2.1 JAVA SCRIPT: AN INTRODUCTION TO JAVASCRIPT

Here to begin with a “Hello World!” JavaScript program. There’s just one problem:JavaScript itself has no statement for performing output. Instead, the JavaScript language specification leaves it up to browsers to supply output (and input) methods.

- alert(), to write a JavaScript “Hello World!” program

window.alert("Hello World!");

- We can execute this program by referencing a file containing it from a script element within an HTML document.
- For example, if the JavaScript code given is placed within a file named JSHelloWorld.js in the same directory as the HTML document of Fig 4.1, then loading this document into a browser will cause the code to be executed.
- If this document is loaded into Mozilla 1.4, then the pop-up window shown in Figure 4.2 will appear.
- The browser window will be unresponsive until this *alert box* is dismissed, either by clicking the OK button or by closing the window .
- The client area of the browser window itself will be completely empty after the alert box is dismissed.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>
JSHelloWorld.html
</title>
<script type="text/javascript" src="JSHelloWorld.js">
</script>
</head>
<body>
</body>
</html>
```

Fig 2.1 HTML document that loads and executes the JavaScript program in file JSHelloWorld.js.

Fig 2.2 Alert box generated by a JavaScript statement.

- As with output, there is no input statement in JavaScript itself. Again, though, most browsers implement a prompt() method that can be called as illustrated by the following code:

var inString = window.prompt("Enter JavaScript code to be tested:","");

- This pops up a window that displays the value of its first string argument and also provides a text box in which a user can enter information and that initially contains the string given by the second argument .
- The value returned by the prompt() method is the string entered by the user, assuming that the user clicks OK after entering the user, assuming that the user clicks OK after entering the string.

2.2 JAVASCRIPT DOM MODEL

- The **document object** represents the whole html document.
- When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.
- As mentioned earlier, it is the object of window. So

window.document,is same as **document**

- According to W3C - "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."

2.2.1 Properties of document object

The properties of document object that can be accessed and modified by the document object.

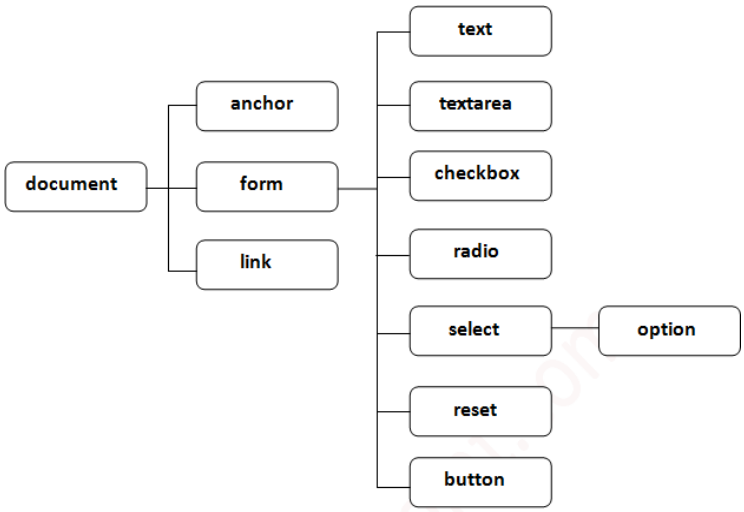

Fig 2.3 Properties of Document Object

Methods of document object

We can access and change the contents of document by its methods.The important methods of document object are as follows:

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

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

- Here, **document** is the root element that represents the html document.
- **form1** is the name of the form.
- **name** is the attribute name of the input text.
- **value** is the property, that returns the value of the input text.

Simple example of document object that prints name with welcome message.

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

2.2.2 Javascript - document.getElementById() method

The **document.getElementById()** method returns the element of specified id.

- In the previous topic, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use **document.getElementById()** method to get value of the input text. But we need to define id for the input field.

Simple example of document.getElementById() method that prints cube of the given number.

```
<script type="text/javascript">
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
```

```
}
</script>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
```

2.2.3 Javascript - document.getElementsByName() method

1. getElementsByName() method
2. Example of getElementsByName()

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

1. document.getElementsByName("name")

Here, name is required.

Example of document.getElementsByName() method

In this example, count total number of genders. Here, we are using getElementsByName() method to get all the genders.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">
  <input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

2.2.4 Javascript - document.getElementsByTagName() method

- getElementsByTagName() method

Example of getElementsByTagName()

- The document.getElementsByTagName() method returns all the element of specified tag name.
- The syntax of the getElementsByTagName() method is given below:

document.getElementsByTagName("name")

Here, name is required.

Example of document.getElementsByTagName() method

- In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
<p>This is a pragraph</p>
<p>Here we are going to count total number of paragraphs by
getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

- Here, we are going to count total number of paragraphs by getElementByTagName() method.
- Another example of document.getElementsByTagName() method.In this example, we going to count total number of h2 and h3 tags used in the document.

```
<script type="text/javascript">
function counth2(){
var totalh2=document.getElementsByTagName("h2");
alert("total h2 tags are: "+totalh2.length);
}
function counth3(){
var totalh3=document.getElementsByTagName("h3");
alert("total h3 tags are: "+totalh3.length);
}
</script>
<h2>This is h2 tag</h2>
<h2>This is h2 tag</h2>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<button onclick="counth2()">count h2</button>
<button onclick="counth3()">count h3</button>
```

Output of the above example

This is h2 tag
This is h2 tag
This is h3 tag
This is h3 tag
This is h3 tag

Note: Output of the given examples may differ on this page because it will count the total number of para , total number of h2 and total number of h3 tags used in this document.

2.2.4 Javascript - innerHTML

1. javascript innerHTML
2. Example of innerHTML property

- The **innerHTML** property can be used to write the dynamic html on the html document.
- It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example of innerHTML property

- In this example, we are going to create the html form when user clicks on the button.
- In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
<script type="text/javascript" >
function showcommentform() {
var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows=
'5' cols='80'></textarea>
<br><input type='submit' value='Post Comment'>";
document.getElementById('mylocation').innerHTML=data;
}
</script>
<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>
```

Show/Hide Comment Form Example using innerHTML

```
<!DOCTYPE html>
<html>
<head>
<title>First JS</title>
<script>
var flag=true;
function commentform(){
var cform="<form action='Comment'>Enter Name:<br><input type='text' name='nam
e'/><br/>
Enter Email:<br><input type='email' name='email'/><br>Enter Comment:<br/>
<textarea rows='5' cols='70'></textarea><br><input type='submit' value='Post Comm
ent'/></form>";
if(flag){
document.getElementById("mylocation").innerHTML=cform;
flag=false;
}else{
document.getElementById("mylocation").innerHTML="";
flag=true;
}
}
```

```
</script>
</head>
<body>
<button onclick="commentform()">Comment</button>
<div id="mylocation"></div>
</body>
</html>
```

2.2.5 Javascript - innerText

1. javascript innerText
2. Example of innerText property

- The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.
- It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

```
<script type="text/javascript" >
function validate() {
var msg;
if(document.myForm.userPass.value.length>5){
msg="good";
}
else{
msg="poor";
}
document.getElementById('mylocation').innerText=msg;
 }

</script>
<form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="mylocation">no strength</span>
</form>
```

2.3 DATE AND OBJECTS

2.3.1 JavaScript Date

- The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.
- You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

You can use 4 variant of Date constructor to create date object.

- Date()
- Date(milliseconds)
- Date(dateString)
- Date(year, month, day, hours, minutes, seconds, milliseconds)

JavaScript Date Methods

Methods	Description
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
getFullYears()	It returns the integer value that represents the year on the basis of local time.
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
getMilliseconds()	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
getSeconds()	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.

JavaScript Date Example

```
Current Date and Time: <span id="txt"></span>
<script>
var today=new Date();
document.getElementById('txt').innerHTML=today;
</script>
```

JavaScript Current Time Example

The simple example to print current time of system.

```
Current Time: <span id="txt"></span>
<script>
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
</script>
```

JavaScript Digital Clock Example

- The simple example to display digital clock using JavaScript date object.
- There are two ways to set interval in JavaScript: by setTimeout() or setInterval() method.

```
Current Time: <span id="txt"></span>
<script>
window.onload=function(){getTime();}
function getTime(){
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
// add a zero in front of numbers<10
m=checkTime(m);
s=checkTime(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
setTimeout(function(){getTime()},1000);
}
//setInterval("getTime()",1000);//another way
function checkTime(i){
if (i<10){
  i="0" + i;
 }
return i;
}
</script>
```

2.3.2 JavaScript Object

- A javaScript object is an entity having state and behavior (properties and method).For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects

Creating Objects in JavaScript

There are 3 ways to create objects.

- By object literal
- By creating instance of Object directly (using new keyword)
- By using an object constructor (using new keyword)

JavaScript Object by object literal

The syntax of creating object using object literal is given below:

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

As you can see, property and value is separated by : (colon).Let's see the simple example of creating object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();

- Here, new keyword is used to create object.Let's see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.The **this keyword** refers to the current object.The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);
  document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
```

```
}
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

JavaScript Object Methods

The various methods of Object are as follows:

S.No	Methods	Description
1	Object.assign()	This method is used to copy enumerable and own properties from a source object to a target object
2	Object.create()	This method is used to create a new object with the specified prototype object and properties.
3	Object.defineProperty()	This method is used to describe some behavioral attributes of the property.
4	Object.defineProperties()	This method is used to create or configure multiple object properties.
5	Object.entries()	This method returns an array with arrays of the key, value pairs.
6	Object.freeze()	This method prevents existing properties from being removed.
7	Object.getOwnPropertyDescriptor()	This method returns a property descriptor for the specified property of the specified object.
8	Object.getOwnPropertyDescriptors()	This method returns all own property descriptors of a given object.
9	Object.getOwnPropertyNames()	This method returns an array of all properties (enumerable or not) found.

2.3.3 JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements. There are 3 ways to construct array in JavaScript

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

JavaScript array literal

The syntax of creating array using array literal is given below:

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

values are contained inside [ ] and separated by , (comma). The simple example of creating and using array in JavaScript.

```
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
```

JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

var arrayname=new Array();

Here, new keyword is used to create instance of array.The example of creating array directly.

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JavaScript array constructor (new keyword)

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

The example of creating object by array constructor is given below.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JavaScript Array Methods

The list of JavaScript array methods with their description.

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.
copywithin()	It copies the part of the given array with its own elements and returns the modified array.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
fill()	It fills elements into an array with static values.
filter()	It returns the new array containing the elements that pass the provided function conditions.
find()	It returns the value of the first element in the given array that satisfies the specified condition.
findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.

indexOf()	It searches the specified element in the given array and returns the index of the first match.
join()	It joins the elements of an array as a string.

2.3.4 JavaScript String

- The **JavaScript string** is an object that represents a sequence of characters. There are 2 ways to create string in JavaScript
  - By string literal
  - By string object (using new keyword)

By string literal

- The string literal is created using double quotes. The syntax of creating string using string literal is given below:

var stringname="string value";

Example of creating string literal.

```
<script>

var str="This is string literal";

document.write(str);

</script>
```

By string object (using new keyword)

The syntax of creating string object using new keyword is given below:

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

Here, new keyword is used to create instance of string.

Example of creating string in JavaScript by new keyword.

```
<script>

var stringname=new String("hello javascript string");

document.write(stringname);

</script>
```

JavaScript String Methods

The list of JavaScript string methods with examples.

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.

JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

```
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
```

JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

```
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1.concat(s2);
document.write(s3);
</script>
```

JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
```

JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
```

JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

```
<script>
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
```

JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

```
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
```

JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<script>
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write(s2);
```

```
</script>
```

JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<script>
var s1="     javascript trim     ";
var s2=s1.trim();
document.write(s2);
</script>
```

2.3.5 JavaScript Math

The **JavaScript math** object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

JavaScript Math Methods

Let's see the list of JavaScript Math methods with description.

Methods	Description
abs()	It returns the absolute value of the given number.
acos()	It returns the arccosine of the given number in radians.
asin()	It returns the arcsine of the given number in radians.
atan()	It returns the arc-tangent of the given number in radians.
cbrt()	It returns the cube root of the given number.
ceil()	It returns a smallest integer value, greater than or equal to the given number.
cos()	It returns the cosine of the given number.
cosh()	It returns the hyperbolic cosine of the given number.
exp()	It returns the exponential form of the given number.

Math.sqrt(n)

The JavaScript math.sqrt(n) method returns the square root of the given number.

```
Square Root of 17 is: <span id="p1"></span>
<script>
document.getElementById('p1').innerHTML=Math.sqrt(17);
```

</script>

Math.random()

The JavaScript math.random() method returns the random number between 0 to 1.

Random Number is:

<script>

document.getElementById('p2').innerHTML=Math.random();

</script>

Math.pow(m,n)

The JavaScript math.pow(m,n) method returns the m to the power of n that is m^n.

3 to the power of 4 is:

<script>

document.getElementById('p3').innerHTML=Math.pow(3,4);

</script>

2.3.6 JavaScript Number Object

- The **JavaScript number** object *enables you to represent a numeric value*. It may be integer or floating-point.
- JavaScript number object follows IEEE standard to represent the floating-point numbers.
- By the help of Number() constructor, you can create number object in JavaScript. For example:

var n=new Number(value);

For example:

var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object

JavaScript Number Constants

Constant	Description
MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.
POSITIVE_INFINITY	returns positive infinity, overflow value.

NEGATIVE_INFINITY	returns negative infinity, overflow value.
NaN	represents "Not a Number" value.

JavaScript Number Methods

Methods	Description
isFinite()	It determines whether the given value is a finite number.
isInteger()	It determines whether the given value is an integer.
parseFloat()	It converts the given string into a floating point number.
parseInt()	It converts the given string into an integer number.
toExponential()	It returns the string that represents exponential notation of the given number.
toFixed()	It returns the string that represents a number with exact digits after a decimal point.
toPrecision()	It returns the string representing a number of specified precision.
toString()	It returns the given number in the form of string.

2.3.7 JavaScript Boolean

- **JavaScript Boolean** is an object that represents value in two states: *true* or *false*. You can create the JavaScript Boolean object by Boolean() constructor as given below. The default value of JavaScript Boolean object is *false*.

Boolean b=new Boolean(value);

JavaScript Boolean Example

```
<script>
document.write(10<20);//true
document.write(10<5);//false
</script>
```

JavaScript Boolean Properties

Property	Description
constructor	returns the reference of Boolean function that created Boolean object.
prototype	enables you to add properties and methods in Boolean prototype.

JavaScript Boolean Methods

Method	Description
toSource()	returns the source of Boolean object as a string.
toString()	converts Boolean into String.
valueOf()	converts other type into Boolean.

2.4 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.

What Is a Regular Expression?

- A regular expression is a sequence of characters that forms a **search pattern**.
- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- 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*;

Example

```
var patt = /w3schools/i;
```

Example explained:

/w3schools/i is a regular expression.

w3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

2.4.1 Using String Methods

- In JavaScript, regular expressions are often used with the two string methods: search() and replace().
- The search() method uses an expression to search for a match, and returns the position of the match.
- The replace() method returns a modified string where the pattern is replaced.
- Using String search() With a String
- The search() method searches a string for a specified value and returns the position of the match:

Example

Use a string to do a search for "W3schools" in a string:

```
var str = "Visit W3Schools!";
var n = str.search("W3Schools");
```

2.4.2 Using String Methods

- In JavaScript, regular expressions are often used with the two string methods: search() and replace().
- The search() method uses an expression to search for a match, and returns the position of the match.
- The replace() method returns a modified string where the pattern is replaced.
- Using String search() With a String
- The search() method searches a string for a specified value and returns the position of the match:

Example

Use a string to do a search for "W3schools" in a string:

```
var str = "Visit W3Schools!";

var n = str.search("W3Schools");
```

2.4.3 Using String replace() With a String

The replace() method replaces a specified value with another value in a string:

```
var str = "Visit Microsoft!";
var res = str.replace("Microsoft", "W3Schools");
```

2.4.4 Use String replace() With a Regular Expression

Example

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
```

2.4.5 Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
i	Perform Case-insensitive Matching
g	Perform a Global Matching
m	Perform Multiline Matching

2.4.6 Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	Description t
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with

- Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal

- Quantifiers define quantities:

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 nnumber xxxx

2.4.7 Using the RegExp Object

In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

Using test()

- The test() method is a RegExp expression method.
- It searches a string for a pattern, and returns true or false, depending on the result.
- The following example searches a string for the character "e":

Using exec()

- The exec() method is a RegExp expression method.
- It searches a string for a specified pattern, and returns the found text as an object.
- If no match is found, it returns an empty *(null)* object.
- The following example searches a string for the character "e":

2.5 EXCEPTION HANDLING

There are three types of errors in programming:

- Syntax Errors
- Runtime Errors, and
- Logical Errors.

Syntax Errors

- Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript.
- For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">
  <!--
    window.print(;
  //-->
</script>
```

- When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

- Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).
- For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
  <!--
    window.printme();
  //-->
</script>
```

- Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

- Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.
- You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

2.5.1 The try...catch...finally Statement

- The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.
- You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.Here is the try...catch...finally block syntax −

```
<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>
```

- The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed ine and the catch block is executed.
- The optional finally block executes unconditionally after try/catch.

Examples

- Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without try...catch−

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               alert("Value of variable a is : " + a );
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>
   </body>
</html>
```

- Now let us try to catch this exception using try...catch and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               try {
                  alert("Value of variable a is : " + a );
               }
               catch ( e ) {
                  alert("Error: " + e.description );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

- You can use **finally** block which will always execute unconditionally after the try/catch. Here is an example.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;

               try {
                  alert("Value of variable a is : " + a );
               }
               catch ( e ) {
                  alert("Error: " + e.description );
               }
               finally {
                  alert("Finally block will always execute!" );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
```

```
      </form>

   </body>
</html>
```

2.5.2 The throw Statement

- You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example:

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a  =  100;
               var b = 0;

               try {
                  if ( b == 0 ) {
                     throw( "Divide by zero error." );
                  } else {
                     var c = a / b;
                  }
               }
               catch ( e ) {
                  alert("Error: " + e );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

- You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a **try...catch** block.

2.5.3 The onerror() Method

- The onerror event handler was the first feature to facilitate error handling in JavaScript. The error event is fired on the window object whenever an exception occurs on the page.

Live Demo

```
<html>
```

```
   <head>

      <script type = "text/javascript">
         <!--
            window.onerror = function () {
               alert("An error occurred.");
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

- The **onerror** event handler provides three pieces of information to identify the exact nature of the error –
  - **Error message** − The same message that the browser would display for the given error
  - **URL** − The file in which the error occurred
  - **Line number**− The line number in the given URL that caused the error.

2.6 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.
- If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information.
- This was really a lengthy process which used to put a lot of burden on the server.
- 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.
- 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.
- 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.

Example

```
   <head>
      <title>Form Validation</title>
      <script type = "text/javascript">
         <!--
            // Form validation code will come here.
         //-->
      </script>
   </head>

   <body>
```

```
      <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit =
"return(validate());">
         <table cellspacing = "2" cellpadding = "2" border = "1">

            <tr>
               <td align = "right">Name</td>
               <td><input type = "text" name = "Name" /></td>
            </tr>

            <tr>
               <td align = "right">EMail</td>
               <td><input type = "text" name = "EMail" /></td>
            </tr>

            <tr>
               <td align = "right">Zip Code</td>
               <td><input type = "text" name = "Zip" /></td>
            </tr>

            <tr>
               <td align = "right">Country</td>
               <td>
                  <select name = "Country">
                     <option value = "-1" selected>[choose yours]</option>
                     <option value = "1">USA</option>
                     <option value = "2">UK</option>
                     <option value = "3">INDIA</option>
                  </select>
               </td>
            </tr>

            <tr>
               <td align = "right"></td>
               <td><input type = "submit" value = "Submit" /></td>
            </tr>

         </table>
      </form>
   </body>
</html>
```

2.6.1 Basic Form Validation

- First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
   <!--
      // Form validation code will come here.
      function validate() {

         if( document.myForm.Name.value == "" ) {
            alert( "Please provide your name!" );
            document.myForm.Name.focus()  ;
            return false;
         }
         if( document.myForm.EMail.value == "" ) {
            alert( "Please provide your Email!" );
            document.myForm.EMail.focus()  ;
            return false;
```

```
         }
         if( document.myForm.Zip.value == "" || isNaN(
document.myForm.Zip.value ) ||
            document.myForm.Zip.value.length != 5 ) {

            alert( "Please provide a zip in the format #####." );
            document.myForm.Zip.focus() ;
            return false;
         }
         if( document.myForm.Country.value == "-1" ) {
            alert( "Please provide your country!" );
            return false;
         }
         return( true );
      }
   //-->
</script>
```

2.6.2 Data Format Validation

- Now we will see how we can validate our entered form data before submitting it to the web server.
- The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.).
- Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

```
<script type = "text/javascript">
   <!--
      function validateEmail() {
         var emailID = document.myForm.EMail.value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");

         if (atpos < 1 || ( dotpos - atpos < 2 )) {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
         }
         return( true );
      }
   //-->
</script>
```

2.7 BUILT IN OBJECTS

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

2.7.1 JavaScript Primitives

- A primitive value is a value that has no properties or methods.
- A primitive data type is data that has a primitive value.
- JavaScript defines 5 types of primitive data types:
  - String
  - Number
  - Boolean
  - Null
  - undefined
- Primitive values are immutable (they are hardcoded and therefore cannot be changed).
- if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

2.7.2 Objects are Variables

JavaScript variables can contain single values:

Example

```
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).

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- A JavaScript object is a collection of **named values**

2.7.3 Object Properties

- The named values, in JavaScript objects, are called properties.

Property	Value
firstName	John
lastName	Doe

age	50
eyeColor	blue

- Objects written as name value pairs are similar to:
  - Associative arrays in PHP
  - Dictionaries in Python
  - Hash tables in C
  - Hash maps in Java
  - Hashes in Ruby and Perl

2.7.4 Object Methods

- Methods are actions that can be performed on objects.
- Object properties can be both primitive values, other objects, and functions.
- An object method is an object property containing a function definition.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

- JavaScript objects are containers for named values, called properties and methods.

2.7.5 Creating a JavaScript Object

- With JavaScript, you can define and create your own objects.
- There are different ways to create new objects:
- Define and create a single object, using an object literal.
- Define and create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.
- In ECMAScript 5, an object can also be created with the function Object.create().

Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs (like age:50) inside curly braces {}.
- The following example creates a new JavaScript object with four properties:

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Using the JavaScript Keyword new

- The following example also creates a new JavaScript object with four properties:

Example

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

JavaScript Objects are Mutable

- Objects are mutable: They are addressed by reference, not by value.
- If person is an object, the following statement will not create a copy of person:
  var x = person; // This will not create a copy of person.
- The object x is **not a copy** of person. It **is** person. Both x and person are the same object.
- Any changes to x will also change person, because x and person are the same object.

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}
varx=person;
x.age = 10;          // This will change both x.age and person.age
```

2.7.6 JavaScript Object Properties

- Accessing JavaScript Properties
- The syntax for accessing the property of an object is:

```
objectName.property        // person.age
                                                or
objectName["property"]     // person["age"]
                                                or
objectName[expression]     // x = "age"; person[x]
```

2.7.7 JavaScript for...in Loop

- The JavaScript for...in statement loops through the properties of an object.

Syntax

```
for (variable in object) {
  // code to be executed
}
```

- The block of code inside of the for...in loop will be executed once for each property.
- Looping through the properties of an object:

2.7.8 Object Methods

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The this Keyword

- In a function definition, this refers to the "owner" of the function.
- In the example above, this is the person object that "owns" the fullName function.
- In other words, this.firstName means the firstName property of this object.

JavaScript Methods

- JavaScript methods are actions that can be performed on objects.
- A JavaScript method is a property containing a function definition.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Accessing Object Methods

- You access an object method with the following syntax:

objectName.methodName()

- You will typically describe fullName() as a method of the person object, and fullName as a property.
- The fullName property will execute (as a function) when it is invoked with ().
- This example accesses the fullName() **method** of a person object:

Example

```
name = person.fullName();
```

Using Built-In Methods

- This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message = "Hello world!";
var x = message.toUpperCase();
```

The value of x, after execution of the code above will be:

HELLO WORLD!

Adding a Method to an Object

Adding a new method to an object is easy:

Example

```
person.name = function () {
  return this.firstName + " " + this.lastName;
};
```

2.7.9 Object Accessors

JavaScript Accessors (Getters and Setters)

- ECMAScript 5 (2009) introduced Getter and Setters.
- Getters and setters allow you to define Object Accessors (Computed Properties).

JavaScript Getter (The get Keyword)

This example uses a lang property to get the value of the language property.

Example

```
// Create an object:
var person = {
firstName: "John",
  lastName : "Doe",
  language : "en",
  get lang() {
    return this.language;
  }
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
```

JavaScript Setter (The set Keyword)

- This example uses a lang property to set the value of the language property.

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  language : "",
  set lang(lang) {
    this.language = lang;
```

```
  }
};

// Set an object property using a setter:
person.lang = "en";
// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
```

Data Quality

- JavaScript can secure better data quality when using getters and setters.
- Using the lang property, in this example, returns the value of the language property in upper case:

Example

```
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  language : "en",
  get lang() {
    return this.language.toUpperCase();
  }
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
```

Object.defineProperty()

- The Object.defineProperty() method can also be used to add Getters and Setters:

Example

```
// Define object
var obj = {counter : 0};

// Define setters
Object.defineProperty(obj, "reset", {
  get : function () {this.counter = 0;}
});
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
```

```
});
Object.defineProperty(obj, "subtract", {
  set : function (value) {this.counter -= value;}
});

// Play with the counter:
obj.reset;
obj.add = 5;
obj.subtract = 1;
obj.increment;
obj.decrement;
```

2.7.10 Object Constructors

- Object Types (Blueprints) (Classes)
- The examples from the previous chapters are limited. They only create single objects.
- Sometimes we need a "blueprint" for creating many objects of the same "type".
- The way to create an "object type", is to use an object constructor function.
- In the example above, function Person() is an object constructor function.
- Objects of the same type are created by calling the constructor function with the new keyword:

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

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

The this Keyword

- In JavaScript, the thing called this is the object that "owns" the code.
- The value of this, when used in an object, is the object itself.
- In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.

Adding a Property to an Object

- Adding a new property to an existing object is easy:

Example

```
myFather.nationality = "English";
```

Adding a Method to an Object

- Adding a new method to an existing object is easy:

Example

```
myFather.name = function () {
  return this.firstName + " " + this.lastName;
};
```

Adding a Property to a Constructor

- You cannot add a new property to an object constructor the same way you add a new property to an existing object:

Example

Person.nationality = "English";

Adding a Method to a Constructor

- Your constructor function can also define methods:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.name = function() {return this.firstName + " " + this.lastName;};
}
```

Built-in JavaScript Constructors

- JavaScript has built-in constructors for native objects:

Example

```
var x1 = new Object(); // A new Object objectvar
x2 = new String(); // A new String object
var x3 = new Number(); // A new Number objectvar
x4 = new Boolean(); // A new Boolean objectvar x5
= new Array(); // A new Array object
var x6 = new RegExp(); // A new RegExp objectvar
x7 = new Function(); // A new Function objectvar
x8 = new Date(); // A new Date object
```

String Objects

- Normally, strings are created as primitives: var firstName = "John"
- But strings can also be created as objects using the new keyword: var firstName = new String("John")

Number Objects

- Normally, numbers are created as primitives: var x = 123
- But numbers can also be created as objects using the new keyword: var x = new Number(123)

Boolean Objects

- Normally, booleans are created as primitives: var x = false
- But booleans can also be created as objects using the new keyword: var x = new Boolean(false)

2.7.11 Object Prototypes

- All JavaScript objects inherit properties and methods from a prototype.

Prototype Inheritance

- All JavaScript objects inherit properties and methods from a prototype:
- Date objects inherit from Date.prototype
- Array objects inherit from Array.prototype
- Person objects inherit from Person.prototype
- The Object.prototype is on the top of the prototype inheritance chain:
- Date objects, Array objects, and Person objects inherit from Object.prototype.

Adding Properties and Methods to Objects

- Sometimes you want to add new properties (or methods) to all existing objects of a given type.
- Sometimes you want to add new properties (or methods) to an object constructor.

Using the prototype Property

- The JavaScript prototype property allows you to add new properties to object constructors:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}
Person.prototype.nationality = "English";
```

- The JavaScript prototype property also allows you to add new methods to objects constructors:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
```

```
  this.age = age;
  this.eyeColor = eyecolor;
}
Person.prototype.name  = function() {
  return this.firstName + " " + this.lastName;
};
```

2.8 EVENT HANDLING

- 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.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

2.8.1 onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type

Example

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

   <body>
      <p>Click the following button and see result</p>
      <form>
         <input type = "button" onclick = "sayHello()" value = "Say Hello"
/>
      </form>
   </body>
</html>
```

2.8.2 onsubmit Event Type

- onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

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

   <body>
      <form method = "POST" action = "t.cgi" onsubmit = "return
validate()">
         .......
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

2.8.3 onmouseover and onmouseout

- These two event types will help you create nice effects with images or even with text as well.
- The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element.

```
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
   </body>
```

```
</html>
```

2.8.4 HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged

2.9 DHTML WITH JAVASCRIPT

2.9.1 JavaScript Alone

document.write()

can be used to display dynamic content to a web page.

Example

Using JavaScript to display the current date:

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

```
document.write(Date());
</script>
</body>
</html>
```

2.9.2 JavaScript and the HTML DOM

- With HTML 4, JavaScript can also be used to change the inner content and attributes of HTML elements dynamically.
- To change the content of an HTML element use:

document.getElementById(*id*).innerHTML=*new HTML*

- To change the attribute of an HTML element use:

document.getElementById(*id*).*attribute=new value*

2.9.3 JavaScript and HTML Events

- New to HTML 4 is the ability to let HTML events trigger actions in the browser, like starting a JavaScript when a user clicks on an HTML element.
- To execute code when a user clicks on an element, use the following event attribute:

onclick=*JavaScript*

2.9.4 JavaScript and CSS

- With HTML 4, JavaScript can also be used to change the style of HTML elements.
- To change the style of an HTML element use:

document.getElementById(*id*).style.*property=new style*

2.10 JSON INTRODUCTION

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

2.10.1 Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

2.10.2 Sending Data

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

```
var myObj = {name: "John", age: 31, city: "New York"}; var myJSON = JSON.stringify(myObj); window.location = "demo_json.php?x=" + myJSON;
```

2.10.2 Receiving Data

- If you receive data in JSON format, you can convert it into a JavaScript object:

Example

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}'; var myObj = JSON.parse(myJSON); document.getElementById("demo").innerHTML = myObj.name;
```

2.10.3 Storing Data

- When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.
- JSON makes it possible to store JavaScript objects as text.

Example

Storing data in local storage

```
// Storing data:
myObj = {name: "John", age: 31, city: "New York"};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

What is JSON?

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent *
- JSON uses JavaScript syntax, but the JSON format is text only.Text can be read and used as a data format by any programming language.

Why use JSON?

- Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.
- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

- So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

2.11 SYNTAX

JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following −

- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).

Example

```
{
   "book": [

      {
         "id": "01",
         "language": "Java",
         "edition": "third",
         "author": "Herbert Schildt"
      },

      {
         "id": "07",
         "language": "C++",
         "edition": "second",
         "author": "E.Balagurusamy"
      }

   ]
}
```

JSON supports the following two data structures −

- **Collection of name/value pairs** − This Data Structure is supported by different programming languages.
- **Ordered list of values** − It includes array, list, vector or sequence etc.

2.12 JSON-FUNCTION FILES

- A common use of JSON is to read data from a web server, and display the data in a web page.

JSON Example

```
<div id="id01"></div>
<script>
function myFunction(arr) {
    var out = "";
    var i;
```

```
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' +
arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>

<script src="myTutorials.js"></script>
```

Example Explaination

1: Create an array of objects.

Use an **array literal** to declare an **array** of **objects**.

Give each object two properties: **display** and **url**.

Name the array **myArray**:

myArray

```
var myArray = [
{
"display": "JavaScript Tutorial",
"url": "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "https://www.w3schools.com/css/default.asp"
}
]
```

2: Create a JavaScript function to display the array.

Create a function **myFunction()** that loops the array objects, and display the content as HTML links:

myFunction()

```
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
```

```
    document.getElementById("id01").innerHTML = out;
}
```

Call **myFunction()** with **myArray** as argument:

3: Use an array literal as the argument (instead of the array variable):

Call **myFunction()** with an array **literal** as argument:

Calling myFunction()

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "https://www.w3schools.com/css/default.asp"
}
]);
```

4: Put the function in an external js file

Put the function in a file named **myTutorials.js**:

myTutorials.js

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "https://www.w3schools.com/css/default.asp"
}
]);
```

Add External Script

```
<script src="myTutorials.js"></script>
```

2.13 HTTP REQUEST

JSONRequest is a global JavaScript object. It provides three methods: post, get, and cancel.

JSONRequest.post

- JSONRequest.post does an HTTP POST of the serialization of a JavaScript object or array, gets the response, and parses the response into a JavaScript value. If the parse is successful, it returns the value to the requesting script. In making the request, no HTTP authentication or cookies are sent. Any cookies returned by the server cause the request to fail. The JSONRequest service can only be used to send and receive JSON-encoded values. JSONRequest cannot be used to retrieve other text formats.

JSONRequest.post takes four parameters:

parameter	type	description
url	*string*	The URL to POST to. The URL does not need to be related to the page's URL.
send	*object*	The JavaScript object or array to send as the POST data. It will be serialized as JSON text. Cyclical structures will fail.
done	*function (requestNumber, value, exception)*	The function to be called when the request is completed. If the request was successful, the function will receive the request number and the returned value. If it is not successful, it will receive the request number and an exception object. The done function will not be called until after the call to JSONRequest returns a serial number.
timeout	*number*	The number of milliseconds to wait for the response. This parameter is optional. The default is 10000 (10 seconds).

- JSONRequest.post returns a serial number if the request parameters are acceptable. It throws a JSONRequestError exception if the request is rejected. The request will be rejected if
- The url string is not a properly formatted URL.
- The send value cannot be serialized. It will be rejected if it is not an object or array or if it is cyclical. (Functions and host objects will not be included in the serialization.)
- The done value is not a function.
- The timeout value is not a positive number.

Example:

```
requestNumber = JSONRequest.post(
   "https://json.penzance.org/request",
   {
      user: "doctoravatar@yahoo.com",
      t: "vllj",
      zip: 94089,
      forecast: 7
   },
   function (requestNumber, value, exception) {
      if (value) {
         processResponse(value);
      } else {
         processError(exception);
      }
   }
);
```

- After JSONRequest.post has verified the parameters, it will queue the request and return the request number. The done function value will be invoked later when the outcome of the request is known.
- No cookies or implicit authentication information are sent with the POST operation. Any authentication information must be placed in the send data or in the url. The JSONtext that was serialized from the send data is used as the body of the request. The character encoding is UTF-8. An implementation may choose to gzip the JSON text.
- The request may use either http or https. This choice is independent of the security of the page.

```
POST /request HTTP/1.1
Accept: application/jsonrequest
Content-Encoding: identity
Content-Length: 72
Content-Type: application/jsonrequest
Host: json.penzance.org
```

JSONRequest.get

JSONRequest.get takes three parameters:

parameter	type	description
url	*string*	The URL to GET from. The URL does not need to be related to the page's URL.
done	*function (requestNumber, value, exception)*	The function to be called when the request is completed. If the request was successful, the function will receive the request number and the returned value. If it is not successful, it will receive the request number and an exception object. The done function will not be called until after the call to JSONRequest returns a serial number.
timeout	*number*	The number of milliseconds to wait for the response. This parameter is optional. The default is 10000 (10 seconds).

- JSONRequest.get returns a serial number if the request parameters are acceptable. It throws a JSONRequestError exception if the request is rejected. The request will be rejected if
- The url string is not a properly formatted URL.
- The done value is not a function.

JSONRequest.clear

- A document can be removed from the GET cache by calling JSONRequest.clear with its url. Nothing is returned. It is not possible to determine with this function if the document had ever been in the cache.

```
JSONRequest.clear(url);
```

JSONRequest.cancel

- A request can be canceled by calling JSONRequest.cancel with the request number as the only parameter. Nothing is returned. There is no guarantee that the request will not be sent to the server since it is possible that it had been transmitted before the cancel request was made.

```
JSONRequest.cancel(requestNumber);
```

HTTP Header Fields

Accept

The only accept type used with JSONRequest is application/jsonrequest. The use of this unique type prevents JSONRequest from interacting with legacy systems that assumed that a firewall was sufficient to protect them from unintended web access.

Content-Type

The only content type used with JSONRequest is application/jsonrequest.

Content-Encoding

The content encoding can be identity (the default) or gzip.

Exceptions

Exceptions can be produced either when the JSONRequest function is called, or when the done callback function is invoked. An exception object contains a name member whose value will always be the string "JSONRequestError", and a message member, which contains a string that explains the error.

{name: "JSONRequestError", message: "*error message*"}

These are the messages that can be produced.

message	meaning
"bad URL"	The URL was not formatted correctly and could not be used to make a request.
"bad data"	The send data was not an object or array, or was cyclical, or was too big.
"bad function"	The callback function was not a function with an arity of 3.
"bad timeout"	The timeout parameter is not a positive integer.
"not ok"	The server supplied a response that was not 200 OK.
"no response"	The server did not respond, or a timeout occurred.
"bad response"	The response was not a valid JSON text, or had unexpected material in the HTTP response header.

2.14 SQL-JSON

Customers.html

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

<h1>Customers</h1>
<div id="id01"></div>

<script>
```

```
var xmlhttp = new XMLHttpRequest();
var url = "https://www.w3schools.com/js/customers_mysql.php";

xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this.responseText);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(response) {
    var arr = JSON.parse(response);
    var i;
    var out = "<table>";

    for(i = 0; i < arr.length; i++) {
        out += "<tr><td>" +
        arr[i].Name +
        "</td><td>" +
        arr[i].City +
        "</td><td>" +
        arr[i].Country +
        "</td></tr>";
    }
    out += "</table>";
    document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```