Introduction to

DHTML

Drawbacks of HTML & CSS

- Is good for publishing only static documents.
- The content cannot be changed once it has been delivered to the browser .
- Though CSS-2 and CSS-3 have bought in some interactivity, the scope is limited.

Interactive Technologies

- Client-side techniques
  - JavaScript is
    - a language for extending HTML to embed small programs.
    - the most popular scripting language on the internet, and it works on all major browser
- Dynamic HTML Technologies
  – Combination of HTML, Cascading Style Sheet and some scripting language.
  – Provides more control over the appearance, layout and behavior of the web page.
- The Document Object Model (DOM)
  - Defines a standard set of objects for HTML, and a standard way to access and manipulate HTML objects.

What is JavaScript?

- JavaScript is designed by Brendan Eich, in 1995
- Many JavaScript engines are based on ECMA script specification
- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A JavaScript consists of lines of executable computer code
- A JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license
- Many HTML editors supply a library of common code that can be adapted and used in pages.

What can JavaScript Do?

- **JavaScript gives HTML designers a programming tool -** but JavaScript is a scripting language with a very simple syntax!
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server.
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

Advantages of using JavaScript

- It is widely supported in Web Browsers.
- It gives easy access to the document objects and can manipulate most of them.
- Can be used for animation without download time
- Web surfers don’t need special plugins to use the scripts.

Issues with JavaScript

- Access to objects differ from browser to browser.
- If script does not work, page is useless
- Web surfers may disable JavaScript support in the browser
- Can run slowly and complex scripts take long time to start up.
- Browser authors contain this risk using two restrictions.
  - Scripts can only perform web-related actions, not general-purpose programming tasks like creating files.
  - Second, scripts are constrained by the same origin policy: scripts from one web site do not have access to information such as usernames, passwords, or cookies sent to another site.

JavaScript Statements

- Single line comments start with //.
- Multi line comments start with /* and end with */.
- Scripts require neither a main function nor an exit condition.
- JavaScript code is case sensitive.
- Each statement is executed by the browser in the sequence they are written.
- Each line of code terminated by semicolon.
- Functions
  - have parameters which are passed inside parenthesis
  - Statements inside a function can also be grouped together in blocks using {}
  - Functions will not be executed before the event occurs.

Where to Put the JavaScript?

- **Scripts in the head section:** Scripts to be executed when they are called, or when an event is triggered, go in the head section.
- <HTML>

  <HEAD>

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

  </HEAD>

  Scripts in the body section: Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.
- <HTML>

  <HEAD> …

  </HEAD>

  <BODY>

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

  </BODY>

Scripts in both the body and the head section

Where to Put the JavaScript?

- **Using an External JavaScript**
- To run the same JavaScript on several pages, without having to write the same script on every page.
- Write a JavaScript in an external file. Save the external JavaScript file with a .js file extension.
- The external script cannot contain the <script> tag!
- To use the external script, point to the .js file in the "src" attribute of the <script> tag:
- <HTML>

  <HEAD> <script src="xxx.js"> …. </script>

  </HEAD>
- Best Practise :
  - Execute a JavaScript when an **event** occurs, such as when a user clicks a button.
  - When this is the case we can put the script inside a **function**.
- Events are normally used in combination with functions (like calling a function when an event occurs).

JavaScript Functions

- Functions can be defined both in the <head> and in the <body> section of a document.
- Syntax :

  function *functionname*(*var1,var2,...,varX*){
  some code
  }
- Function name(parameters) In JavaScript parameters are passed as arrays.
- A function with no parameters must include the parentheses () after the function name.
- The word *function* must be written in **lowercase letters**, otherwise a JavaScript error occurs.
- **The return statement** is used to specify the value that is returned from the function.

HTML DOM Event Object

Attribute	Description	W3C
onblur	The event occurs when an element loses focus	Yes
onchange	The event occurs when the content of an element, the selection, or the checked state have changed	Yes
onclick	The event occurs when the user clicks on an element	Yes
ondblclick	The event occurs when the user double-clicks on an element	Yes
onerror	The event occurs when an error occurs while loading an external file	Yes
onfocus	The event occurs when an element gets focus	Yes
onkeydown	The event occurs when the user is pressing a key or holding down a key	Yes
onkeypress	The event occurs when the user is pressing a key or holding down a key	Yes
onkeyup	The event occurs when a keyboard key is released	Yes
onload	The event occurs when an object has been loaded	Yes
onmousedown	The event occurs when a user presses a mouse button over an element	Yes
onmousemove	The event occurs when a user moves the mouse pointer over an element	Yes
onmouseout	The event occurs when a user moves the mouse pointer out of an element	Yes
onmouseover	The event occurs when a user mouse over an element	Yes
onmouseup	The event occurs when a user releases a mouse button over an element	Yes
onresize	The event occurs when the size of an element has changed	Yes
onselect	The event occurs after some text has been selected in an element	Yes
onunload	The event occurs before the browser closes the document	Yes

Alert Box

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

Alert box example

```
<html>
<head>
    <script type="text/javascript">
    function displaymessage()
    { alert("Hello World!");
     }
    </script>
 </head>
<body>
  <form> <input type="button" value="Click me!"     onclick="displaymessage()" >
</form>
</body>
</html>
```

Confirm box

- is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.
- **Syntax:** confirm("sometext");

Confirm Box example

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

```
<body>
<input type="button"
   onclick="disp_confirm()"
   value="Display a confirm box"
   />
</body>
</html>
```

Prompt Box

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

Prompt Box example

```
<html>
<head>
<script type="text/javascript">
function disp_prompt()
{
var name=prompt("Please enter
   your name","");
if (name!=null&&name!="" )
  {
  document.write("Hello " + name +
    "! How are you today?");
  }
}
</script>
</head>
<body>
  <input type="button"
    onclick="disp_prompt()"
    value="Display a prompt
    box" />
</body>
</html>
```

Data Types

- Primitive
  - String
  - Number
  - Boolean
- Composite
  - Array
  - Object
- Special
  - Null
  - Undefined

Date object

- new Date() // current date and time
- new Date(milliseconds) //milliseconds since 1970/01/01
- new Date(dateString)
  - var d = new Date("July 21, 1983 01:15:00");
- new Date(year, month, day, hours, minutes, seconds, milliseconds)
  - var d = new Date(1986,07,09,08,17,06,88);

Example

```
<html>
<head><script>
function myFunction() {
    var d = new Date(1986,07,09,08,17,06,88);
    document.getElementById("demo").innerHTML = d.toString();
}</script></head>
<body>
<p>Click the button to display the date.</p>
<input type="button" onclick="myFunction()">Click</button>
<p id="demo"></p>
</body>
</html>
```

- Sat Aug 09 1986 08:17:06 GMT+0530 (India Standard Time)

Methods

getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1 1970, and a specified date
getTimezoneOffset()	Returns the time difference between UTC time and local time, in minutes

Methods

getUTCDate()	Returns the day of the month, according to universal time (from 1-31)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time (four digits)
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)

Methods

now()	Returns the number of milliseconds since midnight Jan 1, 1970
parse()	Parses a date string and returns the number of milliseconds since January 1, 1970
setDate()	Sets the day of the month of a date object
setFullYear()	Sets the year (four digits) of a date object
setHours()	Sets the hour of a date object
setMilliseconds()	Sets the milliseconds of a date object
setMinutes()	Set the minutes of a date object
setMonth()	Sets the month of a date object
setSeconds()	Sets the seconds of a date object
setTime()	Sets a date to a specified number of milliseconds after/before January 1, 1970

Methods

setUTCDate()	Sets the day of the month of a date object, according to universal time
setUTCFullYear()	Sets the year of a date object, according to universal time (four digits)
setUTCHours()	Sets the hour of a date object, according to universal time
setUTCMilliseconds()	Sets the milliseconds of a date object, according to universal time
setUTCMinutes()	Set the minutes of a date object, according to universal time
setUTCMonth()	Sets the month of a date object, according to universal time
setUTCSeconds()	Set the seconds of a date object, according to universal time
toDateString()	Converts the date portion of a Date object into a readable string

Methods

toISOString()	Returns the date as a string, using the ISO standard
toJSON()	Returns the date as a string, formatted as a JSON date
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
UTC()	Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time

Example

```
const event = new Date(Date.UTC(2019, 11, 20, 3, 0, 0));
const options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric'};
console.log(event.toLocaleDateString('hi', options));
console.log(event.toLocaleDateString(undefined, options));
```

"शुक्रवार, 20 दिसंबर 2019“

"Friday, December 20, 2019"

String object

- Syntax: var txt = new String("*string*");

String methods

charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings
endsWith()	Checks whether a string ends with specified string/characters
includes()	Checks whether a string contains the specified string/characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string

String methods

match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters

String methods

substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string

Variables

- Variable names
  - must begin with a letter, digit or an underscore.
  - Cannot use spaces
  - are case sensitive
  - Cannot be reserved words
- Examples
  - var first = 23;
  - var second=“Some words”
  - var first_bool=true;
  - Objects MyObj= new Object();
- The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.
  - Ex: var txt="We are the so-called \"Vikings\" from the north."; document.write(txt);

Boolean Object

- Syntax:

  var myBoolean=new Boolean();

- Boolean Object Methods:
  - toString() :Converts a Boolean value to a string, and returns the result
  - valueOf() :Returns the primitive value of a Boolean object

- Note:

If the Boolean object has no initial value, or if the passed value is one of the following: 0, -0, null, "“, false, undefined, NaN

- Then the object it is set to false.
- Else for any other value it is set to true (even with the string "false")!

Number Object

- Syntax: var num = new Number(value);
- 0-51 bits for number, 52-62 bits for exponent, 63rd bit for sign
- Number Object Properties
  - MAX_VALUE - Returns the largest number possible in JavaScript
  - MIN_VALUE - Returns the smallest number possible in JavaScript
  - NEGATIVE_INFINITY - Represents negative infinity (returned on overflow)
  - POSITIVE_INFINITY - Represents infinity (returned on overflow)
- Number Object Methods
  - toExponential(x) Converts a number into an exponential notation
  - toFixed(x) Formats a number with x numbers of digits after the decimal point
  - toPrecision(x) Formats a number to x length
  - toString() Converts a Number object to a string
  - valueOf() Returns the primitive value of a Number object

Math Object

- The Math object allows you to perform common mathematical tasks.
- Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.
- Method include abs(x), random(),sin(x) etc.

Ex : var pivalue=Math.PI;

```
var sqrt_value=Math.sqrt(16);
document.write(Math.round(4.7));
```

Statements & Operators

- Supports if …else statements
- for(counter=0;counter <=n ; counter++)
- while (boolean condition)
- break – to leap out of the middle of the loop
- continue – to remain within the loop
- Switch statement.
- Operators include
  - Arithmetic operators : + , - , * , /, % , ++, --
  - Assignment operators : +=, -=, *= , /= , %=,
  - Comparison operators : == ,!= , <, > , <= , >=
  - Logical operators : &&, || , !
- To add two or more string variables together, use the + operator.
  - txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
- Special operators
  - New – used for instantiation of objects
  - This – used to refer to the current object
  - With – with object
  - Delete - used to delete an object , an object's property or a specified element in an array

Creating Arrays

- var days=[“Mon”,”Tue”,”Wed”,’Thur”,”Fri”];
- var days= new Array(“Mon”,”Tue”);
- Can hold mixed types
- var data= [“Mon”,23,23.4]

Example using array

```
<HTML>
 <HEAD>
  <TITLE>Looping through an array</TITLE>
 </HEAD>
 <BODY>
                <SCRIPT LANGUAGE="JavaScript">
                 document.writeln("<H1>Looping example</H1>");
                document.write("<P>");
                var data=[“Hello",55,84.699];
                var len=data.length;
                for ( var i = 0; i < len ; i++) {
                                document.write(data[i]+",");
                }
                document.write("</P>");
                document.close();
                </SCRIPT>
 </BODY>
</HTML>
```

Array Object

- Properties include length
- Methods include concat( ), pop( ), push( ), reverse( ), sort() etc.
- Ex :
  - var myCars=new Array();
    myCars[0]="Saab";
    myCars[1]="Volvo";
    myCars[2]="BMW";
  - Or as var myCars=new Array("Saab","Volvo","BMW");
  - Or as var myCars=["Saab","Volvo","BMW"];
- Access an Array
  - You can refer to a particular element in an array by referring to the name of the array and the index number.
  - The index number starts at 0.
  - The following code line:
    - document.write(myCars[0]);

For … in Statement

- is used to loop (iterate) through the elements of an array or through the properties of an object.
- The code in the body of the for ... in loop is executed once for each element/property.
- **Syntax**

for (variable in object) { *code to be executed* } The variable argument can be a named variable, an array element, or a property of an object.

Example

```
<html>
<body>
    <script type="text/javascript">
    var x;
    var mycars = new Array();
    mycars[0] = "Saab"; mycars[1] = "Volvo"; mycars[2] = "BMW";
     for (x in mycars) {
        document.write(mycars[x] + "<br />");
     }
     </script>
 </body>
</html>
```

Introduction

- **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.
  - defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.
  - is a W3C standard.
  - defines a standard for accessing documents like HTML and XML:
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents

RegExp Object

- A regular expression is an object that describes a pattern of characters.
- When you search in a text, you can use a pattern to describe what you are searching for.
- Syntax:
  - var patt=new RegExp(pattern,modifiers);
  - or var patt=/pattern/modifiers;
- pattern specifies the pattern of an expression and modifiers specify if a search should be global, case-sensitive, etc.
- Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Regular Expressions - Brackets

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character not between the brackets
[0-9]	Find any digit from 0 to 9
[A-Z]	Find any character from uppercase A to uppercase Z
[a-z]	Find any character from lowercase a to lowercase z
[A-z]	Find any character from uppercase A to lowercase z
(x|y|z)	Find any of the alternatives specified

Regular Expressions Metacharacters

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character

Regular Expressions - Quantifiers

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

Regualar Expression – Modifier and methods

Property	Description
global	Specifies if the "g" modifier is set
ignoreCase	Specifies if the "i" modifier is set
multiline	Specifies if the "m" modifier is set

Method	Description
exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false

Note

- .(dot): Is a literal in bracket based expression
- These {}[]()^$.|*+? and \ may or may not be considered as metacharacters. Use \ (backslash) to convey the literal meaning

Example

- [hc]?at matches "at", "hat", and "cat".
- [hc]*at matches "at", "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on.
- [hc]+at matches "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on, but not "at".
- cat|dog matches "cat" or "dog".

Example

- String: ManipalMIT, MAHEManipal

  Pattern:/\BManipal/

  Match: ManipalMIT, MAHE**Manipal**
- String: ManipalMIT, MAHEManipal

  Pattern:/Manipal\B/

  Match: **Manipal**MIT, MAHEManipa**l**
- String: ManipalMIT, MAHEManipal

  Pattern:/Manipal\b/

  Match: ManipalMIT, MAHE**Manipal**

Examples

- <script>

```
let text = “Hello World";
let result = text.match(/world/i);
document.getElementById("demo").innerHTML = result;
```

</script>

- <script>

```
let text = "HELLO, LOOK AT YOU!";
let result = text.search(/\bLO/);
document.getElementById("demo").innerHTML = result;
```

</script>

Examples

- <script>
```
let text = "Hello World! Hellooo world! Helloooo MIT";
let result = text.match(/o+/g);
document.getElementById("demo").innerHTML = result;
```
</script>
- <script>
```
let text = "Hello World! Hellooo world! Helloooo MIT";
let result = text.match(/lo*/);
document.getElementById("demo").innerHTML = result;
```
</script>

Examples

- <script>

```
const obj = /e/.exec("Hello World! Hellooo world! Helloooo MIT");
document.getElementById("demo").innerHTML =
 "Found " + obj[0] + "in position" + obj.index +" in the text: " + obj.input;
</script>
```

- <script>

```
        let text = document.getElementById("p01").innerHTML;
        const pattern = /e/;
        document.getElementById("demo").innerHTML = pattern.test(text);
</script>
```

Form validation using RegExp

```
<script type="text/javascript">
      function validate() {
         var v1 = document.getElementById("e").value;
         var v2 = document.getElementById("e");
         var re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
         if (re.test(v1)) {
            alert("done");
            return true;
         }
         else {
            v2.style.border = "red solid 3px";
            return false;
         }
      }
 </script>
```

Form validation using RegExp

```
<script type="text/javascript">
        function validate() {

            var v1 = document.getElementById("c").value;
            var v2 = document.getElementById("c");
            var re = /^[7-9][0-9]{9}$/;
            if (re.test(v1)) {
                alert("done");
                return true;
            }
            else {

                v2.style.border = "red solid 3px";
                return false;
            }
        }
</script>
```

Example

- String: ManipalMIT, MAHEManipal

  Pattern: /^Manipal.*$/

  Match: **ManipalMIT, MAHEManipal**

Example

- 1280x720 , 1920x1600 , 1024x768
  - (\d+)x(\d+)
  - 1(\d{3})x[7|1](\d){2,3}
- Jan 1987 , May 1969, Aug 2011
  - [A-z]{3}\s\d{4}
- file_record_transcript.pdf, file_07241999.pdf

  (file_.+)\.pdf$

Examples

- Number range

1. 000..255
   - ^([01][0-9][0-9]|2[0-4][0-9]|25[0-5])$
2. 1..999
   - ^([1-9]|[1-9][0-9]|[1-9][0-9][0-9])$
3. 0 or 000..999
   - ^[0-9]{1,3}$

Example

- Pattern format : yyyy-mm-dd
  - ^(19|20)\d\d[- /](0[1-9]|1[012])[- /](0[1-9]|[12][0-9]|3[01])$
- Email
  - \w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
- All MasterCard numbers start with the numbers 51 through 55. All have 16 digits.
  - ^5[1-5][0-9]{14}$

Example on modifier “m”

- String: ManipalMIT, MAHE Manipal \nManipal MAHE \nmanipal
- Pattern: /^Manipal/mig
- Match: **Manipal**MIT, MAHE Manipal \n**Manipal** MAHE \n**manipal**
- The m modifier treat beginning (^) and end ($) characters to match the beginning or end of **each line** of a string (delimited by \n or \r)
- Rather than just the beginning or end of the string.
- The m modifier is case-sensitive and will stop the search after the first match
- To perform a global, case-insensitive, multiline search, use this modifier together with "g" and “i”

```
<html><body>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var str = “1999-09-31";
    var patt1 = /^(19|20)\d\d[- /](0[1-9]|1[012])[- /](0[1-9]|[12][0-9]|3[01])$/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}</script></body>
</html>
```

Form Object

- The Form object represents an HTML <form> element.
- https://www.w3schools.com/jsref/dom_obj_form.asp

Accessing a form

<form id="form1">

< input type="text" id ="t1" name="fname" /></form>

- document.getElementById("form1");
- document.forms.namedItem("form1");
- document.forms.item(0);
- document.forms[0];

Form Object

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

<script>
function myFunction() {
  var x = document.createElement("FORM");
  x.setAttribute("id", "myForm");
  document.body.appendChild(x);

  var y = document.createElement("INPUT");
  y.setAttribute("type", "text");
  y.setAttribute("value", "Donald");
  document.getElementById("myForm").appendChild(y);
}
</script>
```

Form Object Properties

Property	Description
acceptCharset	Sets or returns the value of the accept-charset attribute in a form
action	Sets or returns the value of the action attribute in a form
autocomplete	Sets or returns the value of the autocomplete attribute in a form
encoding	Alias of enctype
enctype	Sets or returns the value of the enctype attribute in a form
length	Returns the number of elements in a form
method	Sets or returns the value of the method attribute in a form
name	Sets or returns the value of the name attribute in a form
noValidate	Sets or returns whether the form-data should be validated or not, on submission
target	Sets or returns the value of the target attribute in a form

Form Object Methods

Method	Description
reset()	Resets a form
submit()	Submits a form

Form Object Collections

- elements: Returns a collection of all elements in a form
- *formObject*.elements

Properties

length	Returns the number of elements in the <form> element. **Note:** This property is read-only

Form Object Collections..

Methods

Method	Description
[*index*]	Returns the element in <form> with the specified index (starts at 0). **Note:** Returns null if the index number is out of range
item(*index*)	Returns the element in <form> with the specified index (starts at 0). **Note:** Returns null if the index number is out of range
namedItem(*id*)	Returns the element in <form> with the specified id. **Note:** Returns null if the id does not exist

Form Object Collections..

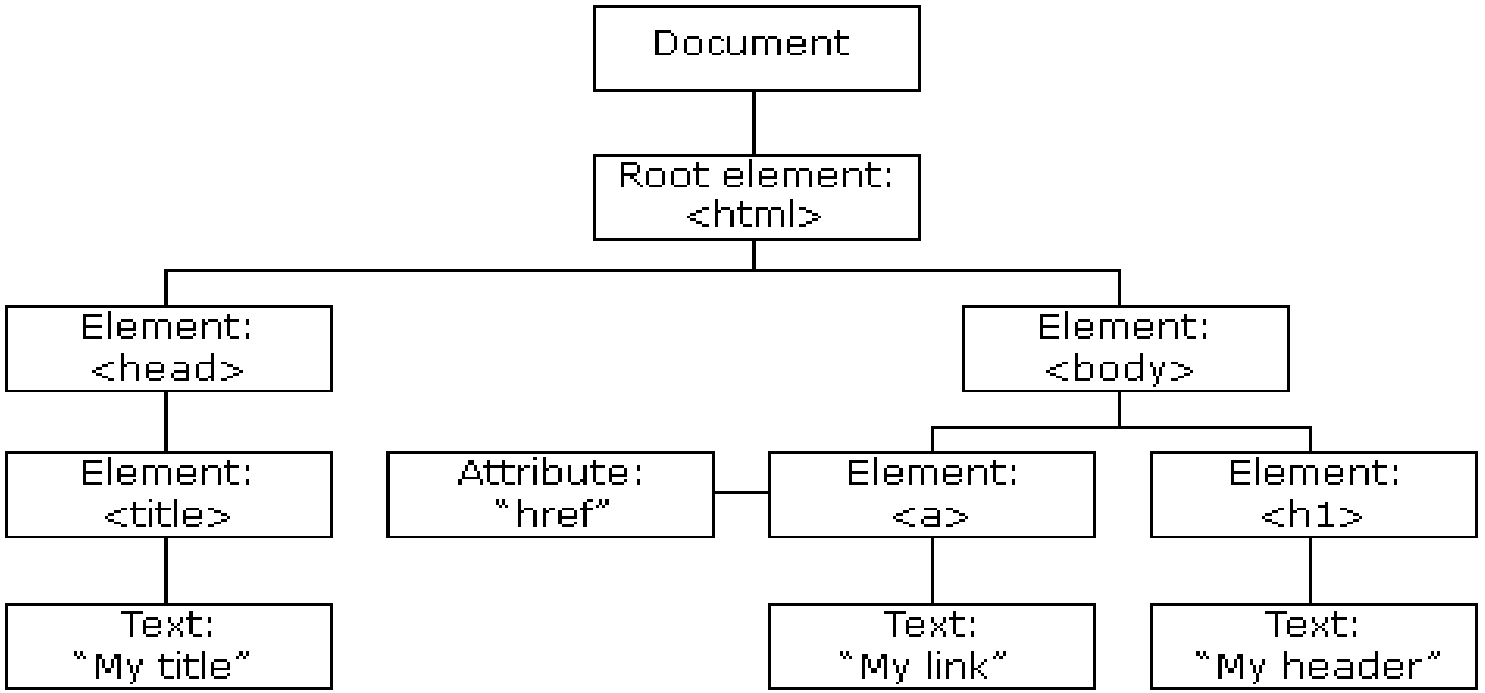
Methods eg:

- document.getElementById(“form1").elements[0].value;
- document.getElementById(“form1").elements.namedItem("fname").value;

DOM Nodes

- The DOM says:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text in the HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - Comments are comment nodes
- The programming interface of the DOM is defined by standard properties and methods.

DOM Tree


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

Typical DOM properties & methods

- **DOM properties**
  - x.innerHTML - the inner text value of x (a HTML element)
  - x.nodeName - the name of x
  - x.nodeValue - the value of x
  - x.parentNode - the parent node of x
  - x.childNodes - the child nodes of x
  - x.attributes - the attributes nodes of x
  - document.documentElement - returns the root node of the document
  - document.body - gives direct access to the <body> tag
  - document.cookie -Returns all name/value pairs of cookies in the document
  - document.domain -Returns the domain name of the server that loaded the document
  - document.forms Returns a collection of all <form> elements in the document
  - document.head Returns the <head> element of the document
  - document.images Returns a collection of all <img> elements in the document
  - document.anchors Returns a collection of all <a> elements in the document that have a name attribute

Typical DOM properties & methods

- **DOM properties**
- document.links -Returns a collection of all <a> and <area> elements in the document that have a href attribute
- document.write() -Writes HTML expressions or JavaScript code to a document
- document.writeln() - Same as write(), but adds a newline character after each statement

Typical DOM properties & methods

- **DOM methods**
  - x.getElementById(*id*) - get the element with a specified id
  - x.getElementsByTagName(*name*) - get all elements with a specified tag name
  - x.appendChild(*node*) - insert a child node to x
  - x.removeChild(*node*) - remove a child node from x
- document.createAttribute() Creates an attribute node
- document.createComment() Creates a Comment node with the specified text
- document.createElement() Creates an Element node
- document.createTextNode() Creates a Text node
- document.getElementsByClassName()- Returns a NodeList containing all elements with the specified class name

Type check

```
<html>
<body>
<script>
var x = "John";             // x is a string
var y = new String("John");  // y is an object

if(x===y)
alert("equal");
else alert("Not equal");
</script>
</body>
</html>
```

Example

```
<html>
<body>
<p id="intro">Hello World!</p>
        <script type="text/javascript">
        txt=document.getElementById("intro").innerHT
ML;
        document.write("<p>The text from the intro
paragraph: " + txt + "</p>");
        </script>
</body>
</html>
```

Example

```
<html>
< body>
< p id="p1">Hello World!</p>

< script>
document.getElementById("p1").innerHTML="New text!";
< /script>

< /body>
< /html>
```

Example

```
<html>
  <title>Illustrate the use of getElementByID</title>
<body>
    <p id="intro">Example</p>
    <div id="main">
                <p id="main1">The DOM is very useful</p>
                <p id="main2">This example demonstrates how to use the <b>getElementById</b>
    method</p>
    </div>
     <script type="text/javascript">
    x=document.getElementById("intro");
    document.write("Intro paragraph text: " + x.innerHTML);
     </script>
</body>
</html>
```

Example

```
<html>
<body id="body1">
		<p>Hello World!</p>
<div id="main">
		<p>The DOM is very useful.</p>
		<p>This example demonstrates the <b>getElementsByTagName</b> method</p>
</div>
<script language="javascript" type="text/javascript">
		var x=document.getElementById("body1");
		var y=x.getElementsByTagName("p");
	for(var ii=0;ii<y.length;ii++)
		document.write(y[ii].innerHTML +"<br/>");
</script>
</body>
</html>
```

Creating new element or node

```
<html>
<style>
                .pstyle{
                color:blue;
                  text-align:right;  }
</style>
<body>
<div id="div1"><p id="p1">This is a paragraph.</p><p id="p2">This is another paragraph.</p>
</div>
<script>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);
//para.style.color="red";
//para.style.textAlign="right";
//para.align="center";
para.className="pstyle";
var element=document.getElementById("div1");
element.appendChild(para);
</script>
</body>
</html>
```

Creating new element before existing element

```
<html>
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);
var element=document.getElementById("div1");
var child=document.getElementById("p1");
element.insertBefore(para,child);
</script>

</body>
</html>
```

Replacing an element

```
<html>
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```

Removing Existing HTML Elements

```
<html>
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
parent.removeChild(child);
</script>

</body>
</html>
```

```
<html><head>
 <title>Attributes example</title>
 <script type="text/javascript">
  function listAttributes() {
    var paragraph = document.getElementById("paragraph");
    var result = document.getElementById("result");
if (paragraph.hasAttributes()) {
      var attrs = paragraph.attributes;
      var output = "";
      for(var i = attrs.length - 1; i >= 0; i--) {
        output += attrs[i].nodeName + "->" + attrs[i].nodeValue;}
      result.innerText = output;
    } else {
      result.innerText = "No attributes to show"; }  }
 </script>
</head>
<body>
 <p id="paragraph" style="color: green;">Sample Paragraph</p>
   <input type="button" value="Show first attribute name and value"
     onclick="listAttributes();">
   <p id="result"></p></body></html>
```

Set and get attribute methods

```
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);

para.setAttribute("align","center");
var getsttri=para.getAttribute("align");
alert(getsttri);
```

Try...Catch Statement

- The try...catch statement allows you to test a block of code for errors.
- The try block contains the code to be run
- The catch block contains the code to be executed if an error occurs.
- **Syntax**

```
Try
  { //Run some code here }
catch(err)
{ //Handle errors here }
```

Example of try catch

```
<html>
<head> <script type="text/javascript">
var txt="" ;
function message()
{
try {
  adddlert("Welcome guest!");
  }
catch(err) {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.description + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
}
 }
 </script>
</head>
<body>
        <input type="button" value="View message" onclick="message()" />
</body>
</html>
```

The Throw Statement

- The throw statement is used to create an exception.
- If you use this statement together with the try...catch statement, the program flow can be controlled and accurate error messages can be generated.
- **Syntax:** throw (exception)

Try Catch with throw

```
<html>
<body>
<script type="text/javascript">
 var x=prompt("Enter a number between 0 and 10:","");
 try
{
     if(x>10)
         throw "Err1";
     else if(x<0)
         throw "Err2";
}
catch(er)
{
   if(er=="Err1")
      alert("Error! The value is too high");
   if(er == "Err2")
      alert("Error! The value is too low");
}
</script>
</body>
</html>
```

The onerror Event

- The onerror event is fired whenever there is a script error in the page.
- To use the onerror event, you must create a function to handle the errors.
- Then you call the function with the onerror event handler.
- The event handler is called with three arguments:
  - msg (error message)
  - url (the url of the page that caused the error)
  - the line (the line where the error occurred).
- **Syntax**

  onerror=handleErrfunction handleErr(msg,url,l)
  { //Handle the error here return true or false }

Example

```
<html>
 <head>
    <script type="text/javascript">
       window.onerror=handleErr;
       var txt="";
function handleErr(msg,url,l)
{
   txt="There was an error on this page.\n\n";
   txt+="Error: " + msg + "\n";
   txt+="URL: " + url + "\n"; txt+="Line: " + l + "\n\n"; txt+="Click OK to continue.\n\n";
   alert(txt);
   return true;
}
function message()
{
        adddlert("Welcome guest!"); }
        </script>
            </head>
               <body> <input type="button" value="View message" onclick="message()" />                    </body>
</html>
```

Example

```
<!DOCTYPE html>
<html>
<body>
<h3>On error event handler</h3>
<img id="myImg" src="img_villa" alt="The villa" onerror="handle(this)">
<button onclick="myFunction()">Try it</button> <p id="demo"></p>
<script>
function handle(im){
 im.onerror=null;
 im.src="villa.jpg";}
function myFunction() {
  var x = document.getElementById("myImg").src;
  document.getElementById("demo").innerHTML = x;}
</script>
</body>
</html>
```

Cookie

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

- When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

Cookie using JavaScript

JavaScript can create, read, and delete cookies with the **document.cookie** property.

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

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

HTML Canvas

- The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript.
- Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Event Object

- All event objects in the DOM are based on the Event Object.
- Therefore, all other event objects (like MouseEvent and KeyboardEvent) has access to the Event Object's properties and methods.

https://developer.mozilla.org/en-US/docs/Web/API/Event