
FRONT END DEVELOPMENT

MODULE 02



FRONT END DEVELOPMENT – JAVA SCRIPT

- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
- It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMAScript is the official name of the language ECMA(European Computer Manufacturers Association).
- ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.
- Since 2016 new versions are named by year (ECMAScript 2016 / 2017 / 2018).
- ES2020s



FEATURES OF JAVA SCRIPT

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.



APPLICATIONS OF JAVASCRIPT

- 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.

JAVASCRIPT EXAMPLE

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

```
document.write("Hello JavaScript");
```

```
</script>
```

JavaScript provides 3 places to put the JavaScript code:

- within body tag
- within head tag
- external JavaScript file.

JAVASCRIPT EXAMPLE : CODE BETWEEN THE BODY TAG

- `<html>`
- `<body>`
- `<script type="text/javascript">`
- `alert("Hello Javascript");`
- `</script>`
- `</body>`
- `</html>`

JAVASCRIPT EXAMPLE : CODE BETWEEN THE HEAD TAG

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    alert("Hello Javascript"); }
</script>
</head>
<body><h2>Demo JavaScript in Head</h2>

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

■ </body>
  </html>
```

EXTERNAL JAVASCRIPT FILE

- We can create external JavaScript file and embed it in many html page.
- It provides **code re usability** because single JavaScript file can be used in several html pages.
- An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.
- **message.js**
 1. function msg(){
 2. alert("Hello JavaScript");
 3. }
 4. <html>
 5. <head>
 6. <script type="text/javascript" src="message.js"></script>
 7. </head>

JAVASCRIPT VARIABLE

- A **JavaScript variable** is simply a name of storage location.
- 1. JavaScript variables are case sensitive, for example x and X are different variables.

<script>

```
var x = 10;
```

```
var y = 20;
```

```
var z=x+y;
```

```
document.write(z);
```

</script>

JAVA SCRIPT DATA TYPES

- JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.
 1. Primitive data type
 2. Non-primitive (reference) data type
- JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:
 1. var **a**=40;//holding number
 2. var **b**="Rahul";//holding string

JAVASCRIPT PRIMITIVE DATA TYPES

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JAVASCRIPT NON-PRIMITIVE DATA TYPES

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JAVASCRIPT IF-ELSE

- The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

<script>

```
var a=20;
```

```
if(a>10){
```

```
document.write("value of a is greater than 10");
```

```
}
```

</script>

JAVASCRIPT SWITCH

- The **JavaScript switch statement** is used *to execute one code from multiple expressions*.

<script>

```
var grade='B';
```

```
var result;
```

```
switch(grade){
```

```
case 'A':
```

```
result="A Grade";
```

```
break;
```

```
case 'B':
```

```
result="B Grade";
```

```
break;
```

```
case 'C':
```

```
result="C Grade";
```

```
break;
```

```
default:
```

```
result="No Grade";
```

```
}
```

```
document.write(result);
```

</script>

JAVASCRIPT LOOPS

- The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
- There are four types of loops in JavaScript.
 1. for loop
 2. while loop
 3. do-while loop
 4. for-in loop

JAVASCRIPT FOR LOOP

```
for (initialization; condition; increment)
{
    code to be executed
}
<script>
for (i=1; i<=5; i++)
{
    document.write(i + "<br/>")
}
</script>
```

JAVASCRIPT WHILE LOOP

```
while (condition)
{
    code to be executed
}
<script>
var i=11;
while (i<=15)
{
    document.write(i + "<br/>");
    i++;
}
</script>
```

```
do{
    code to be executed
}while (condition);
<script>
var i=21;
do{
    document.write(i + "<br/>");
    i++;
}while (i<=25);
</script>
```

JAVASCRIPT DO WHILE LOOP

JAVASCRIPT FUNCTIONS

■ JavaScript Function Syntax

```
function functionName([arg1, arg2, ...argN]){  
  //code to be executed  
}
```

JavaScript Functions can have 0 or more arguments

1. **<script>**
2. `function msg(){`
3. `alert("hello! this is message");`
4. `}`
5. **</script>**
6. `<input type="button" onclick="msg()" "/>`

JAVASCRIPT FUNCTION ARGUMENTS

<script>

```
function getcube(number){  
  //alert(number*number*number);  
  document.write(number*number*number);  
}
```

</script>

<form>

<input type="button" value="click" onclick="getcube(4)"**>/>**

</form>

FUNCTION WITH RETURN VALUE

<script>

```
function getInfo(){  
    return "hello! How r u?";  
}
```

</script>

<script>

```
document.write(getInfo());
```

</script>

JAVASCRIPT OBJECTS

- 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.
 1. By object literal
 2. By creating instance of Object directly (using new keyword)
 3. By using an object constructor (using new keyword)
- 1) JavaScript Object by object literal
- `object={property1:value1,property2:value2.....propertyN:valueN}`

`<script>`

`emp={id:102,name:"Shyam Kumar",salary:40000}`

`document.write(emp.id+" "+emp.name+" "+emp.salary);`

`</script>`

2) BY CREATING INSTANCE OF OBJECT

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Example

```
<script>
```

```
var emp=new Object();
```

```
emp.id=101;
```

```
emp.name="Ravi Malik";
```

```
emp.salary=50000;
```

```
document.write(emp.id+" "+emp.name+" "+emp.salary);
```

```
</script>
```

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)

1) JavaScript array literal

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

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

```
<script>
```

```
var emp=["Ajay","Vimal","Ratan"];
```

```
for (i=0;i<emp.length;i++){
```

```
document.write(emp[i] + "<br/>");
```

```
}
```

```
</script>
```

2) JAVASCRIPT ARRAY DIRECTLY (NEW KEYWORD)

- **Syntax**

- var **arrayname**=**new** Array();

- **Example**

<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] + "**
**");

}

</script>

JAVASCRIPT STRING

- 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
 2. By string object (using new keyword)
- 1) 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";
```

```
<script>
```

```
var str="This is string literal";
```

```
document.write(str);
```

```
</script>
```


2) BY STRING OBJECT (USING NEW KEYWORD)

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

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

Example

```
<script>
```

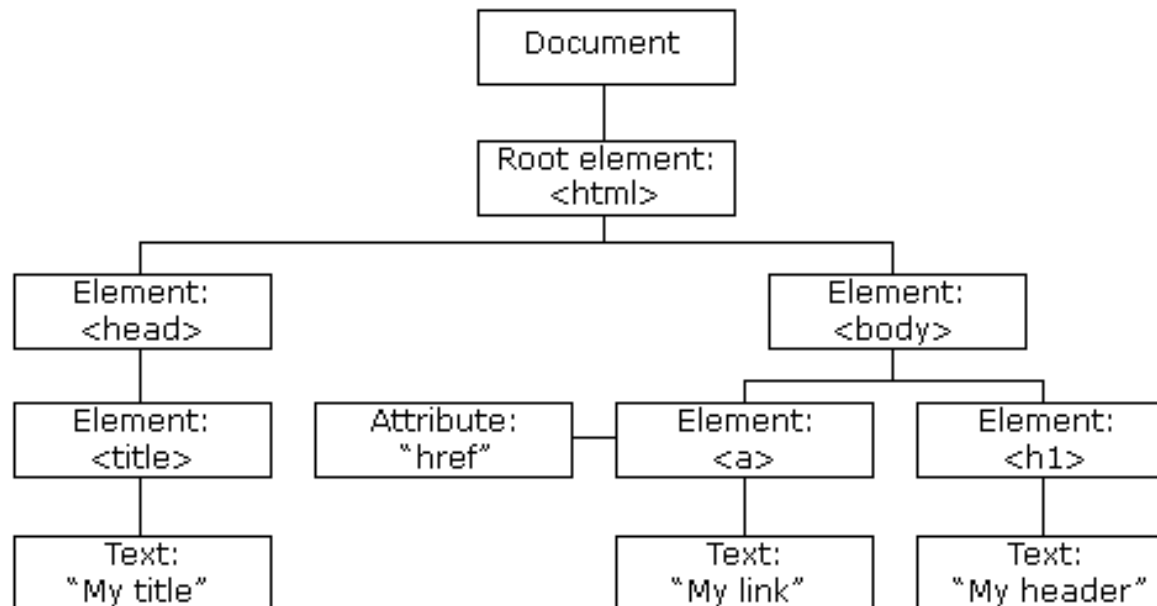
```
var stringname=new String("hello javascript string");
```

```
document.write(stringname);
```

```
</script>
```

JAVASCRIPT DOM MODEL

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- When a web page is loaded, the browser creates a **Document Object Model** of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**



WITH THE OBJECT MODEL, JAVASCRIPT GETS ALL THE POWER IT NEEDS TO CREATE DYNAMIC HTML

- 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 HTML DOM is a standard for how to get, change, add, or delete HTML elements

DOM MODEL-DOCUMENT OBJECT 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
 1. `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."*

THE DOM PROGRAMMING INTERFACE

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.
- A **property** is a value that you can get or set (like changing the content of an HTML element).
- A **method** is an action you can do (like add or deleting an HTML element).
- The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`
- ```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```
- The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

# METHODS OF DOCUMENT OBJECT

- We can access and change the contents of document by its methods.

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

# THE HTML DOM DOCUMENT OBJECT

- Changing HTML Elements

Property	Description
<code>element.innerHTML = new html content</code>	Change the inner HTML of an element
<code>element.attribute = new value</code>	Change the attribute value of an HTML element
<code>element.style.property = new style</code>	Change the style of an HTML element
Method	Description
<code>element.setAttribute(attribute, value)</code>	Change the attribute value of an HTML element

# ADDING AND DELETING ELEMENTS

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



# JAVASCRIPT DATE OBJECT

- 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.
  1. Date()
  2. Date(milliseconds)
  3. Date(dateString)
  4. Date(year, month, day, hours, minutes, seconds, milliseconds)

# JAVASCRIPT DATE METHODS

Methods	Description
<a href="#">getDate()</a>	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.
<a href="#">getDay()</a>	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.
<a href="#">getFullYear()</a>	It returns the integer value that represents the year on the basis of local time.
<a href="#">getHours()</a>	It returns the integer value between 0 and 23 that represents the hours on the basis of local time.
<a href="#">getMilliseconds()</a>	It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.
<a href="#">getMinutes()</a>	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.
<a href="#">getMonth()</a>	It returns the integer value between 0 and 11 that represents the month on the basis of local time.
<a href="#">getSeconds()</a>	It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.
<a href="#">getUTCDate()</a>	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.
<a href="#">getUTCDay()</a>	It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.
<a href="#">getUTCFullYear()</a>	It returns the integer value that represents the year on the basis of universal time.
<a href="#">getUTCHours()</a>	It returns the integer value between 0 and 23 that represents the hours on the basis of universal time.
<a href="#">getUTCMinutes()</a>	It returns the integer value between 0 and 59 that represents the minutes on the basis of universal time.
<a href="#">getUTCMonth()</a>	It returns the integer value between 0 and 11 that represents the month on the basis of universal time.
<a href="#">getUTCSeconds()</a>	It returns the integer value between 0 and 60 that represents the seconds on the basis of universal time.
<a href="#">setDate()</a>	It sets the day value for the specified date on the basis of local time.
<a href="#">setDay()</a>	It sets the particular day of the week on the basis of local time.
<a href="#">setFullYear()</a>	It sets the year value for the specified date on the basis of local time.
<a href="#">setHours()</a>	It sets the hour value for the specified date on the basis of local time.
<a href="#">setMilliseconds()</a>	It sets the millisecond value for the specified date on the basis of local time.
<a href="#">setMinutes()</a>	It sets the minute value for the specified date on the basis of local time.
<a href="#">setMonth()</a>	It sets the month value for the specified date on the basis of local time.
<a href="#">setSeconds()</a>	It sets the second value for the specified date on the basis of local time.

# JAVASCRIPT MATH

- The **JavaScript math** object provides several constants and methods to perform mathematical operation.
- **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.
- `var n=new Number(value);`
  1. `var x=102;//integer value`
  2. `var y=102.7;//floating point value`
  3. `var z=13e4;//exponent value, output: 130000`
  4. `var n=new Number(16);//integer value by number object`

# EXCEPTION HANDLING IN JAVASCRIPT

- In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them.
- It also enables to handle the flow control of the code/program.
- For handling the code, various handlers are used that process the exception and execute the code.
- **For example**, the Division of a non-zero value with zero will result into infinity always, and it is an exception.

Thus, with the help of exception handling, it can be executed and handled.

- A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.

# EXCEPTION HANDLING IN JAVASCRIPT

- **try{} statement:** Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.
- **catch{} statement:** This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.
- Syntax:

```
try{
expression; } //code to be written.
catch(error){
expression; } // code for handling the error.
```

# THROW STATEMENT

- Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

- Syntax:

```
throw exception;
```

```
<html>
```

```
<head>Exception Handling</head>
```

```
<body>
```

```
<script>
```

```
try {
```

```
 throw new Error('This is the throw keyword'); //user-defined throw statement.
```

```
}
```

```
catch (e) {
```

```
 document.write(e.message); // This will generate an error message
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

# TRY...CATCH...FINALLY STATEMENTS

- Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

- **Syntax:**

```
try{
expression;
}
catch(error){
expression;
}
finally{
expression; } //Executable code
```

# JAVA SCRIPT REGULAR EXPRESSION

- **RegExp Object**
- A regular expression is an object that describes a pattern of characters.
- Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.
- Syntax
- */pattern/modifiers;*

## Modifiers

Modifiers are used to perform case-insensitive and global searches:

<u>g</u>	Perform a global match (find all matches rather than stopping after the first match)
<u>i</u>	Perform case-insensitive matching
<u>m</u>	Perform multiline matching



# BRACKETS

- Brackets are used to find a range of characters:

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

# METACHARACTERS

- Metacharacters are characters with a special meaning:

Metacharacter	Description
<code>.</code>	Find a single character, except newline or line terminator
<code>\w</code>	Find a word character
<code>\W</code>	Find a non-word character
<code>\d</code>	Find a digit
<code>\D</code>	Find a non-digit character
<code>\s</code>	Find a whitespace character
<code>\S</code>	Find a non-whitespace character
<code>\b</code>	Find a match at the beginning/end of a word, beginning like this: <code>\bHI</code> , end like this: <code>HI\b</code>
<code>\B</code>	Find a match, but not at the beginning/end of a word
<code>\0</code>	Find a NULL character

# QUANTIFIERS

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

# REGEXP OBJECT PROPERTIES

Property	Description
<a href="#"><u>constructor</u></a>	Returns the function that created the RegExp object's prototype
<a href="#"><u>global</u></a>	Checks whether the "g" modifier is set
<a href="#"><u>ignoreCase</u></a>	Checks whether the "i" modifier is set
<a href="#"><u>lastIndex</u></a>	Specifies the index at which to start the next match
<a href="#"><u>multiline</u></a>	Checks whether the "m" modifier is set
<a href="#"><u>source</u></a>	Returns the text of the RegExp pattern

## REGEXP OBJECT METHODS

Method	Description
<a href="#"><u>compile()</u></a>	Compiles a regular expression
<a href="#"><u>exec()</u></a>	Tests for a match in a string. Returns the first match
<a href="#"><u>test()</u></a>	Tests for a match in a string. Returns true or false
<a href="#"><u>toString()</u></a>	Returns the string value of the regular expression

The `compile()` method is used to compile a regular expression during execution of a script.

**Syntax:**

```
RegExpObject.compile(regex, modifier)
```

The `exec()` method tests for a match in a string.

This method returns the matched text if it finds a match, otherwise it returns null.

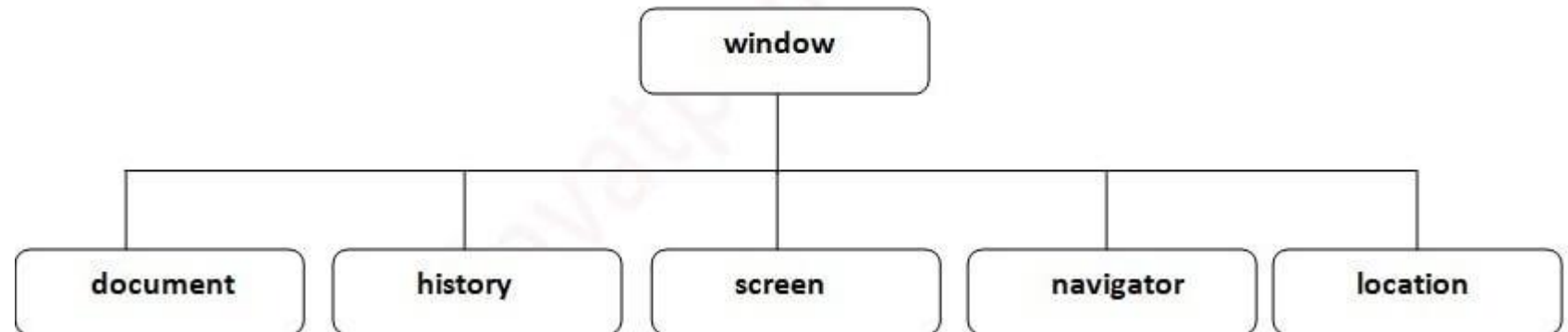
**Syntax:**

```
RegExpObject.exec(string)
```

Return value: An array containing the matched text if it finds a match, otherwise it returns null

# BROWSER OBJECT MODEL

- The **Browser Object Model** (BOM) is used to interact with the browser.
- The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:
  1. `window.alert("hello everyone");`
- is same as:
  1. `alert("hello everyone");`



# WINDOW OBJECT

- The **window object** represents a window in browser. An object of window is created automatically by the browser.
- Methods of window object

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

# JAVASCRIPT HISTORY OBJECT

- The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

`window.history`

1. `history.back();`//for previous page
2. `history.forward();`//for next page
3. `history.go(2);`//for next 2nd page
4. `history.go(-2);`//for previous 2nd page

No.	Method	Description
1	<code>forward()</code>	loads the next page.
2	<code>back()</code>	loads the previous page.
3	<code>go()</code>	loads the given page number.



# JAVASCRIPT NAVIGATOR OBJECT

- The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.
- window.navigator

```
1. <script>
2. document.writeln("
navigator.appCodeName: "+navigator.appCodeName);
3. document.writeln("
navigator.appName: "+navigator.appName);
4. document.writeln("
navigator.appVersion: "+navigator.appVersion);
5. document.writeln("
navigator.cookieEnabled: "+navigator.cookieEnabled);
6. document.writeln("
navigator.language: "+navigator.language);
7. document.writeln("
navigator.userAgent: "+navigator.userAgent);
8. document.writeln("
navigator.platform: "+navigator.platform);
9. document.writeln("
navigator.onLine: "+navigator.onLine);
10. </script>
```

No.	Property	Description
1	appName	returns the name
2	appVersion	returns the version
3	appCodeName	returns the code name
4	cookieEnabled	returns true if cookie is enabled otherwise false
5	userAgent	returns the user agent

# THE WINDOW LOCATION OBJECT

The **location object** contains information about the current URL.

The **location object** is a property of the **window object**.

The **location object** is accessed with:

`window.location` or just `location`

Property	Description
<code>hash</code>	Sets or returns the anchor part (#) of a URL
<code>host</code>	Sets or returns the hostname and port number of a URL
<code>hostname</code>	Sets or returns the hostname of a URL
<code>href</code>	Sets or returns the entire URL
<code>origin</code>	Returns the protocol, hostname and port number of a URL
<code>pathname</code>	Sets or returns the path name of a URL
<code>port</code>	Sets or returns the port number of a URL
<code>protocol</code>	Sets or returns the protocol of a URL
<code>search</code>	Sets or returns the querystring part of a URL

Method	Description
<code>assign()</code>	Loads a new document
<code>reload()</code>	Reloads the current document
<code>replace()</code>	Replaces the current document with a new one

# THE WINDOW SCREEN OBJECT

- The screen object contains information about the visitor's screen.

Property	Description
avail Height	Returns the height of the screen (excluding the Windows Taskbar)
<u>availWidth</u>	Returns the width of the screen (excluding the Windows Taskbar)
<u>colorDepth</u>	Returns the bit depth of the color palette for displaying images
<u>height</u>	Returns the total height of the screen
<u>pixelDepth</u>	Returns the color resolution (in bits per pixel) of the screen
<u>width</u>	Returns the total width of the screen

# JAVASCRIPT EVENTS

- HTML events are "**things**" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.
- **HTML 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

# COMMON HTML EVENTS

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

# REACTING TO EVENTS

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

`<h1 onclick="this.innerHTML = 'Welcome!!!'">Click on this text!</h1>`

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

# JAVASCRIPT FORM VALIDATION

- 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.
- Constraint Validation DOM Methods

checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validation Message property of an input element.




# JAVASCRIPT FORM VALIDATION

- HTML form validation can be done by JavaScript.
- If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted
- JavaScript is often used to validate numeric input



# JAVASCRIPT EMAIL VALIDATION

- There are many criteria that need to be follow to validate the email id such as:
  - email id must contain the @ and . character
  - There must be at least one character before and after the @.
  - There must be at least two characters after . (dot).

- 
- Create a form that has following field “username” ,”Password” and “confirm password” using java script validate each field as follows:
  - 1. Username should of minimum 10 characters
  - 2.Password should contain 1 uppercase letter, 1 lower case letter, 1 digit, and one special character and the length of the password should be minimum 8.
  - 3. Confirm password should match password entered.

# CONSTRAINT VALIDATION DOM PROPERTIES

Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

## Validity Properties

Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

# DHTML JAVASCRIPT

- **DHTML** stands for **Dynamic Hypertext Markup language** i.e., **Dynamic HTML**.
- Dynamic HTML is not a markup or programming language but it is a term that combines the features of various web development technologies for creating the web pages dynamic and interactive.
- **Components of Dynamic HTML**
  - HTML 4.0
  - CSS
  - JavaScript
  - DOM.

# DHTML JAVASCRIPT

- JavaScript can be included in HTML pages, which creates the content of the page as dynamic. We can easily type the JavaScript code within the <head> or <body> tag of a HTML page. If we want to add the external source file of JavaScript, we can easily add using the <src> attribute.
- **JavaScript and HTML event**
- A JavaScript code can also be executed when some event occurs. Suppose, a user clicks an HTML element on a webpage, and after clicking, the JavaScript function associated with that HTML element is automatically invoked. And, then the statements in the function are performed.

HTML (Hypertext Markup Language)	DHTML (Dynamic Hypertext Markup Language)
HTML is a mark-up language.	DHTML is a collection of technology.
HTML is used to build a static document page and to specify the hyperlinks.	DHTML describes the technologies used to build dynamic and interactive web pages.
HTML does not include server-side code.	DHTML includes server-side code.
HTML is a simple static page with no JavaScript or styles.	DHTML is a dynamic page with HTML, CSS, DOM and JavaScript.
HTML files have the.htm or.html extension.	DHTML files have the .dhtml extension.
HTML does not require database connectivity.	DHTML may require database connectivity as it interacts with the user.
HTML pages do not use event methods.	DHTML pages make use of event methods.
HTML provides tags such as <body>, <li>, <form>, etc., to control the presentation of information on the web pages.	DHTML enables the incorporation of minor animations and dynamic menus into Web pages. It employs events, methods, and properties to provide dynamism to HTML pages.
HTML does not permit any changes to the current pages without returning to the webserver first.	DHTML also allows you to modify the current pages at any time. Without initially returning to the webserver.

# JSON INTRODUCTION

- JSON stands for JavaScript Object Notation.
- JSON is lightweight data-interchange format.
- JSON is easy to read and write than XML.
- JSON is language independent.
- JSON supports array, object, string, number and values.
- JSON is a **text format** for storing and transporting data
- JSON is "self-describing" and easy to understand

# WHY USE JSON?

- The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.
- Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.
- JavaScript has a built in function for converting JSON strings into JavaScript objects:
- `JSON.parse()`
- JavaScript also has a built in function for converting an object into a JSON string:
- `JSON.stringify()`



# 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.

```
{"employees":[
 {"name":"Shyam", "email":"shyam@gmail.com"},
 {"name":"Bob", "email":"bob32@gmail.com"},
 {"name":"Jai", "email":"jai87@gmail.com"}
]}
```

# JSON SYNTAX

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
- JSON Data - A Name and a Value
- JSON data is written as name/value pairs
- Ex. `"name": "John"`

# JSON VALUES

■ In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- Null

■ **Example 1:** [1, 2, 7.8, 5, 9, 10];

■ **Example 2:** ["red", "yellow", "green"];

■ **Example 3:** [8, "hello", null, true];

• JSON Object Example

```
{
 "employee": {
 "name": "Ajay",
 "salary": 56000,
 "married": true
 }
}
```

# JSON FUNCTION FILES

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<div id="id01"></div>
```

```
<script>
```

```
function myFunction(arr) {
```

```
 var out = "";
```

```
 var i;
```

```
 for(i = 0; i<arr.length; i++) {
```

```
 out += '' +
```

```
 arr[i].display + '
';
```

```
 }
```

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

```
}
```

```
</script>
```

```
<script src="myfile.js"></script> </body></html>
```

# STEPS

- **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**:
- ```
var myArray = [  
  {  
    "display": "JavaScript Tutorial",  
    "url": "https://js/default.asp"  
  },  
  {  
    "display": "HTML Tutorial",  
    "url": "https://html/default.asp"  
  },  
  {  
    "display": "CSS Tutorial",  
    "url": "https://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
- ```
function myFunction(arr) {
 var out = "";
 var i;
 for(i = 0; i < arr.length; i++) {
 out += '' + arr[i].display + '
';
 }
 document.getElementById("id01").innerHTML = out;
}
```
- Call **myFunction()** with **myArray** as argument:
- `myFunction(myArray);`

### 3: USE AN ARRAY LITERAL AS THE ARGUMENT (INSTEAD OF THE ARRAY VARIABLE):

- Call **myFunction()** with an array **literal** as argument:
- **4: Put the function in an external js file**
- Put the function in a file named **myfile.js**

# JSON HTTP REQUEST

- `<div id="id01"></div>`

```
<script>
var xmlhttp = new XMLHttpRequest();
var url = "myfiles.txt";

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

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



# STEPS

- **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**.
- **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.
- Call **myFunction()** with **myArray** as argument
- **3: Create a text file**
- Put the **array literal** in a file named **myTutorials.txt**
- **4: Read the text file with an XMLHttpRequest**
- Write an **XMLHttpRequest** to read the text file, and use **myFunction()** to display the array

# JSON SQL

```
■ <!DOCTYPE html>
<html>
<body>

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

<script>
var xmlhttp = new XMLHttpRequest();
var url = "https://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>
```

# THE PHP CODE ON THE SERVER

```
■ <?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

$conn = new mysqli("myServer", "myUser", "myPassword", "Northwind");

$result = $conn->query("SELECT CompanyName, City, Country FROM Customers");

$outp = "[";
while($rs = $result->fetch_array(MYSQLI_ASSOC)) {
 if ($outp != "[") {$outp .= ",";}
 $outp .= '{"Name":"' . $rs["CompanyName"] . '",' . '
 $outp .= '"City":"' . $rs["City"] . '",' . '
 $outp .= '"Country":"' . $rs["Country"] . '"}';
}
$outp .= "]";

$conn->close();

echo($outp);
?>
```