CACS 205: Script Language

PREPARED BY KRISHNA PD. ACHARYA

(MECHI MULTIPLE CAMPUS)

Java Script

- JavaScript is used to create client-side dynamic web pages.
- JavaScript is an object-based scripting language which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- JavaScript is a compact, object-based scripting language for developing client Internet applications.
- JavaScript was designed to add interactivity to HTML pages.
- A JavaScript is lines of executable computer code.
- A JavaScript is usually embedded directly in HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation).
- Everyone can use JavaScript without purchasing a license.
- All major browsers, like Netscape and Internet Explorer, support JavaScript.
- JavaScript was developed by Netscape as Live Script - changed to JavaScript when endorsed by Sun micro system 1993, version 1.0 released with Netscape 2.0.
- JavaScript is a powerful scripting language that is also capable of performing extensive form data collecting and form processing functions.

Java Script

- JavaScript / ECMAScript
- JavaScript was developed for Netscape. Netscape 2 was the first browser to run JavaScript.
- After Netscape the Mozilla foundation continued to develop JavaScript for the Firefox browser.
- The latest JavaScript version was 1.8.5. (Identical to ECMAScript 5).
- ECMAScript was developed by ECMA International after the organization adopted JavaScript.
- The first edition of ECMAScript was released in 1997.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMAScript is the official name of the language.
- From 2015 ECMAScript is named by year (ECMAScript 2015).
- ECMAScript is often abbreviated to ES.
- ECMAScript 3 is fully supported in all browsers.
- ECMAScript 5 is fully supported in all modern browsers.

Advantages of Java Script

- Cross Browser supporting:

This means that the JavaScript codes are supported by various browsers like Internet Explorer, Netscape Navigator, Mozilla etc.

- Platform Independent:

JavaScript codes can be executed in any platform like Linux, Microsoft Windows and many other operating systems.

- Lightweight for fast downloading:

The JavaScript codes runs directly in the client machine. The code should be downloaded all the way from server to the client and this time duration is very minimum and the executing the codes of JavaScript is also very fast.

- Validating Data:

The data can be validated in the two different way:

Validating in server side or in server machine.
Validating in client side or in client machine.

In this two different types of validation of data the second one is much more faster, and this is done through JavaScript.

Advantages of Java Script

- Sophisticated User Interfaces:

By using JavaScript you can create a user interactive interfaces that can communicate with the user and the Browser.

- In-Built software:

To you don't need any extra tools to write JavaScript, any plain text or HTML editor will do, so there's no expensive development software to buy.

- Easy to Learn:

The JavaScript programmer should know the minimal syntax of JavaScript since it supports many syntax and data types as C and C++.

It's also an easy language to learn, and there's a thriving and supportive online community of JavaScript developers and information resources.

- Designed for programming User-Events:

JavaScript supports Object/Event based programming. So the code written in JavaScript can easily be break down into sub-modules.

Disadvantages of Java Script

- Launching an application on the client computer.

JavaScript is not used to create stand-alone application; it is only used to add some functionality in any web page.

- Reading or writing files:

JavaScript cannot read and write files into the client machines. It can only be used as a utility language to develop any web site.

- Retrieving the text contents of HTML pages or files from the server.
- Reading and Writing files to the server:

JavaScript can read and write to any file in the server as well.

- Sending secret e-mails from Web site visitors to you:
- JavaScript cannot be used to send email to the visitors or user of the web site. This can be done only with the server side scripting.
- Cannot create Database Application:

By using JavaScript you cannot connect the web site to the database. For this you need to use server-side scripting.

- Browser Compatibility Issues:

Not all browsers support the JavaScript codes. The browser may support JavaScript as a whole, but may not support the codes or lines of codes written in JavaScript and may interpret differently.

- JavaScript does not implement multiprocessing or multithreading.
- Use printers or other devices on the user's system or the client-side LAN.
- JavaScript has limitations of writing in a client machine. It can only write the cookie in client machine that is also of a certain size i.e. 4K.

Application of Java Script

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.
- Provide event of based on user action to design CSS and HTML.
- JavaScript can update and change both HTML and CSS
- JavaScript can calculate, manipulate and validate data

What it does?

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles CSS)
- JavaScript Can Hide HTML Elements
- JavaScript Can Show HTML Elements

Comparision java and Java Script

JavaScript	Java
JavaScript is a small, lightweight programming language.	Java is a full-blown powerful, sophisticated programming language.
Developed by Netscape communications.	Developed by Sun Microsystems.
Scripting language interpreted at runtime.	True programming compiled to byte-code.
Object-based, has limited number of built-in objects.	Object-oriented, can create their own classes.
Not fully extensible.	Fully extensible
Code integrated with, and embedded in HTML.	Applies distinct from HTML (accessed from HTML pages).
Variables type must not be declared.	Variables type must be declared.
JavaScript source code can be viewed by everybody using "view source command".	Your Java source is hidden because it's only the compiled byte-code, which the browser uses, but this is not a guarantee of security.

Coding Conventions

Coding conventions are style guidelines for programming. Coding conventions can be documented rules for teams to follow, or just be your individual coding practice. They typically include:

- Naming and declaration rules for variables and functions.
- Rules for the use of white space, indentation, and comments.
- Programming practices and principles
- Coding conventions secure quality: which may include improves code readability and make code maintenance easier.

Naming Conventions

- Always use the same naming convention for all your code. For example:
- Variable and function names written as camelCase
- Global variables written in UPPERCASE (We don't, but it's quite common)
- Constants (like PI) written in UPPERCASE
- Should you use hyp-hens, camelCase, or under_scores in variable names.

Hyphens in HTML and CSS:

- HTML5 attributes can start with data- (data-quantity, data-price).
- CSS uses hyphens in property-names (font-size).
- Hyphens are not allowed in JavaScript names.

Underscores:

- Many programmers prefer to use underscores (date_of_birth), especially in SQL databases.
- Underscores are often used in PHP documentation.

Coding Conventions

File Extensions

- HTML files should have a .html extension (not .htm).
- CSS files should have a .css extension.
- JavaScript files should have a .js extension.

Use Lower Case File Names

- Most web servers (Apache, Unix) are case sensitive about file names:
- london.jpg cannot be accessed as London.jpg.
- Other web servers (Microsoft, IIS) are not case sensitive:
- london.jpg can be accessed as London.jpg or london.jpg.
- If you use a mix of upper and lower case, you have to be extremely consistent.
- If you move from a case insensitive, to a case sensitive server, even small errors can break your web site.
- To avoid these problems, always use lower case file names (if possible).

Performance

- Coding conventions are not used by computers. Most rules have little impact on the execution of programs.
- Indentation and extra spaces are not significant in small scripts.
- For code in development, readability should be preferred. Larger production scripts should be minified.

Coding Conventions

Script Best Practices:

Avoid global variables, avoid new, avoid = =, avoid eval()

Avoid Global Variables

Minimize the use of global variables.

This includes all data types, objects, and functions.

Global variables and functions can be overwritten by other scripts.

Always Declare Local Variables

All variables used in a function should be declared as local variables.

Local variables must be declared with the **var** keyword or the **let** keyword, otherwise they will become global variables.

Strict mode does not allow undeclared variables.

Declarations on Top

It is a good coding practice to put all declarations at the top of each script or function. This will:

Give cleaner code

Provide a single place to look for local variables

Make it easier to avoid unwanted (implied) global variables

Reduce the possibility of unwanted re-declarations.

By default, JavaScript moves all declarations to the top.

Initialize Variables

It is a good coding practice to initialize variables when you declare them. This will:

Give cleaner code

Provide a single place to initialize variables

Avoid undefined values

JavaScript Comments

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.
- JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

- Single line comments start with //.
- Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

```
// Change heading:
// heading h1 has the id = t1
document.getElementById(“t1").innerHTML = "My First Page";
```

Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored by JavaScript.

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById(“t1").innerHTML = "My First Page";
```

<noscript> tag in java Script

- The <noscript> tag defines an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support script.
- The <noscript> element can be used in both <head> and <body>.
- When used inside the <head> element: <noscript> must contain only <link>, <style>, and <meta> elements.
- The content inside the <noscript> element will be displayed if scripts are not supported, or are disabled in the user's browser.
- It is also a good practice to use the comment tag to "hide" scripts from browsers without support for client-side scripts (so they don't show them as plain text):

```
<script>
<!--
function displayMsg()  {
  alert("Hello World!")
}
//-->
</script>
```

Embedding script in html

- In HTML, JavaScript code must be inserted between <script> and </script> tags.
- Old JavaScript examples may use a type attribute: <script type="text/javascript">.
- The type attribute is not required. JavaScript is the default scripting language in HTML.

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

<h2>JavaScript in Body</h2>

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

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

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

</body>
</html>
```


- Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

External JavaScript

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag: <script src="myScript.js"></script>

Embedding script in html

- You can place an external script reference in <head> or <body> as you like.
- The script will behave as if it was located exactly where the <script> tag is located.
- External scripts cannot contain <script> tags.

External JavaScript Advantages

- It separates HTML and code.
- It makes HTML and JavaScript easier to read and maintain.
- Cached JavaScript files can speed up page loads.

We can add several script files to one page - use several script tags: Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

```
<script src="https://www.mechicampus.eud.np/js/myScript1.js"></script>
```

Operators in JavaScript

JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.

1. JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that y = 5, the table below explains the arithmetic operators:

Operator	Description	Example	Result in y	Result in x
+	Addition	x = y + 2	y = 5	x = 7
-	Subtraction	x = y - 2	y = 5	x = 3
*	Multiplication	x = y * 2	y = 5	x = 10
/	Division	x = y / 2	y = 5	x = 2.5
%	Modulus (division remainder)	x = y % 2	y = 5	x = 1
++	Increment	x = ++y	y = 6	x = 6
		x = y++	y = 6	x = 5
--	Decrement	x = --y	y = 4	x = 4
		x = y--	y = 4	x = 5

Operators in JavaScript

2. JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that x = 10 and y = 5, the table below explains the assignment operators:

Operator	Example	Same As	Result in x
=	x = y	x = y	x = 5
+=	x += y	x = x + y	x = 15
-=	x -= y	x = x - y	x = 5
*=	x *= y	x = x * y	x = 50
/=	x /= y	x = x / y	x = 2
%=	x %= y	x = x % y	x = 0

3. JavaScript String Operators

The + operator, and the += operator can also be used to concatenate (add) strings. Given that text1 = "Good ", text2 = "Morning", and text3 = "", the table below explains the operators:

Operator	Example	text1	text2	text3
+	text3 = text1 + text2	"Good "	"Morning"	"Good Morning"
+=	text1 += text2	"Good Morning"	"Morning"	""

Operators in JavaScript

4. Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. Given that x = 5, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	*true*

Operators in JavaScript

5. Conditional (Ternary) Operator

The conditional operator assigns a value to a variable based on a condition.

Syntax	Example
variablename = (*condition*) ? *value1*:*value2*	voteable = (age < 18) ? "Too young":"Old enough";

Example explained: If the variable "age" is a value below 18, the value of the variable "voteable" will be "Too young", otherwise the value of voteable will be "Old enough".

6.Logical Operators

Logical operators are used to determine the logic between variables or values. Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x === 5 || y === 5) is false
!	not	!(x === y) is true

Operators in JavaScript

6. JavaScript Bitwise Operators

Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~ 5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2

7. The typeof Operator

The typeof operator returns the type of a variable, object, function or expression:

```
typeof "John"                // Returns string
typeof 3.14                  // Returns number
typeof NaN                   // Returns number
typeof false                 // Returns boolean
typeof [1, 2, 3, 4]          // Returns object
typeof {name:'John', age:34} // Returns object
typeof new Date()            // Returns object
typeof function () {}        // Returns function
typeof myCar                 // Returns undefined (if myCar is not declared)
typeof null                  // Returns object
```

Operators in JavaScript

8. The delete Operator

The delete operator deletes a property from an object:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age;   // or delete person["age"];
```

9. The in Operator

The in operator returns true if the specified property is in the specified object, otherwise false:

```
// Arrays
var cars = ["Saab", "Volvo", "BMW"];
"Saab" in cars          // Returns false (specify the index number instead of value)
0 in cars               // Returns true
1 in cars               // Returns true
4 in cars               // Returns false (does not exist)
"length" in cars        // Returns true  (length is an Array property)

// Objects
var person = {firstName:"John", lastName:"Doe", age:50};
"firstName" in person   // Returns true
"age" in person         // Returns true

// Predefined objects
"PI" in Math            // Returns true
"NaN" in Number         // Returns true
"length" in String      // Returns true
```

Operators in JavaScript

10. The instanceof Operator

The instanceof operator returns true if the specified object is an instance of the specified object:

```
var cars = ["Saab", "Volvo", "BMW"];
cars instanceof Array;          // Returns true
cars instanceof Object;         // Returns true
cars instanceof String;         // Returns false
cars instanceof Number;         // Returns false
```

11. The void Operator

The void operator evaluates an expression and returns undefined. This operator is often used to obtain the undefined primitive value, using "void(0)" (useful when evaluating an expression without using the return value).

```
<a href="javascript:void(0);">
  Useless link
</a>

<a href="javascript:void(document.body.style.backgroundColor='red');">
  Click me to change the background color of body to red
</a>
```

Control structure in JavaScript

Control structure actually controls the flow of execution of a program. Following are the several control structure supported by JavaScript.

- **if … else**
- **switch case**
- **do while loop**
- **while loop**
- **for loop**
- **If … else**

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

```
<script type="text/javascript">
  <!--
     var age = 20;
     if( age > 18 ){
       document.write("<b>Qualifies for driving</b>");
     }
  //-->
</script>
```

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

```
<script type="text/javascript">
   <!--
      var grade='A';
      document.write("Entering switch block<br/>");
      switch (grade) {
         case 'A': document.write("Good job<br/>");
            break;
         case 'B': document.write("Pretty good<br/>");
            break;
         case 'C': document.write("Passed<br/>");
            break;
         case 'D': document.write("Not so good<br/>");
            break;
         case 'F': document.write("Failed<br/>");
            break;
         default:  document.write("Unknown grade<br/>")
      }
      document.write("Exiting switch block");
   //-->
</script>
```

Control structure in JavaScript

Do while Loop

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

```
<script type="text/javascript">
   <!--
      var count = 0;
      document.write("Starting Loop" + "<br/>");
      do{
         document.write("Current Count : " + count + "<br/>");
         count++;
      }while (count < 0);
      document.write("Loop stopped!");
   //-->
</script>
```

While Loop

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

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

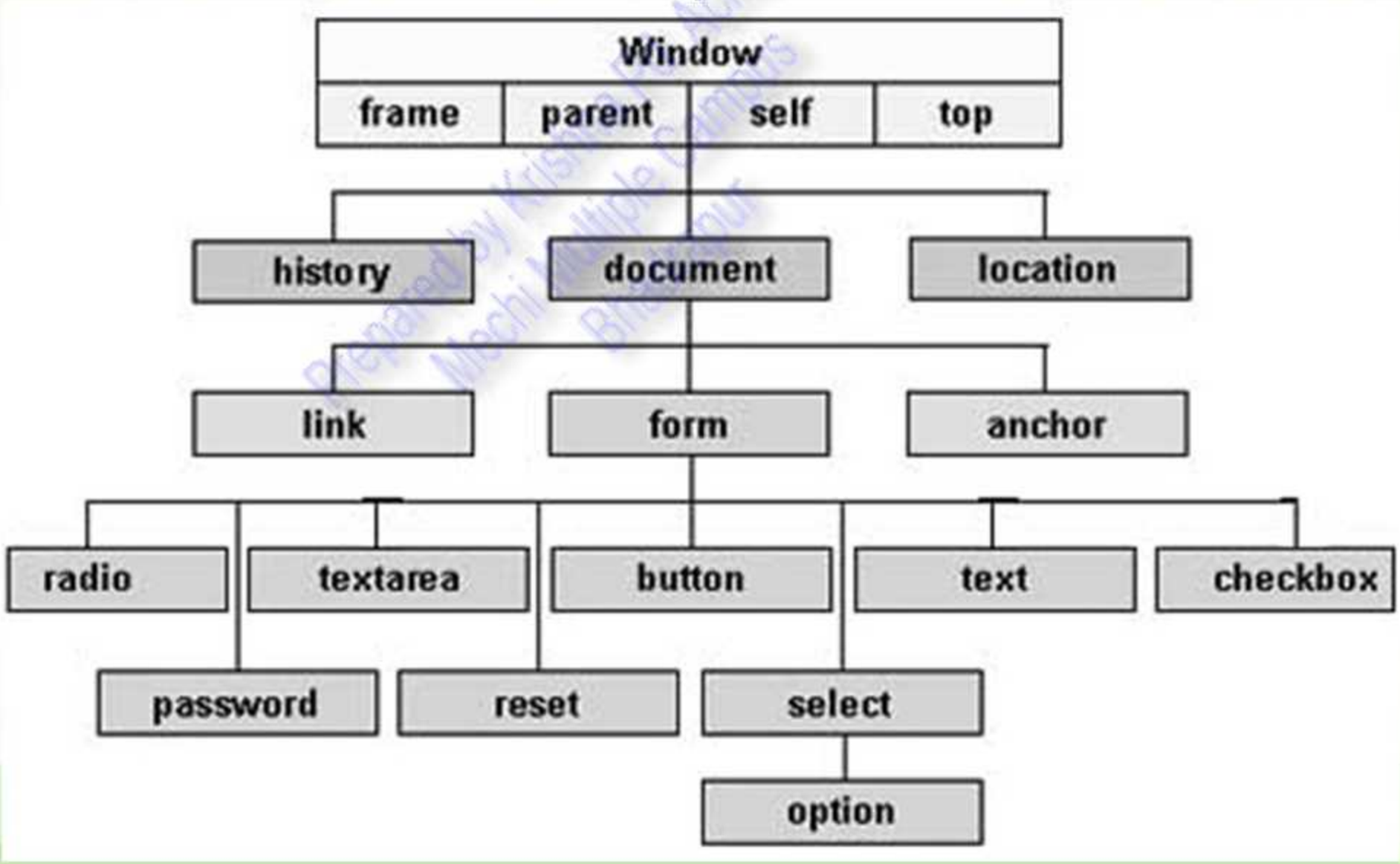
Control structure in JavaScript

For Loop

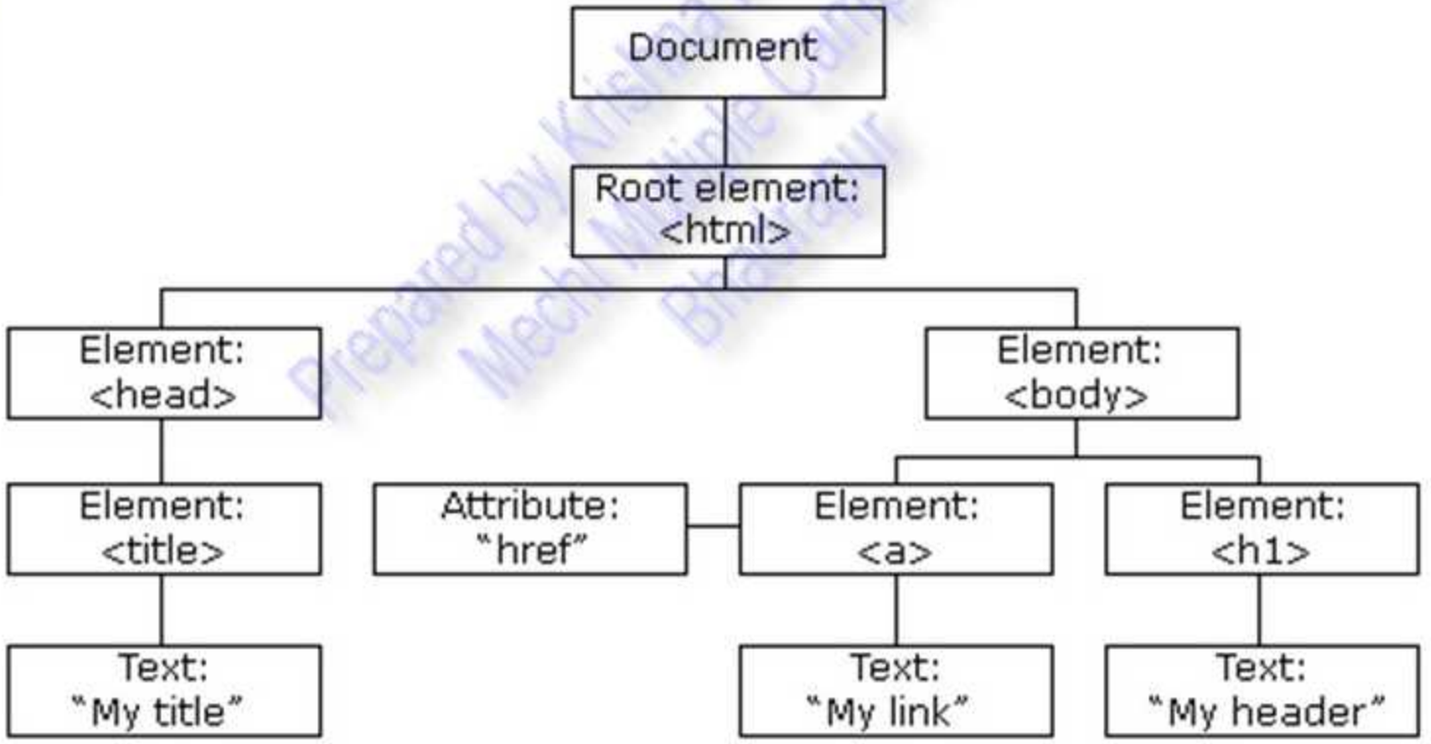
- The for loop is the most compact form of looping and includes the following three important parts –
- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

```
<script type="text/javascript">
   <!--
      var count;
      document.write("Starting Loop" + "<br/>");
      for(count = 0; count < 10; count++){
         document.write("Current Count : " + count );
         document.write("<br/>");
      }
      document.write("Loop stopped!");
   //-->
</script>
```

Document Object Model or DOM



Document Object Model or DOM



Document Object Model or DOM

- Every web page resides inside a browser window which can be considered as an object.
- A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.
- 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.

Document Object Model or DOM

Why DOM is required in JavaScript

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page
- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents:
- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Definition:

- The HTML DOM is a standard object model and programming interface for HTML. It defines:
- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Document Object Model or DOM

```
<html>
  <head>
    <script>
       window.onload = function(){
        var paragraphs = document.getElementsByTagName("P");
        for(i=0;i<=paragraphs.length;i++)
         alert(paragraphs[i].nodeName);
      }
    </script>
  </head>
  <body>
  <p>krishna</p>
  <p>Acharya</p>
  </body>
</html>
```

```
<html>
  <head>
    <script>
      // run this function when the document is loaded
      window.onload = function() {
        // create a couple of elements in an otherwise empty HTML page
        var heading = document.createElement("h1");
        var heading_text = document.createTextNode("Big Head!");
        heading.appendChild(heading_text);
        document.body.appendChild(heading);
      }
    </script>
  </head>
  <body>
```


Why DOM is required in JavaScript

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Document Object Model or DOM

- HTML DOM **methods** are actions you can perform (on HTML Elements).
- HTML DOM **properties** are values (of HTML Elements) that you can set or change.

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
var x = document.getElementsByTagName("p");
</script>
```

Finding HTML Elements

Method	Description
document.getElementById(id)	//Find an element by element id
document.getElementsByTagName(name)	//Find elements by tag name
document.getElementsByClassName(name)	//Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Document Object Model or DOM

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

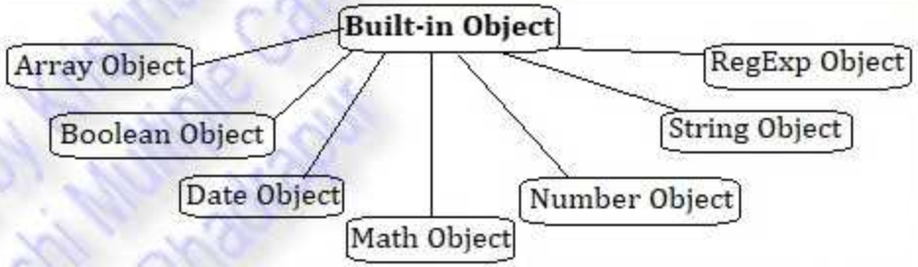
Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

Example

```
<img id="myImage"  src="smiley.gif">
<script>
document.getElementById("myImage").src = "landscape.jpg";
document.getElementById("p2").style.color = "blue";
</script>
document.getElementById("myBtn").addEventListener("click", displayDate);
element.addEventListener("click", function(){ alert("Hello World!"); });
```

JavaScript Built in Objects

JavaScript has several built-in or core language objects. These built-in objects are available regardless of window content and operate independently of whatever page your browser has loaded.



Array Object

- Multiple values are stored in a single variable using the Array object.
- In JavaScript, an array can hold different types of data types in a single slot, which implies that an array can have a string, a number or an object in a single slot.
- An Array object can be created by using following ways:

Using the Array Constructor:

To create empty array when don't know the exact number of elements to be inserted in an array:

var arrayname = new Array(); example var a = new Array()

To create an array of given size

var arrayname = new Array(size); Example var a =new Arrary(10);

JavaScript Built in Objects

To create an array with given elements

var arrayname = new Array("element 1","element 2",……..,"element n");

Example var a=new Arrary("Ram","Shayam","Hari");

Using the Array Literal Notation:

To create empty array

var arrayname =[ ]; example var a=[];

To create an array when elements are given

var arrayname =["element 1","element 2",……..,"element n"];

Example var a=[4,5,6,7,8];

JavaScript Built in Objects

Properties of the Array object

- **Length** - Returns the number of elements in the array. (array.length)
- **Constructor** - Returns the function that created the Array object.
- **Prototype** - Add properties and methods to an object.

Methods of the Array object

- **reverse()** - Reverses the array elements
- **concat()** - Joins two or more arrays
- **sort()** - Sort the elements of an array
- **push()** - Appends one or more elements at the end of an array
- **pop()** - Removes and returns the last element
- **shift()** - Removes and returns the first element

- unshift(), join(), indexOf(), lastIndexOf(), slice(startindex, endindex) are some of the methods used in Array object.

Array Methods

Print all item in array cars

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

Array as object:

var student = {firstName:"Ankit", lastName:"Siwakoti", age:20};

Accessing the property of class:

student.firstName or student["firstName"]

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

fruits.pop(); // Removes the last element ("Mango") from fruits

fruits.push("Kiwi"); // Adds a new element ("Kiwi") to fruits

fruits.shift(); // Removes the first element "Banana" from fruits(begining)

fruits.unshift("Lemon"); // Adds a new element "Lemon" to fruits(begining)

fruits[fruits.length] = "Kiwi"; // Appends "Kiwi" to fruits

delete fruits[0]; // Changes the first element in fruits to **undefined**

fruits.splice(2, 0, "Lemon", "Kiwi"); //how many element add and how many remove

```
<script type="text/javascript">
function Student() {
    this.name = 'John';
    this.gender = 'M';
}
Student.prototype.age = 15;
Student.prototype.getGenderName=function(){
return this.name + "   " + this.gender
}
var studObj = new Student();
alert(studObj.age);
alert(studObj.getGenderName());
</script>
```

Array Methods

```
var myGirls = [“sita", “gita"];

var myBoys = [“mohan", “ram", “hari"];

var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3); //start from 1  and less than 3

var citrus = fruits.slice(2); // 2 onwards

fruits.sort();        // Sorts the elements of fruits

fruits.reverse();     // Then reverse the order of the elements
```

Date Object

- At times when user need to access the current date and time and also past and future date and times. JavaScript provides support for working with dates and time through the Date object.
- The Date object provides a system-independent abstraction of dates and times.
- Date object can be created as : var today = new Date( );
- Dates may be constructed from a year, month, day of the month, hour, minute, and second, and those six components, as well as the day of the week, may be extracted from a date.
- Dates may also be compared and converted to a readable string form. A Date is represented to a precision of one millisecond.

```
var d = new Date();
alert(d);
```

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

Sat Jun 22 2019 10:11:48 GMT+0545 (Nepal Time)

Date Object

- var d = new Date(2018, 11, 24, 10, 33, 30, 0);

alert(a) Mon Dec 24 2018 10:33:30 GMT+0545 (Nepal Time)

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Method	Description
getUTCDate()	Same as getDate(), but returns the UTC date
getUTCDay()	Same as getDay(), but returns the UTC day
getUTCFullYear()	Same as getFullYear(), but returns the UTC year
getUTCHours()	Same as getHours(), but returns the UTC hour
getUTCMilliseconds()	Same as getMilliseconds(), but returns the UTC milliseconds
getUTCMinutes()	Same as getMinutes(), but returns the UTC minutes
getUTCMonth()	Same as getMonth(), but returns the UTC month
getUTCSeconds()	Same as getSeconds(), but returns the UTC seconds

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Math Object

- The Math object is used to perform simple and complex arithmetic operations.
- The Math object provides a number of properties and methods to work with Number values
- The Math object does not have any constructors. All of its methods and properties are static; that is, they are member functions of the Math object itself. There is no way to create an instance of the Math object.

Properties of Math object

PI - The value of Pi

E - The base of natural logarithm e

LN2 - Natural logarithm of 2

LN10 - Natural logarithm of 10

LOG2E - Base 2 logarithm of e

LOG10E - Base 10 logarithm of e

SQRT2 - Square root of 2

SQRT1_2 - Square root of ½

Methods of Math object

max(a,b) - Returns largest of a and b
min(a,b) - Returns least of a and b
round(a) - Returns nearest integer
ceil(a) - Rounds up. Returns the smallest integer greater than or equal to a
floor(a) - Rounds down. Returns the largest integer smaller than or equal to a
exp(a) - Returns ea
pow(a,b) - Returns ab
abs(a) - Returns absolute value of a
random() - Returns a pseudo random number between 0 and 1
sqrt(a) - Returns square root of a
sin(a) - Returns sin of a (a is in radians)
cos(a) - Returns cos of a (a is in radians)

Math Object

```
Math.PI;            // returns 3.141592653589793
Math.pow(8, 2);      // returns 64
Math.round(4.7);    // returns 5
Math.round(4.4);    // returns 4
Math.sqrt(64);      // returns 8
Math.abs(-4.7);     // returns 4.7
Math.floor(4.7);    // returns 4
Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 degrees)
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
Math.max(0, 150, 30, 20, -8, -200);  // returns 150
Math.random();     // returns a random number
```

Regular Expressions

- A **regular expression** is a sequence of characters that forms a search pattern. The search pattern can be used for text search and text replace operations.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replace operations.

Syntax :

/pattern/modifiers;

Examples:

var patt = /mechicampus/i;

Explanation:

/mechicampus/i is a regular expression.

mechicampus is a pattern (to be used in a search).

i is a modifier (modifies the search to be Case-Insensitive).

Note: search(), replace(),match(),test() method can be used to with string.

Regular Expressions

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

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string n

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd

Regular Expressions

- var letters = /^[A-Za-z]+$/;
- var numbers = /^[0-9]+$/;
- var decimal= /^[-+]?[0-9]+\.[0-9]+$/;
- var letters = /^[0-9a-zA-Z]+$/;
- var email=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
- var phoneno = /^\d{10}$/;
- var passw= /^[A-Za-z]\w{7,14}$/;
- ipformat = /^(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)$/;
- var dateformat= /^(0?[1-9] | [12][0-9] | 3[01])[\/\-](0?[1-9] | 1[012])[\/\-]\d{4}$/;

```
<script type="text/javascript">
function myFunction()
{
var numbers = /^[0-9]+$/;
  x=document.getElementById("t1").value;
  if(x.match(numbers))
   alert("Math the pattren");
  else
   alert("Not Math the pattren");
}
```

```
y=document.getElementById("t2").value;
x=new RegExp(email);
if(x.test(y))
alert("yes");
else
alert("no");
```

Loop in JavaScript

1. for - loops through a block of code a number of times
2. for/in - loops through the properties of an object
3. for/of - loops through the values of an iterable object
4. while - loops through a block of code while a specified condition is true
5. do/while - also loops through a block of code while a specified condition is true

```
arr = ['cat', 'dog', 'fish'];
arr.forEach(element => {
 console.log(element);
});
```

```
var a =[5,5,3,5,6];
  for(i=0;i< a.length;i++)
  alert(a[i]);

 for ( value of a)
    alert( value);

a = {id:1,name:2,salary:4};
for ( value in a)
  alert( a[value]);
```

String Object in JavaScript

The JavaScript string is an object that represents a sequence of characters. There are 2 ways to create string in JavaScript.

1. By string literal

   example var stringname="string value";

2. By string object (using new keyword)

var stringname=new String("string literal");

Methods	Description
charAt()	It provides the char value present at the specified index.
charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase()	It converts the given string into lowercase letter.
toLocaleLowerCase()	It converts the given string into lowercase letter on the basis of host?s current locale.
toUpperCase()	It converts the given string into uppercase letter.
toLocaleUpperCase()	It converts the given string into uppercase letter on the basis of host?s current locale.
toString()	It provides a string representing the particular object.
valueOf()	It provides the primitive value of string object.

String Object in JavaScript

1. charAt(x) Returns the character at the "x" position within the string.

```
var myString = 'jQuery FTW!!!';
console.log(myString.charAt(7));   //output F
```

2. charCodeAt(x) Returns the Unicode value of the character at position "x" within the string.

```
//charAt(position)
var message="jquery4u"
alert(message.charAt(1)) //output"q"
```

3. concat(v1, v2,...) Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.

```
//concat(v1, v2,..)
var message="Sam"
var final=message.concat(" is a"," hopeless romantic.")
alert(final) //output "Sam is a hopeless romantic."
```

4. **fromCharCode(**c1, c2,...) Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance. For example: String.fromCharCode().

```
console.log(String.fromCharCode(97,98,99,120,121,122))  //output: abcxyz
console.log(String.fromCharCode(72,69,76,76,79))  //output: HELLO
```

String Object in JavaScript

5. indexOf(substr, [start]) Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is 0.

```
var sentence="Hi, my name is Sam!"
if (sentence.indexOf("Sam")!=-1)
alert("Sam is in there!")
```

6. lastIndexOf(substr, [start]) Searches and (if found) returns the index number of the searched character or substring within the string. Searches the string from end to beginning. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is string.length-1.

7. match(regexp) Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match is found.

```
var intRegex = /^[0-9]+$/;
var myNumber = '999';
var myInt = myNumber.match(intRegex);
alert(myInt);
```

String Object in JavaScript

8. replace(regexp/substr, replacetext) Searches and replaces the regular expression (or sub string) portion (match) with the replaced text instead.

```
var myString = '999 JavaScript Coders';
console.log(myString.replace(/JavaScript/i, "jQuery")); //output: 999 jQuery Coders
var myString = '999 JavaScript Coders';
console.log(myString.replace(new RegExp( "999", "gi" ), "The")); //output: The JavaScript Coders
```

9. search(regexp) Tests for a match in a string. It returns the index of the match, or -1 if not found.

```
var intRegex = /[0-9 -()+]+$/;
var myNumber = '999';
var isInt = myNumber.search(intRegex);  console.log(isInt); //output: 0
```

10. slice(start, [end]) Returns a substring of the string based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.

```
//slice(start, end)
var text="excellent"
text.slice(0,4) //returns "exce"
text.slice(2,4) //returns "ce"
```

String Object in JavaScript

11. split(delimiter, [limit]) Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return.

```
//split(delimiter)
var message="Welcome to jQuery4u"
//word[0] contains "We"
//word[1] contains "lcome to jQuery4u"
var word=message.split("l");
```

12. substr(start, [length]) Returns the characters in a string beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, to)
var text="excellent"
text.substring(0,4) //returns "exce"
text.substring(2,4) //returns "ce"
```

String Object in JavaScript

13. substring(from, [to]) Returns the characters in a string between "from" and "to" indexes, NOT including "to" inself. "To" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, [to])
var myString = 'javascript rox';
myString = myString.substring(0,10);
console.log(myString) //output: javascript
```

14. toLowerCase() Returns the string with all of its characters converted to lowercase.

```
//toLowerCase()
var myString = 'JAVASCRIPT ROX';
myString = myString.toLowerCase();
console.log(myString)   //output: javascript rox
```

15. toUpperCase() Returns the string with all of its characters converted to uppercase.

```
var myString = 'javascript rox';
myString = myString.toUpperCase();
console.log(myString)     //output: JAVASCRIPT ROX
```

Define and invocation function in JavaScript

- JavaScript functions are defined with the function keyword.
- You can use a function declaration or a function expression.

Function Declarations

```
function functionName(parameters) {
  // code to be executed
}
```

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

Define and invocation function in JavaScript

- var x = function (a, b) {return a * b};
- var x = function (a, b) {return a * b};
  var z = x(4, 3);
- Note: After a function expression has been stored in a variable, the variable can be used as a function:
- The function bellow is actually an **anonymous function** (a function without a name).

```
var myFunction = function (a, b) {return a * b};
var x = myFunction(4, 3);
```

anonymous self-invoking function

```
function () {
  var x = "Hello!!";  // I will invoke myself
})();
```

Define and invocation function in JavaScript

```
function myFunction(a, b) {
  return a * b;
}
var x = myFunction(4, 3) * 2;

var txt = myFunction.toString();
```

Arrow Functions

```
// ES5
var x = function(x, y) {
  return x * y;
}
// ES6
const x = (x, y) => x * y;
```

Event in JavaScript

- HTML events are "things" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "react" on these events.
- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked
- An HTML mouse over to html components
- Often, when events happen, you may want to do something.
- JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

Event in JavaScript

Example:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
<button onclick="this.innerHTML = Date()">The time is?</button>
<button onclick="displayDate()">The time is?</button>
```

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Event in JavaScript

Sn	Function	Application
1	onAbort	It occurs when the user stops the loading of an image.
2	onBlur	It occurs when an object on the page is no longer the 'focus'.
3	onChange	It occurs when a text field is changed by the user.
4	onClick	It occurs when the user clicks an object.
5	onError	It occurs when a document or image can't load correctly.
6	onFocus	It occurs when an object takes the 'focus'.
7	onLoad	It occurs when a page is loaded.
8	onMouseOver	It occurs when the mouse cursor moves over an object.
9	onMouseOut	It occurs when the mouse cursor moves off an object.
10	onSelect	It occurs when the user selects text in a text area.
11	onSubmit	It occurs when the user clicks the "submit" button on a form.
12	onUnload	It occurs when the user leaves a document.

Event in JavaScript

```
<img class='imagem_artigo'
        SRC = cei.jpg
        onClick = "alert(Welcome to Computer Educational Institution.')">

<img class='imagem_artigo'
        SRC = cei.jpg
        onDblClick = "alert(Welcome to Computer Educational Institution.')">

<INPUT type = text
         onKeyDown = "alert('Key pressed Welcome to Computer Educational Centre')">

<INPUT type = text
         onKeyDown = "alert('Key pressed Welcome to Computer Educational Centre')">
<INPUT TYPE = text
         onBlur = "alert ('Lost focus from here……..')"> //change the cursor position or lose focus
<form name = form1 onsubmit = “return validate();">
  <input type = submit value = " send ">
</form>
<form name="mypage" action="cei_form.jsp"
onsubmit="return validate_form()" method="post">
```

Form validation

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful. Typical validation tasks are:
has the user filled in all required fields?
has the user entered a valid date?
has the user entered text in a numeric field?

- Most often, the purpose of data validation is to ensure correct user input.
- Validation can be defined by many different methods, and deployed in many different ways.
- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
- JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Server side validation is performed by a web server, after input has been sent to the server.
Client side validation is performed by a web browser, before input is sent to a web server.

Form Validation

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

Example: Number validation

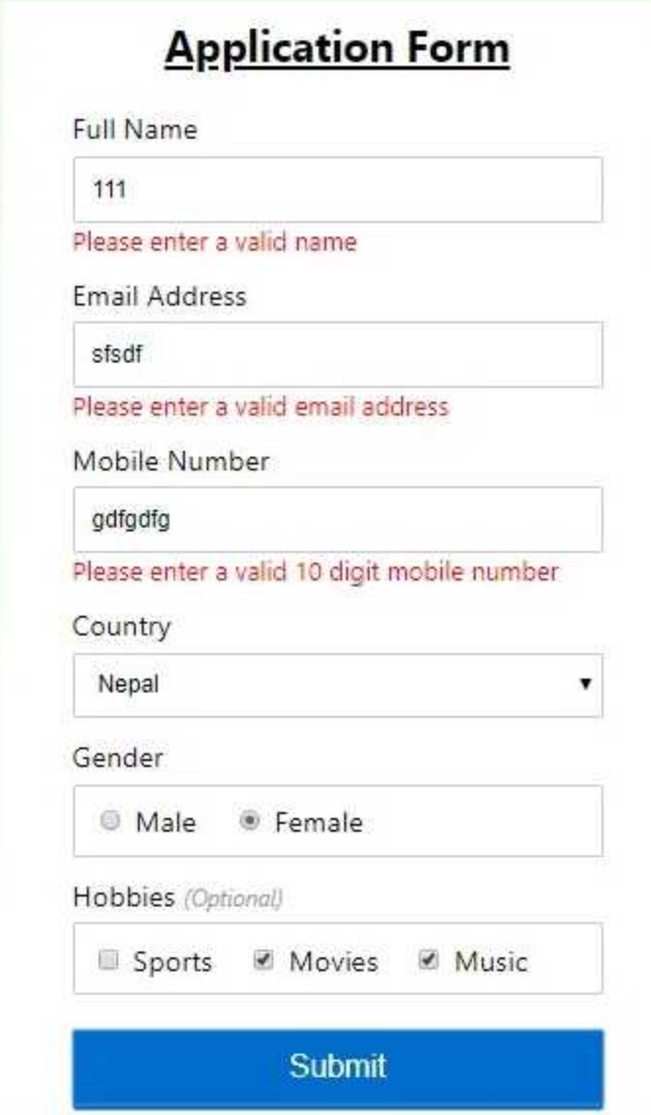
```
x = document.getElementById("numb").value;
if (isNaN(x) || x < 1 || x > 10)
   text = "Input not valid";
 else
   text = "Input OK";
```

Automatic validation

```
<input type="text" name="num" required>
```

Form Validation

```
<!DOCTYPE html>
<html>
<body>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" max="10
<button onclick="myFunction()">OK</bu
<p id="demo"></p>
<script>
function myFunction() {
x=document.getElementById("id1").valu
  var txt = "";
  if (document.getElementById("id1").
    txt = "Value too large";
    else if(x.length==0)
    txt="Empty value";
    else
    txt = "Input OK";
  document.getElementById("demo").inn
}
</script>
</body>
</html>
```



Form Validation

```
<form name="contactForm" onsubmit="return validateForm()"
                    action="/examples/actions/confirmation.php" method="post">
    <h2>Application Form</h2>
    <div class="row">
        <label>Full Name</label>
        <input type="text" name="name">
        <div class="error" id="nameErr"></div>
    </div>
    <div class="row">
        <label>Email Address</label>
        <input type="text" name="email">
        <div class="error" id="emailErr"></div>
    </div>
    <div class="row">
        <label>Mobile Number</label>
        <input type="text" name="mobile" maxlength="10">
        <div class="error" id="mobileErr"></div>
    </div>
```

Form Validation

```
<div class="row">
    <label>Country</label>
    <select name="country">
        <option>Select</option>
        <option>Australia</option>
        <option>Nepal</option>
        <option>United States</option>
        <option>United Kingdom</option>
    </select>
    <div class="error" id="countryErr"></div>
</div>
<div class="row">
    <label>Gender</label>
    <div class="form-inline">
        <label><input type="radio" name="gender" value="male"> Male</label
        <label><input type="radio" name="gender" value="female"> Female</l
    </div>
    <div class="error" id="genderErr"></div>
</div>
```

Form Validation

```
<div class="row">
    <label>Hobbies <i>(Optional)</i></label>
    <div class="form-inline">
        <label><input type="checkbox" name="hobbies" value="sports"> Sport:
        <label><input type="checkbox" name="hobbies" value="movies"> Movie:
        <label><input type="checkbox" name="hobbies" value="music"> Music</
    </div>
</div>
<div class="row">
    <input type="submit" value="Submit">
</div>
```

Form Validation

```
function printError(elemId, hintMsg) {
    document.getElementById(elemId).innerHTML = hintMsg;
}

function validateForm() {
    // Retrieving the values of form elements
    var name = document.contactForm.name.value;
    var email = document.contactForm.email.value;
    var mobile = document.contactForm.mobile.value;
    var country = document.contactForm.country.value;
    var gender = document.contactForm.gender.value;
    var hobbies = [];
    var checkboxes = document.getElementsByName("hobbies");
    for(var i=0; i < checkboxes.length; i++) {
        if(checkboxes[i].checked) {
            // Populate hobbies array with selected values
            hobbies.push(checkboxes[i].value);
        }
    }
```

Form Validation

```
// Defining error variables with a default value
var nameErr = emailErr = mobileErr = countryErr = genderErr = true;

// Validate name
if(name == "") {
    printError("nameErr", "Please enter your name");
} else {
    var regex = /^[a-zA-Z\s]+$/;
    if(regex.test(name) === false) {
        printError("nameErr", "Please enter a valid name");
    } else {
        printError("nameErr", "");
        nameErr = false;
    }
}
```

Form Validation

```
  // Defining error variables with a default value
// Validate email address
if(email == "") {
    printError("emailErr", "Please enter your email address");
} else {
    // Regular expression for basic email validation
    var regex = /^\S+@\S+\.\S+$/;
    if(regex.test(email) === false) {
        printError("emailErr", "Please enter a valid email address");
    } else{
        printError("emailErr", "");
        emailErr = false;
    }
}
```

Form Validation

```
// Validate mobile number
if(mobile == "") {
    printError("mobileErr", "Please enter your mobile number");
} else {
    var regex = /^[1-9]\d{9}$/;
    if(regex.test(mobile) === false) {
        printError("mobileErr", "Please enter a valid 10 digit mobile num
    } else{
        printError("mobileErr", "");
        mobileErr = false;
    }
}
```

```
// Validate country
if(country == "Select") {
    printError("countryErr", "Please select your country");
} else {
    printError("countryErr", "");
    countryErr = false;
}
```

Form Validation

```
// Validate gender
if(gender == "") {
    printError("genderErr", "Please select your gender");
} else {
    printError("genderErr", "");
    genderErr = false;
}
```

```
// Prevent the form from being submitted if there are any errors
if((nameErr || emailErr || mobileErr || countryErr || genderErr) == true)
   return false;
} else {
    // Creating a string from input data for preview
    var dataPreview = "You've entered the following details: \n" +
                      "Full Name: " + name + "\n" +
                      "Email Address: " + email + "\n" +
                      "Mobile Number: " + mobile + "\n" +
                      "Country: " + country + "\n" +
                      "Gender: " + gender + "\n";
    if(hobbies.length) {
        dataPreview += "Hobbies: " + hobbies.join(", ");
    }
    // Display input data in a dialog box before submitting the form
    alert(dataPreview);
}
```

Cookies

What are Cookies?

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

How to create cookie :

document.cookie = "username=mohan";

we 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";

How to read a cookie:

var x = document.cookie; //x is array

Delete a Cookie with JavaScript

Deleting a cookie is very simple.

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

Just set the expires parameter to a passed date:

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

Cookies

Cookie Pros

Convenience: Cookies not only remember which websites you have been to, they also remember information about forms.

Personalization: cookie ca be used for serving up personalized content that is geared towards that specific user's preferences. Amazon uses cookies to offer you related products, Google uses cookies to better understand your searches, and Facebook uses cookies to do just about everything.

Effective Advertising: How nice would it be to only be offered products or services that are relevant to you? Internet marketing companies collect data from cookies to run marketing campaigns aimed at a very specific market segment including product group, geolocation, search term, and demographics.

Ease of Control: It is actually really easy to manage your cookies if you know how. Most browsers make it easy for you to clear your browsing history. Just go to tools, clear history and select cookies. Cookies are stored on your hard drive in a text file under cookie.txt, and since it is a text file you can use just about any viewer or text editor to display, edit, and delete them.

Cookie Cons

Privacy: The main concern for most users is privacy. Cookie enabled web browsers keep track of all the websites you have visited.

Security: Cookie security is a large problem. The concern is that many security holes have been found in different browsers. Some of these holes were so serious that they allowed malicious webmasters to gain access to users' email, different passwords, and credit card information.

Secrecy: Although third party cookies can be blocked through your browser settings, most people don't have the technical expertise to do this. Most browsers purposely make it difficult to find this setting in order to prevent you from turning them off. No cookies mean no data, which in turn means less money.

Cookies

```
<!DOCTYPE html>
<html>
<head>
    <title>Cookie</title>
<script type="text/javascript">
    window.onload = function ()
    {
        if (document.cookie.length != 0) {
            var nameValueArray = document.cookie.split("=");
            document.getElementById("bcolor").value = nameValueArray[1];
            document.bgColor = nameValueArray[1];
        }
    }
    function setColorCookie()
    {
        var selectedValue = document.getElementById("bcolor").value;
        if (selectedValue != "Select Color")
        {
            document.bgColor = selectedValue;
```

Cookies

```
        if (selectedValue != "Select Color")
        {
            document.bgColor = selectedValue;
            document.cookie="color="+selectedValue+";expires=Fri,
            1 July 2019 01:00:00 UTC;";
        }
    }
</script>
</head>
<body>
    <select id="bcolor" onchange="setColorCookie();">
    <option value="Select Color">Select Color</option>
    <option value="red">Red</option>
    <option value="green">Green</option>
    <option value="blue">Blue</option>
</select>
</body>
</html>
```

setInterval function

- The setInterval() method calls a function or evaluates an expression at specified intervals (in milliseconds).
- The setInterval() method will continue calling the function until clearInterval() is called, or the window is closed.
- The ID value returned by setInterval() is used as the parameter for the clearInterval() method.
- Tip: 1000 ms = 1 second.
- Tip: To execute a function only once, after a specified number of milliseconds, use the setTimeout() method.


```
<!DOCTYPE html>
<html>
<body>
<p>A script on this page starts this clock:</p>
<p id="demo"></p>
<button onclick="myStopFunction()">Stop time</button>
<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
  var d = new Date();
  var t = d.toLocaleTimeString();
  document.getElementById("demo").innerHTML = t;
}
function myStopFunction() {
  clearInterval(myVar);
}
</script>
</body>
</html>
```

A script on this page starts this clock:

1:01:46 PM

Stop time

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

<p>In this example, the setInterval() method executes
300 milliseconds, which will toggle between two backgr

<button onclick="stopColor()">Stop Toggling</button>

<script>
var i=0;
color=["red","blue","green","yellow"];
var myVar = setInterval(setColor, 300);

function setColor() {
  var x = document.body.bgColor=color[i];
  if(i==3)
    i=0;
  i++;
}

function stopColor() {
  clearInterval(myVar);
}
</script>

</body>
</html>
```

setTimeout function

- The setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.
- Tip: 1000 ms = 1 second.
- Tip: The function is only executed once. If you need to repeat execution, use the setInterval() method.
- Tip: Use the clearTimeout() method to prevent the function from running.

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

<p>Click the button to wait 3 seconds, then alert "Hello".</p>

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

<script>
var myVar;

function myFunction() {
  myVar = setTimeout(alertFunc, 3000);
}

function alertFunc() {
  alert("Hello!");
}
</script>

</body>
</html>
```

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

<p>Click the button to open a new window and close the window after three seconds (3000
milliseconds)</p>

<button onclick="openWin()">Open "myWindow"</button>

<script>
function openWin() {
  var myWindow = window.open("", "myWindow", "width=200, height=100");
  myWindow.document.write("<p>This is 'myWindow'</p>");
  setTimeout(function(){ myWindow.close() }, 3000);
}

</script>

</body>
</html>
```

Dialog Boxes in JavaScript

- The dialog box is a graphical control element in the form of a small window that communicates information to the user and prompts them for a response. Dialog boxes are classified as "modal" or "modeless", depending on whether they block interaction with the software that initiated the dialog.
- A **modal dialog** temporarily locks you out of the rest of an application until it is closed. While a modal dialog box is open, it will always be the top window of the application — no other windows will be in front of it.
- A **modeless dialog** allows you to use the rest of the application while it is open. It can be hidden by other application windows.

1. **Alert Dialog Box:** An alert dialog box is the most simple dialog box. It enables you to display a short message to the user. It also includes OK button, and the user has to click this OK button to continue.

```
var message = "Hi there! Click OK to continue.";
alert(message);
/* The following line won't execute until you dismiss previous alert */
alert("This is another alert box.");
```

2. confirm dialog box allows user to confirm or cancel an action. A confirm dialog looks similar to an alert dialog but it includes a Cancel button along with the OK button.

```
var result = confirm("Are you sure?");
if(result)
 document.write("You clicked OK button!");
else
 document.write("You clicked Cancel button!");
```

Dialog Boxes in JavaScript

3. The prompt dialog box is used to prompt the user to enter information. A prompt dialog box includes a text input field, an OK and a Cancel button.

```
var name = prompt("What's your name?");
if(name.length > 0 && name != "null") {
    document.write("Hi, " + name);
} else {
    document.write("Anonymous!");
}
```

```
var age = Number(prompt("What's your age?"));  //type casting
```