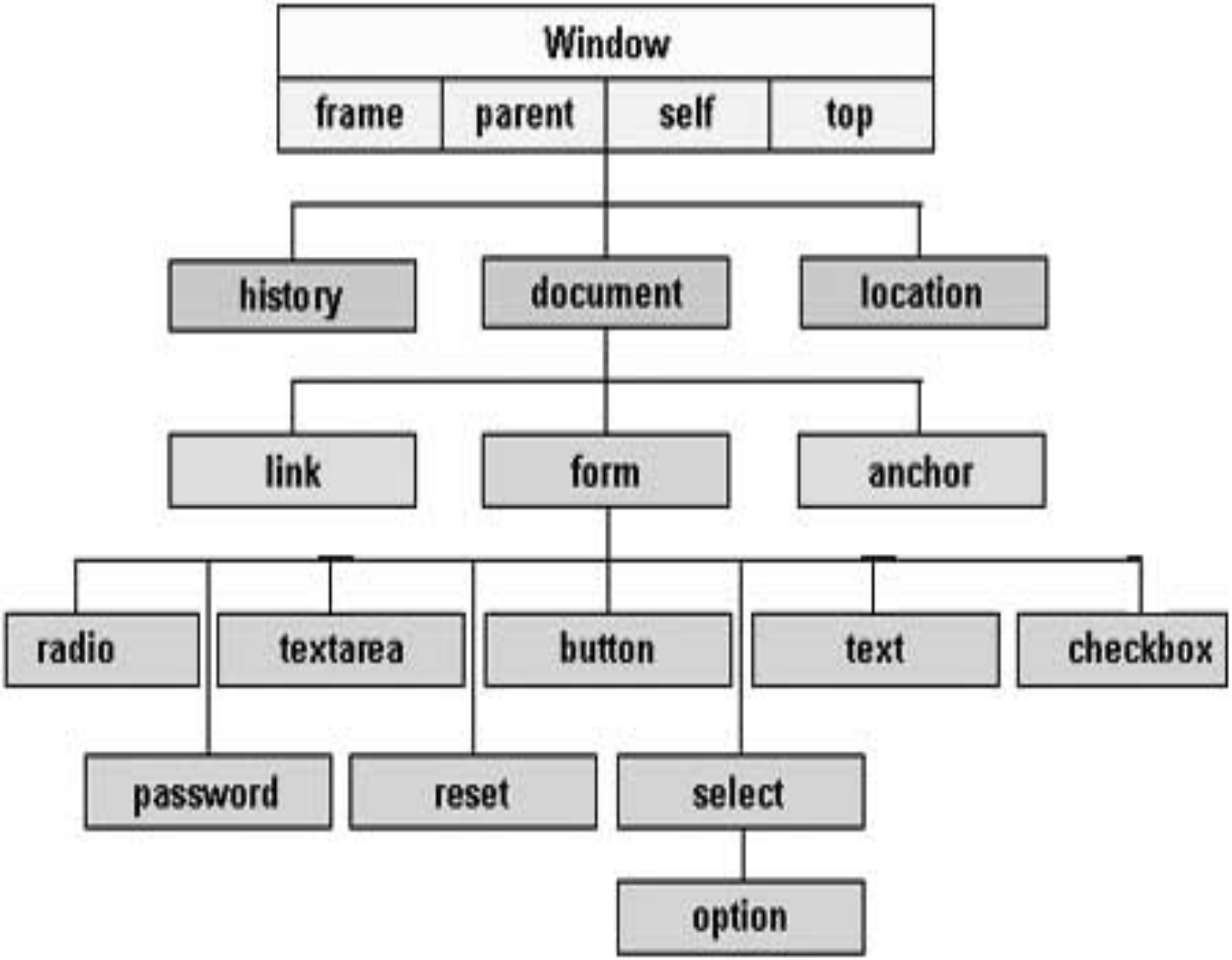
JS-Part2

Document object model

- Every web page resides inside a browser window which can be considered as an object.
- A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.
- The way that document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.
- **Window object:** Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object:** Each HTML document that gets loaded into a window becomes a document object. The document contains the content of the page.
- **Form object:** Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements:** The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.


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

- The **window** object is supported by all browsers. It represent the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.
- Even the document object (of the HTML DOM) is a property of the window object:

Window Object Properties

- document: Returns the Document object for the window.
- history: Returns the History object for the window.
- location: Returns the Location object for the window.
- navigator: Returns the Navigator object for the window

The Document Object

- When an HTML document is loaded into a web browser, it becomes a **document object**.
- The document object is the root node of the HTML document and the "owner" of all other nodes:
  (element nodes, text nodes, attribute nodes, and comment nodes).

History Object

- The history object contains the URLs visited by the user (within a browser window).
- The history object is part of the window object and is accessed through the window.history property.

Location Object

- The location object contains information about the current URL.
- The location object is part of the window object and is accessed through the window.location property.

Navigator Object

- The navigator object contains information about the browser.
- appCodeName:Returns the code name of the browser
- appVersion:Returns the version information of the browser
- Geolocation: Returns a Geolocation object that can be used to locate the user's position
- Language: Returns the language of the browser

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. The **HTML DOM** model is constructed as a tree of **Objects.**
- *The 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 HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements
- In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

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

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

- <h1>My First Page</h1>

- <p id=“raman"></p>

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

- </body>
- </html>

- getElementById is a **method**, while innerHTML is a **property**.

The HTML DOM Document

- In the HTML DOM object model, the document object represents your web page.
- The document object is the owner of all other objects in your web page.
- If you want to access objects in an HTML page, you always start with accessing the document object.

Finding HTML Elements

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

- <!DOCTYPE html>
- <html>
- <body>
- cdac
- <p id=“raman">
- <script>
- myElement = document.getElementById(“ashoka");
- document.getElementById(“raman").innerHTML =
- "The text from the paragraph is " + myElement.innerHTML;
- </script>

- </body>
- </html>

- var x = document.getElementById(“ashoka");
- If the element is found, the method will return the element as an object (in x).
- If the element is not found, x will contain null.
- var x =document.getElementsByTagName("p");
- This example finds all <p> elements:
- var x = document.getElementById("main");
  var y = x.getElementsByTagName("p");
- This example finds the element with id="main", and then finds all <p> elements inside "main":
- var x = document.getElementsByClassName("intro");
- This example returns a list of all elements with class="intro".

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

  <h1 id=“raman">Old Header</h1>

  <script>
  var element = document.getElementById(“raman");
  element.innerHTML = "New Header";
  </script>

  </body>
  </html>

Changing the Value of an Attribute

- document.getElementById(*id*).*attribute=new value*
- ```
  <!DOCTYPE html>
  <html>
  <body>

  <img id="myImage" src="smiley.gif">

  <script>
  document.getElementById("myImage").src = "landscape.jpg";
  </script>

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

- document.getElementById(*id*).style.*property=new style*
- ```
  <html>
  <body>

  <p id=“raman">Hello World!

  <script>
  document.getElementById(“raman").style.color = "blue";
  </script>

  The paragraph above was changed by a script.

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

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

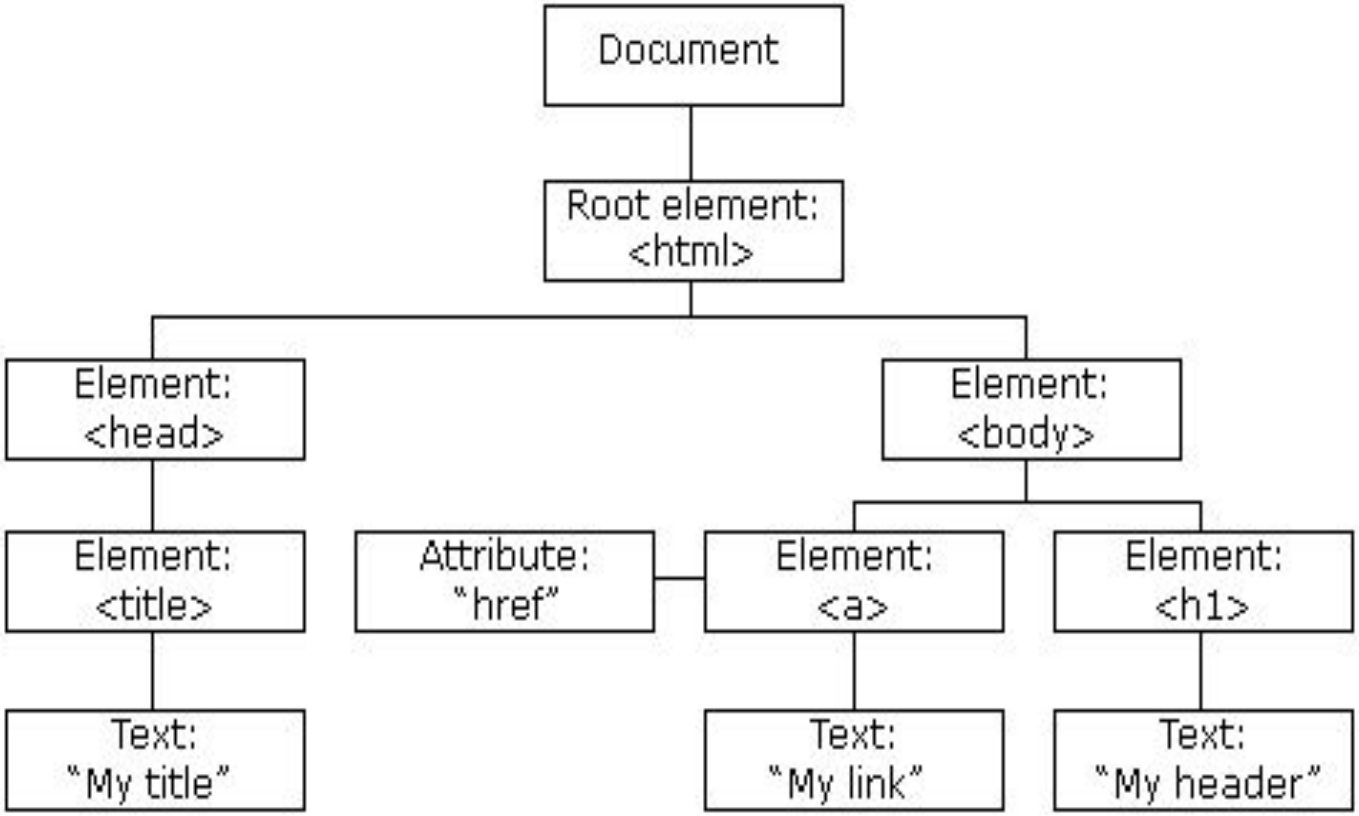
  <h1 id=“raman">My Heading 1</h1>

  <button type="button"
  onclick="document.getElementById(‘raman').style.col
  or = 'red'">
  Click Me!</button>

  </body>
  </html>

DOM Navigation

- you can navigate the node tree using node relationships.
- everything in an HTML document is a node:
- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node
- All comments are comment nodes


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"

Node Relationships

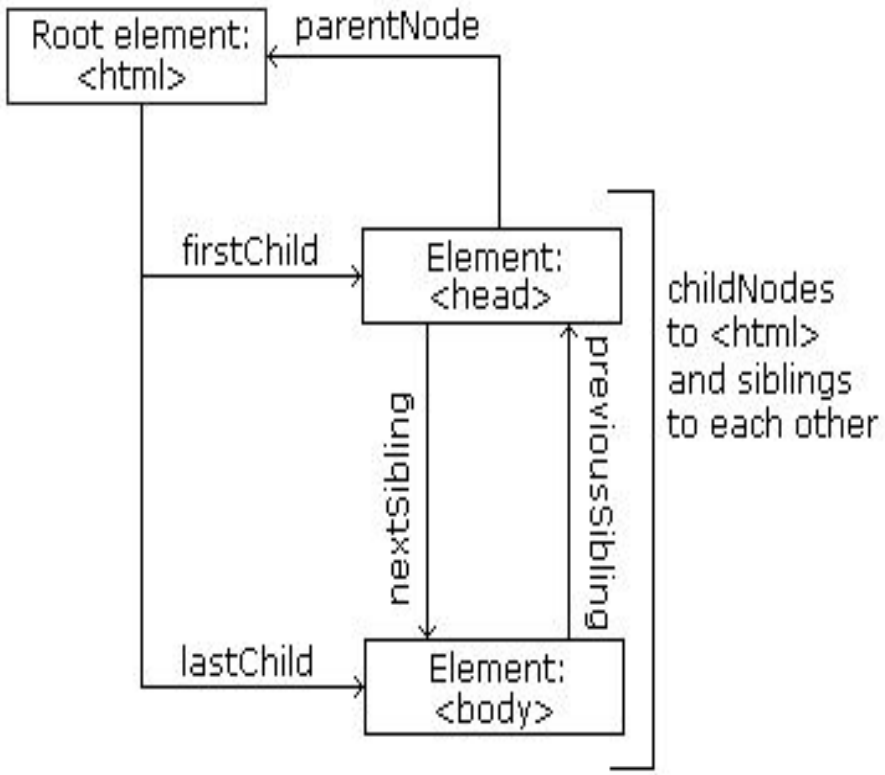
- The nodes in the node tree have a hierarchical relationship to each other.
- The terms parent, child, and sibling are used to describe the relationships.
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<html>

 <head>
  <title>cdac</title>
 </head>


 <body>
  <h1>heading1</h1>
  <p>Hello world!</p>
 </body>

</html>
```



- From the HTML above you can read:
- <html> is the root node
- <html> has no parents
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>
- and:
- <head> has one child: <title>
- <title> has one child (a text node): “cdac"
- <body> has two children: <h1> and <p>
- <h1> has one child: “heading1"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings
-

Navigating Between Nodes

- You can use the following node properties to navigate between nodes with JavaScript:
- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling
- **<title>cdac</title>**, the element node <title> does not contain text. It contains a **text node** with the value "**cdac**".
- The value of the text node can be accessed by the node's **innerHTML** property, or the **nodeValue**.
-

Child Nodes and Node Values

- In addition to the innerHTML property, you can also use the childNodes and nodeValue properties to get the content of an element.
- The following example collects the node value of an <h1> element and copies it into a <p> element:

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

- <h1 id=“ashoka">My First Page</h1>

- <script>
- var myText = document.getElementById(“ashoka").childNodes[0].nodeValue;
- document.write(myText);
- </script>

- </body>
- </html>
- Using the firstChild property is the same as using childNodes[0]:

- **The nodeName Property**
- The nodeName property specifies the name of a node.
- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

- **The nodeValue Property**
- The nodeValue property specifies the value of a node.
- nodeValue for element nodes is undefined
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value
- <title> cdac </title>
- Element node-textnode--textvalue

- **The nodeType Property**
- The nodeType property returns the type of node. nodeType is read only.
- The most important node types are:

Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Creating New HTML Elements (Nodes)

- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<!DOCTYPE html>
<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);    # this is new---new paragraph
var element = document.getElementById("div1");
element.appendChild(para);
</script>

</body>
</html>
```

- This is a paragraph.
- This is another paragraph.
- This is new.

Example explained

- This code creates a new <p> element:
- var para = document.createElement("p");
- To add text to the <p> element, you must create a text node first. This code creates a text node:
- var node = document.createTextNode("This is a new paragraph.");
- Then you must append the text node to the <p> element:
- para.appendChild(node);
- Finally you must append the new element to an existing element.
- This code finds an existing element:
- var element = document.getElementById("div1");
- This code appends the new element to the existing element:
- element.appendChild(para);
-

Creating new HTML Elements - insertBefore()

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

-
- This is a paragraph.
- This is another paragraph.
-

- <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>

- This is new.
- This is a paragraph.
- This is another paragraph.

Removing Existing HTML Elements

- To remove an HTML element, you must know the parent of the element:

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

output

- This is another paragraph.

Example Explained

- This HTML document contains a element with two child nodes (two elements):
-
  This is a paragraph.
  This is another paragraph.

- Find the element with id="div1":
- var parent = document.getElementById("div1");
- Find the element with id="p1":
- var child = document.getElementById("p1");
- Remove the child from the parent:
- parent.removeChild(child);
-

Replacing HTML Elements

- To replace an element to the HTML DOM, use the replaceChild() method <!DOCTYPE html>

- <html>
- <body>

-
- This is a paragraph.
- This is another paragraph.
-

- <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>

HTML DOM Node List

- The getElementsByTagName() method returns a **node list**. A node list is an array-like collection of nodes.
- The following code selects all <p> nodes in a document:
- Example
- var x = document.getElementsByTagName("p");
- x= {p1,p2,p3,p4,p5}
- 0,1,2,3,4
- X[1]
- The nodes can be accessed by an index number. To access the second <p> node you can write:
- y = x[1];(indexing starts at 0)

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

- Hello World!

- CDAC noida

- <p id=“raman">

- <script>
- var myNodelist = document.getElementsByTagName("p"); p1,p2,p3
- document.getElementById(“raman").innerHTML =
- "The innerHTML of the second paragraph is: " +
- myNodelist[1].innerHTML; CDAC noida p2 content
- </script>

- </body>
- </html>
- myNodelist.length defines the number of nodes in a node list

NaN Property

- "Not-a-Number" value. This property indicates that a value is not a legal number.
- isNaN() : function determines whether a value is an illegal number (Not-a-Number).
- This function returns true if the value is NaN, and false if not.

- var a = isNaN(123) + "
"; =false
- var b = isNaN(-1.23) + "
"; == false
- var c = isNaN(5-2) + "
"; =false
- var d = isNaN(0) + "
"; =false
- var e = isNaN("Hello") + "
"; =true
- var f = isNaN("2005/12/12") + "
"; =true
- document.write(a+b+c+d+e+f);

isFinite()

- The isFinite() function determines whether a number is a finite, legal number.
- This function returns false if the value is +infinity, -infinity, or NaN (Not-a-Number), otherwise it returns true.

- var a = isFinite(123) + "
";
- var b = isFinite (-1.23) + "
";
- var c = isFinite (5-2) + "
";
- var d = isFinite (0) + "
";
- var e = isFinite ("Hello") + "
";
- var f = isFinite ("2005/12/12") + "
";
- document.write(a+b+c+d+e+f);

Number()

- The Number() function converts the object argument to a number that represents the object's value.
- If the value cannot be converted to a legal number, NaN is returned.
- **Note:** If the parameter is a Date object, the Number() function returns the number of milliseconds since midnight January 1, 1970 UTC.

example

- var x1 = true; = 1
  var x2 = false; = 0
  var x3 = new Date(); = number of milliseconds

  var x4 = "999 888"; =Nan

  var n =
  Number(x1) + "
" +
  Number(x2) + "
" +
  Number(x3) + "
" +
  Number(x4);
- document.write(n);

output

- 1
  0
  1427012402858
  NaN

Parseint

- The parseInt method parses a value as a string and returns the first integer.
- If the first character cannot be converted, NaN is returned.
- Only the first integer found is returned.

Example

- parseInt(“30") = 30
- parseInt(“20.00") = 20
- parseInt(“20.33") = 20
- parseInt(“43 45 46") = 43
- parseInt(" 50 “) = 50
- parseInt(“29 cars") = 29
- parseInt(“I am 30"); =nan

String()

- The String() function converts the value of an object to a string.
- **Note:** The String() function returns the same value as toString() of the individual objects.

- var x1 = Boolean(0); false
  var x2 = Boolean(1); true
  var x3 = new Date();
  var x4 = "12345"; 12345
  var x5 = 12345; 12345

  var res =
  String(x1) + "
" +
  String(x2) + "
" +
  String(x3) + "
" +
  String(x4) + "
" +
  String(x5);
- document.write(res);

- false
  true
  current date will be shown
  12345
  12345

JavaScript Strings

- stores a series of characters.
- used for storing and manipulating text.
- var personname = “suman gupta";
  var personname = ' suman gupta ';
- String Length:Text.length

String Methods

- indexOf():returns the index of (the position of) the **first** occurrence of a specified text in a string:
- var str = “Rain Rain go away”;
  var pos = str.indexOf(“Rain“,0);
- lastIndexOf():returns the index of the **last** occurrence of a specified text in a string:
- var str = “Rain Rain go away;
  var pos = str. lastIndexOf(“Rain");

- **search():**searches a string for a specified value and returns the position of the match:
- var str = “Rain Rain go away;
  var pos = str.search(“Rain");

- Extracting string parts:
- slice(start, end)
- substr(start, length)

- slice():extracts a part of a string and returns the extracted part in a new string.
- var str = “Rain Rain go away”;
- var res = str.slice(5,13);
- slice(starting index, lastindex)

- substr():similar to slice().
- The difference is that the second parameter specifies the **length** of the extracted part.

- var res = str.substr(7,5); (starting index,lenth of substring to be fetched)

- **replace()** : replaces a specified value with another value in a string:
- str = “I play badminton";
  var n = str.replace(" badminton ",“table tennis");
- toUpperCase():
- var text2 = str.toUpperCase();
- toLowerCase():
- concat():

- var text1 = “first";
  var text2 = “last";
  text3 = text1.concat(" ",text2);
- var text = “first" + " " + ”last";
  var text = “first".concat(" ",“last");
- These two lines are same.

- charAt() method returns the character at a specified index (position) in a string:
- Example
- var str = "HELLO WORLD";
  str.charAt(0);

- **Reacting to events**

The oninput Event

- The oninput event are often used in combination with validation of input fields.The upperCase() function will be called when a user changes the content of an input field.
- <!DOCTYPE html>
- <html>
- <head>
- <script>
- function myFunction() {
- var x = document.getElementById("fname");
- x.value = x.value.toUpperCase();
- }
- </script>
- </head>

- <body>

- Enter your name: <input type="text" id="fname" oninput="myFunction()">
- When you leave the input field, a function is triggered which transforms the input text to upper case.

- </body>
- </html>

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

<div id=“one” onmouseover="mOver()" onmouseout="mOut()"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver() {
    document.getElementById(“one”).innerHTML = "Thank You"
}

function mOut() {
   document.getElementById(“one”).innerHTML = “Bye everyone"
}
</script>

</body>
</html>
```

The addEventListener() method

- The addEventListener() method attaches an event handler to the specified element.
- Syntax
- *element*.addEventListener(*event, function*);

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

<p>This example uses the addEventListener() method to add two click events to the same button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
    alert ("Hello World!");
}

function someOtherFunction() {
    alert ("This function was also executed!");
}
</script>

</body>
</html>
```

- The removeEventListener() method
- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:
- Example
- *element*.removeEventListener("mousemove", myFunction);