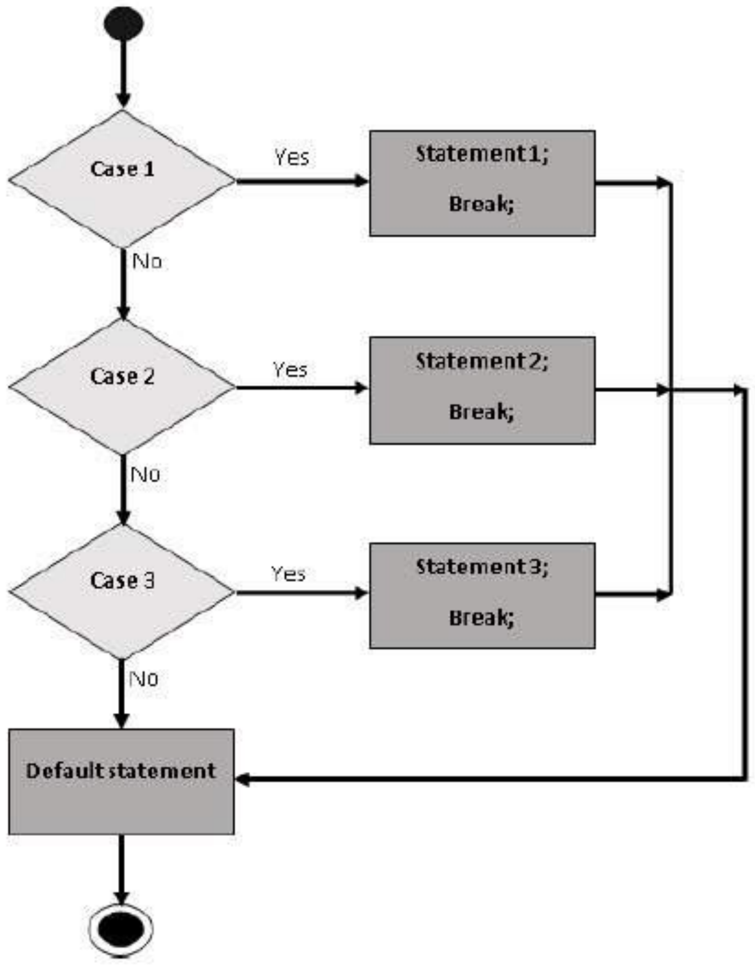
Sr.No	Operator and Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Condition



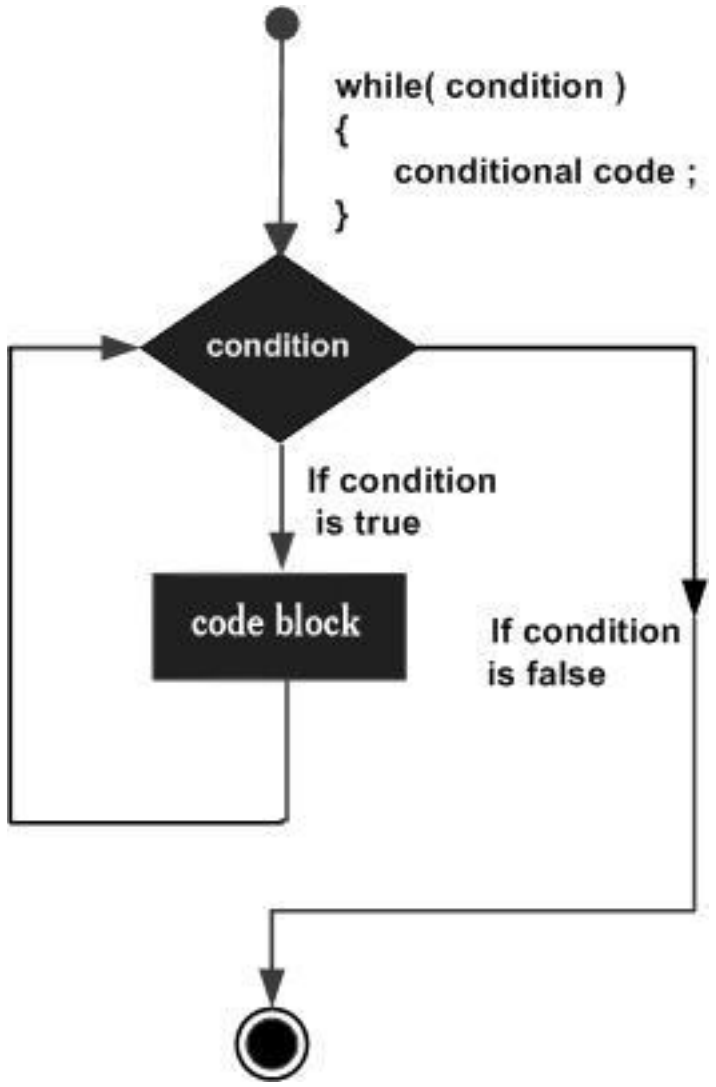
```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}

else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}

else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}

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

Condition



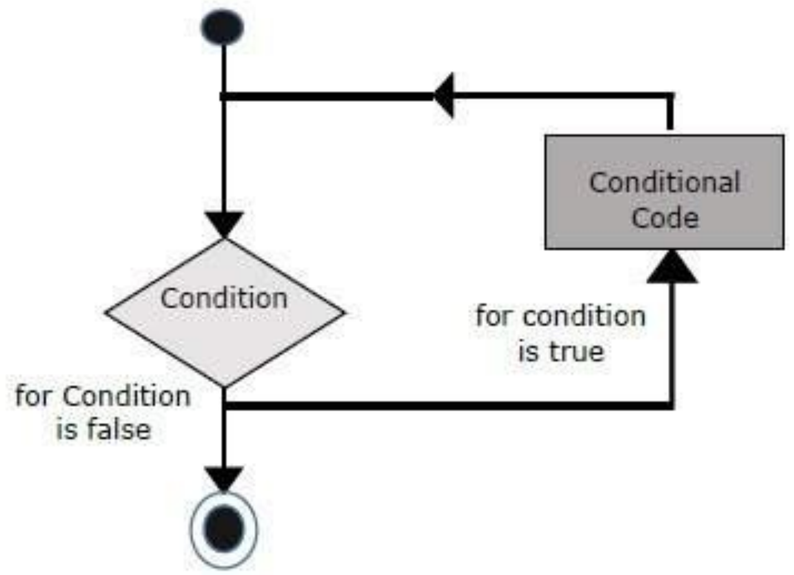
```
switch (expression)
{
   case condition 1: statement(s)
   break;

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

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

Loop



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

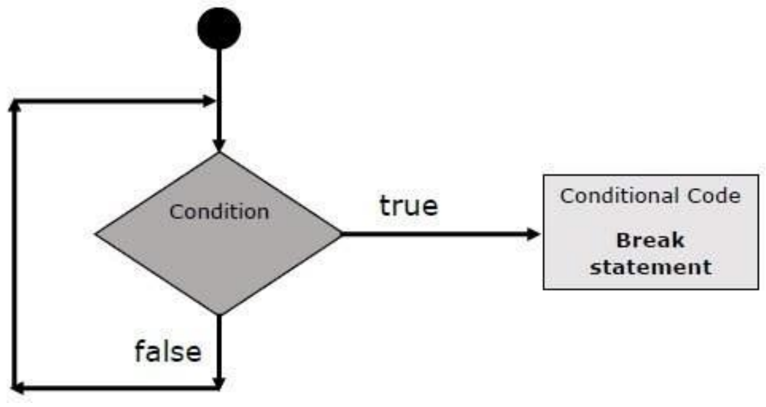
Loop



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

```
for(count = 0; count < 10; count++){

document.write("Current Count : " + count );

document.write("<br />");

}
```

Loop Control



break Statement

continue Statement

Function

```
<script type="text/javascript">
  <!--
    function functionname(parameter-list)
    {
      statements
    }
  //-->
</script>
```

```
<html>
   <head>
      <script type="text/javascript">
         function sayHello()
         {
            document.write ("Hello there!");
         }
      </script>

   </head>
   <body>
      <form>
         <input type="button" onclick="sayHello()" value="Say Hello">
      </form>
   </body>
</html>
```

Event

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function validation() {
               all validation goes here
               .........
               return either true or false
            }
         //-->
      </script>

   </head>
   <body>

      <form method="POST" action=“" onsubmit="return
validate()">
         .......
         <input type="submit" value="Submit" />
      </form>

   </body>
</html>
```

Event

Offline	ondragenter	onmouseout	onreadystatechange
Onabort	ondragleave	onmouseover	onredo
onafterprint	ondragover	onmouseup	onresize
onbeforeonload	ondragstart	onmousewheel	onscroll
onbeforeprint	ondrop	onoffline	onseeked
onblur	ondurationchange	onoine	onseeking
oncanplay	onemptied	onsuspend	onselect
oncanplaythrough	onended	ontimeupdate	onstalled
onchange	onerror	onundo	onstorage
onclick	onfocus	onunload	onsubmit
oncontextmenu	onformchange	onvolumechange	onwaiting
ondblclick	onforminput	onloadedmetadata	onplay
ondrag	onhaschange	onloadstart	onplaying
ondragend	oninput	onmessage	onpopstate
onkeydown	oninvalid	onmousedown	onprogress
onkeypress	ononline	onmousemove	onratechange
onkeyup	onpagehide	onloadeddata	onpause
onload	onpageshow		

Re-direction

```
function Redirect() {

window.location="http://www.google.com";

}

document.write("You will be redirected to main page in 10 sec.");

setTimeout('Redirect()', 10000);
```

Dialog Boxes

- Alert()

```
alert ("This is a warning message!");
```

- Confirm()

```
var retVal = confirm("Do you want to continue ?");
          if( retVal == true ){
            document.write ("User wants to continue!");
            return true;
          }
```

- Promote()

```
var retVal = prompt("Enter your name : ", "your name here");
```

Object

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers

- ❖ **Encapsulation** − the capability to store related information, whether data or methods, together in an object.
- ❖ **Aggregation** − the capability to store one object inside another object.
- ❖ **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- ❖ **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Object

```
var val = new Number(number);
var val = new Boolean(value);
var val = new String(string);

var fruits = new Array( "apple", "orange", "mango" );
```

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element

```
new Date( )
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond ])

var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

String

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

```
var val = new String(string);
```

Array

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

```
var fruits = new Array( "apple", "orange", "mango" );
```

```
var fruits = [ "apple", "orange", "mango" ];
```

Date

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date( )** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

```
new Date(year,month,date[,hour,minute,second,millisecond ])
```

Number

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

```
var val = new Number(number);
```

Math

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

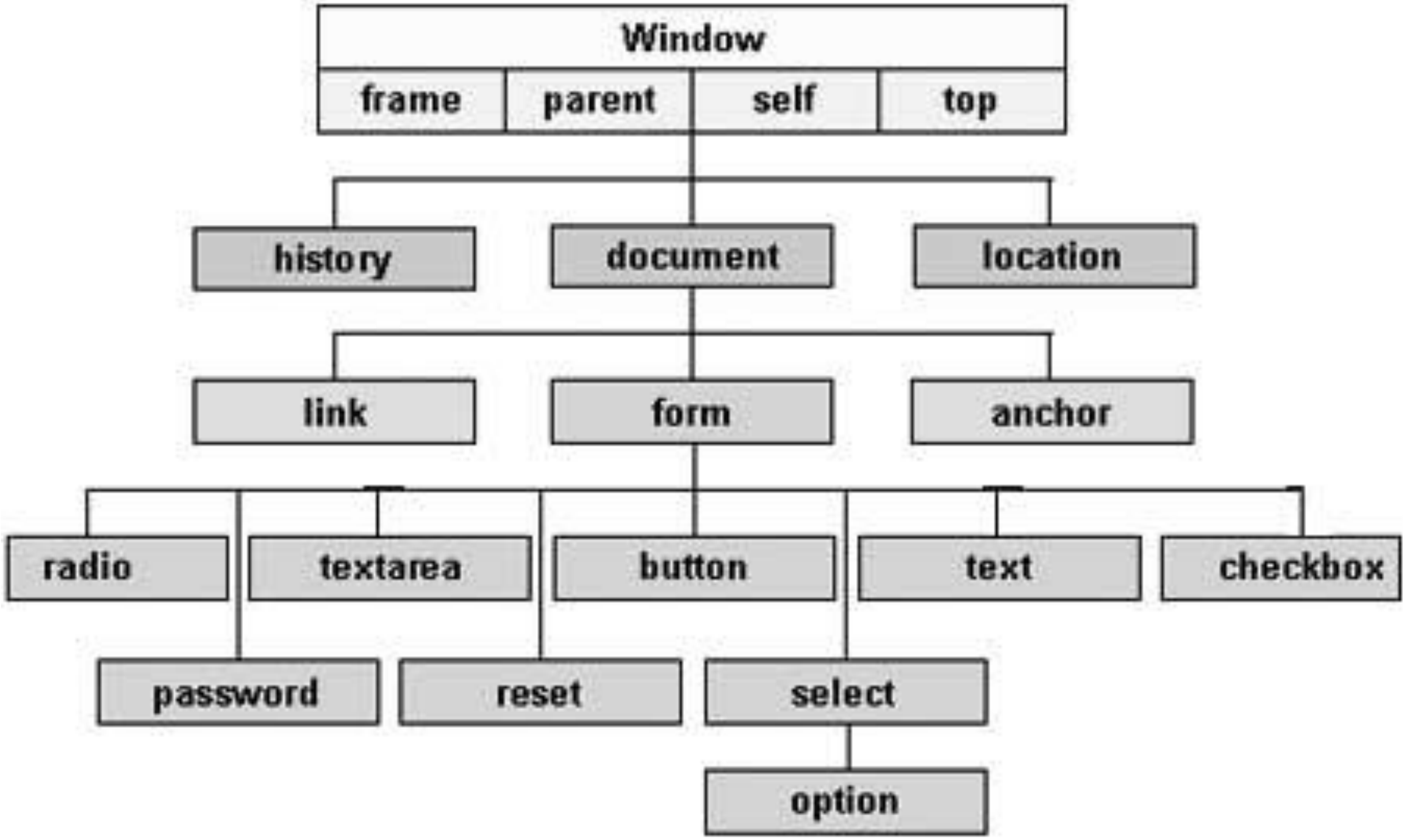
```
var pi_val = Math.PI;

var sine_val = Math.sin(30);
```

DOM

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- ❖ **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- ❖ **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- ❖ **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- ❖ **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

DOM


Window
frame
parent
self
top
history
document
location
link
form
anchor
radio
textarea
button
text
checkbox
password
reset
select
option

Error Handlling

There are three types of errors in programming:
(a) Syntax Errors
(b) Runtime Errors
(c) Logical Errors

```
<script type="text/javascript">
   <!--
      try {
         // Code to run
         [break;]
      }

      catch ( e ) {
         // Code to run if an exception occurs
         [break;]
      }

      [ finally {
         // Code that is always executed regardless of
         // an exception occurring
      }]
   //-->
</script>
```

Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

1. **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.

2. **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Animation

JavaScript can be used to move a number of DOM elements (<img />, <div> or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.
JavaScript provides the following two functions to be frequently used in animation programs.

•**setTimeout(function, duration)**

− This function calls **function** after **duration** milliseconds from now.

•**setInterval(function, duration)**

− This function calls **function** after every **duration** milliseconds.

•**clearTimeout(setTimeout_variable)**

•− This function calls clears any timer set by the setTimeout() functions.

jQuery

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code.

jQuery is a framework built using JavaScript capabilities. So while developing your applications using jQuery, you can use all the functions and other capabilities available in JavaScript.

- **DOM manipulation** − The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.

- **Event handling** − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.

- **AJAX Support** − The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.

- **Animations** − The jQuery comes with plenty of built-in animation effects which you can use in your websites.

jQuery

jQuery is a framework built using JavaScript capabilities. So while developing your applications using jQuery, you can use all the functions and other capabilities available in JavaScript.

```
<html>


   <head>
      <title>The jQuery Example</title>
      <script type = "text/javascript"  src = "/jquery/jquery-
2.1.3.min.js"></script>

      <script type = "text/javascript">
         $(document).ready(function(){
            document.write("Hello, World!");
         });
      </script>
   </head>

   <body>
      <h1>Hello</h1>
   </body>

</html>
```

Selector

jQuery selectors start with the dollar sign and parentheses − **$()**. The factory function **$()** makes use of following three building blocks while selecting elements in a given document

S.N.	Selector & Description
1	**Tag Name** Represents a tag name available in the DOM. For example **$('p')**selects all paragraphs <p> in the document.
2	**Tag ID** Represents a tag available with the given ID in the DOM. For example**$('#some-id')** selects the single element in the document that has an ID of some-id.
3	**Tag Class** Represents a tag available with the given class in the DOM. For example **$('.some-class')** selects all elements in the document that have a class of some-class.

Attribute

The **attr()** method can be used to either fetch the value of an attribute from the first element in the matched set or set attribute values onto all matched elements.

Some of the more common properties are: className, tagName, id, href, title, rel, src

Get Attribute Value

```
<script type = "text/javascript" language = "javascript">
   $(document).ready(function() {
      var title = $("em").attr("title");
      $("#divid").text(title);
   });
</script>
```

Set Attribute Value

```
<script type = "text/javascript" language = "javascript">
   $(document).ready(function() {
      $("#myimg").attr("src", "/jquery/images/jquery.jpg");
   });
</script>
```

Applying Styles

```
<script type = "text/javascript" language = "javascript">
   $(document).ready(function() {
      $("em").addClass("selected");
      $("#myid").addClass("highlight");
   });
</script>
```

Traversing

Find Elements by index

```
<script type = "text/javascript" language = "javascript">
         $(document).ready(function() {
            $("li").eq(2).addClass("selected");
         });
      </script>
```

Filtering out Elements

```
<script type = "text/javascript" language = "javascript">
         $(document).ready(function() {
            $("li").filter(".middle").addClass("selected");
         });
</script>
```

Locating descendant Elements

```
<script type = "text/javascript" language = "javascript">
         $(document).ready(function() {
            $("p").find("span").addClass("selected");
         });
      </script>
```

CSS

Apply CSS Properties

```
<script type = "text/javascript" language = "javascript">
$(document).ready(function() {
$("li").eq(1).css("color", "red");
$("li").eq(2).css({"color":"red", "background-color":"green"});
});
</script>

<div>
<ul>
<li>list item 1</li>
<li>list item 2</li>
<li>list item 3</li>
<li>list item 4</li>
<li>list item 5</li>
<li>list item 6</li>
</ul>
</div>
```

DOM Manipulation

JQuery provides methods to manipulate DOM in efficient way. You do not need to write big code to modify the value of any element's attribute or to extract HTML code from a paragraph or division.

Content Manipulation

```
var content = $(this).html();
$("#result").text( content );
```

DOM Element Replacement

```
$("div").click(function () {
   $(this).replaceWith("<h1>JQuery is Great</h1>");
});
```

Removing DOM Elements

```
$("div").click(function () {
        $(this).remove( );
});
```

Inserting DOM elements

```
$("div").click(function () {
        $(this).before('<div class="div"></div>' );
});
```

Events

We have the ability to create dynamic web pages by using events. Events are actions that can be detected by your Web Application.

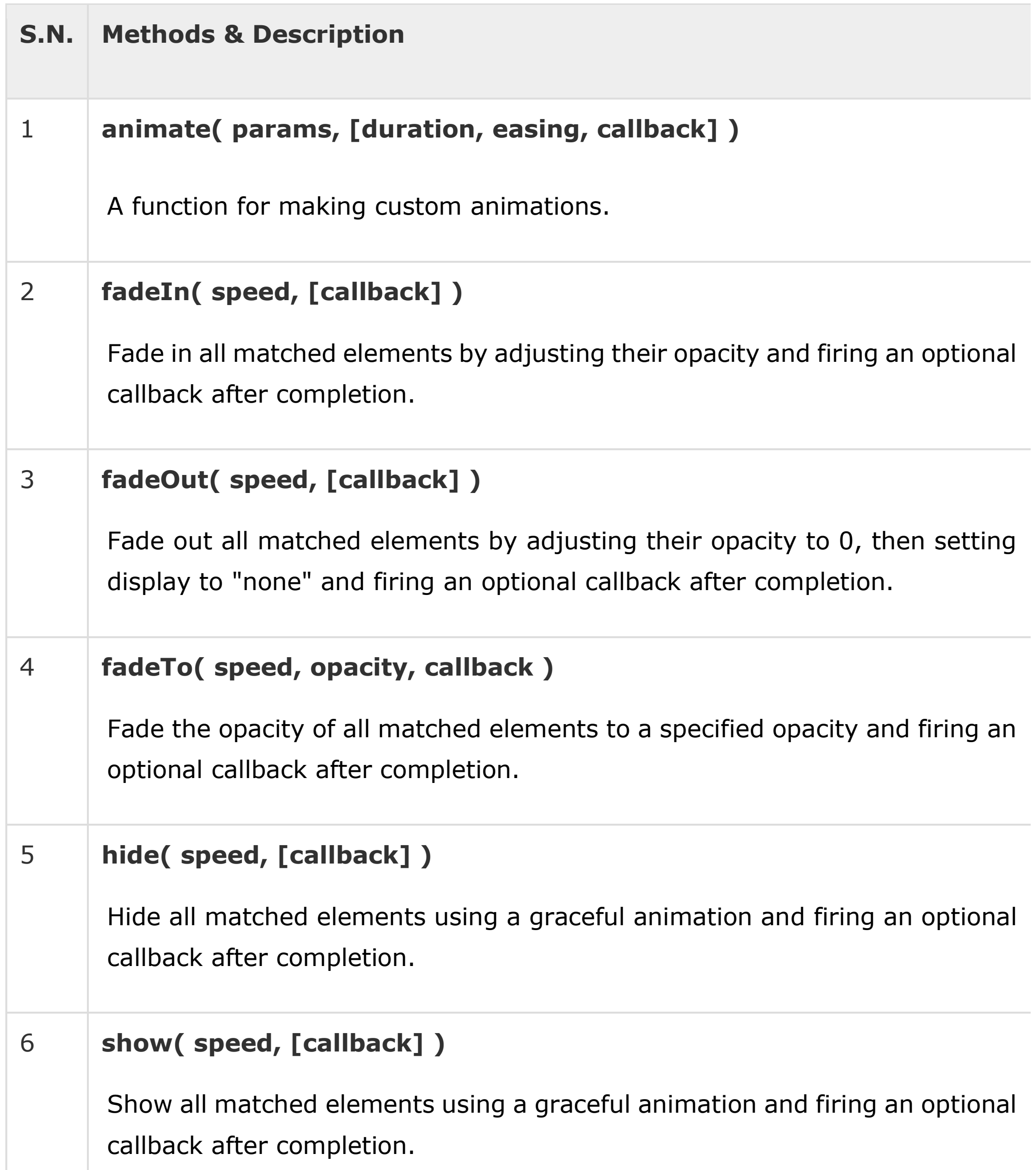
Following are the examples events −

- A mouse click
- A web page loading
- Taking mouse over an element
- Submitting an HTML form
- A keystroke on your keyboard

```
$( "#foo" ).bind( "click", function() {
  alert( "User clicked on 'foo.'" );
});

$( "#foo" ).bind( "mouseenter mouseleave", function() {
  $( this ).toggleClass( "entered" );
});

$( "form" ).bind( "submit", function() {
  return false;
})
```

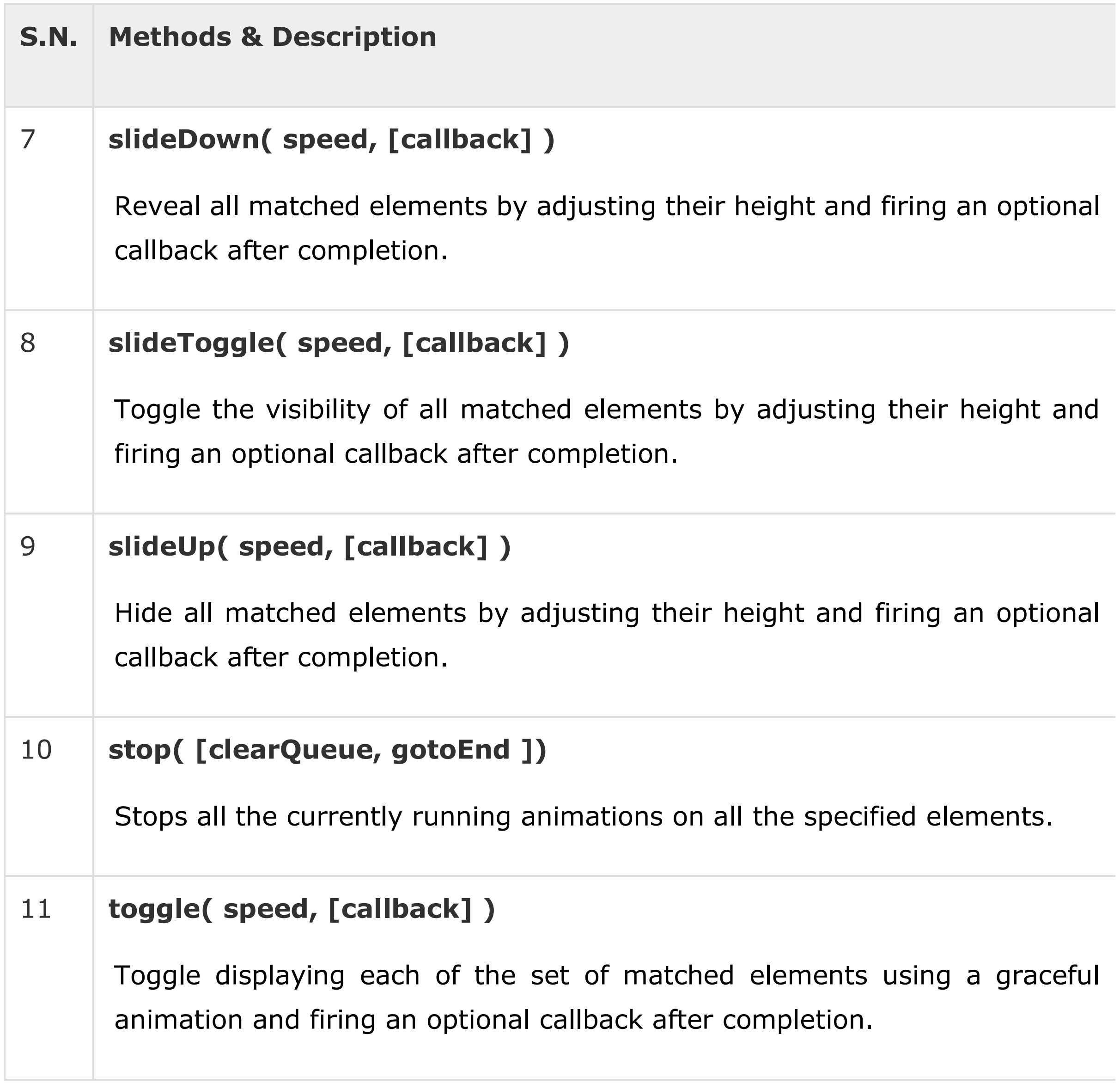
Animations

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.



S.N.	Methods & Description
1	**animate(params, [duration, easing, callback])** A function for making custom animations.
2	**fadeIn(speed, [callback])** Fade in all matched elements by adjusting their opacity and firing an optional callback after completion.
3	**fadeOut(speed, [callback])** Fade out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.
4	**fadeTo(speed, opacity, callback)** Fade the opacity of all matched elements to a specified opacity and firing an optional callback after completion.
5	**hide(speed, [callback])** Hide all matched elements using a graceful animation and firing an optional callback after completion.
6	**show(speed, [callback])** Show all matched elements using a graceful animation and firing an optional callback after completion.

Animations



S.N.	Methods & Description
7	**slideDown(speed, [callback])** Reveal all matched elements by adjusting their height and firing an optional callback after completion.
8	**slideToggle(speed, [callback])** Toggle the visibility of all matched elements by adjusting their height and firing an optional callback after completion.
9	**slideUp(speed, [callback])** Hide all matched elements by adjusting their height and firing an optional callback after completion.
10	**stop([clearQueue, gotoEnd])** Stops all the currently running animations on all the specified elements.
11	**toggle(speed, [callback])** Toggle displaying each of the set of matched elements using a graceful animation and firing an optional callback after completion.