CSE 4004: Web Technologies

MODULE 2: JavaScript, XML, JSON and Ajax

Faculty: Prof. Daphna Chacko,
Assistant Professor, SCOPE

Java and JavaScript

- JavaScript was developed by Brendan Eich in 1990 at Netscape
- Java was developed by James Gosling and team at Sun Microsystems in 1995
- JavaScript is mainly used to make web pages more interactive
- Java is typically used for all server side development, while JavaScript is reserved for developing client side scripts for functions like validation and interactivity.
- Java code must be compiled, and JavaScript code is just text.
- JavaScript code can be executed in a browser, while Java creates applications that run in a virtual machine or browser
- Using JavaScript HTML elements could be displayed/ hidden, attribute values and CSS styling can also be changed

Where to insert JavaScript code?

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both
- There can be multiple script elements in an HTML page
- Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.
- Scripts can also be placed in external files with '.js' extension
- To use an external script, the name of the script file is placed in the src (source) attribute of a <script> tag
- An external script can also be referenced in head or body of an HTML page

Displaying data

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- Once the element is accessed the data that it displays can be changed using *innerHTML* property.
- *document.write()*is used to write data into HTML output. It is used for testing purposes.
- document.write() will overwrite the data of an HTML page if it is invoked after the page is loaded.
- *window.alert()*is used to display data in an alert box
- *console.log()*method is used to display data for debugging purposes.
- window.print() method can be used to print the content of the current window.

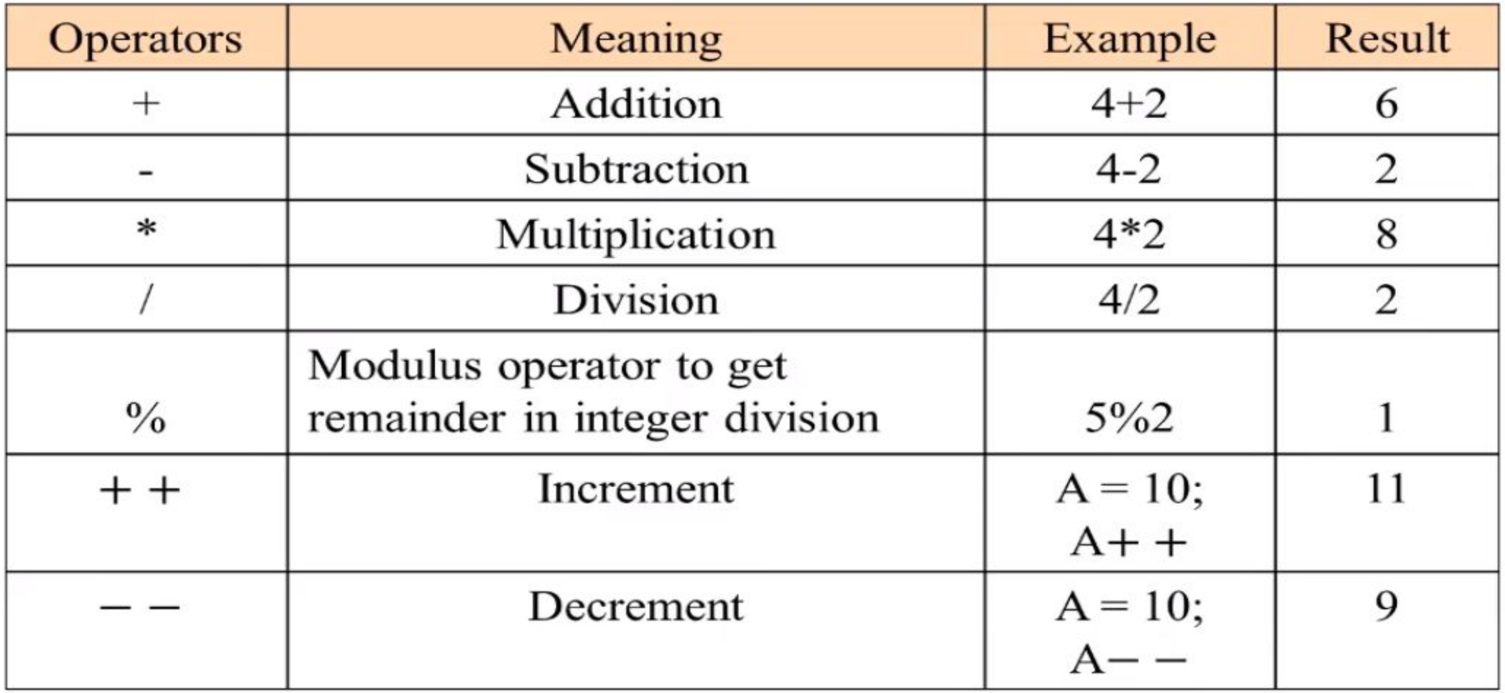
Outline of a JavaScript program

- JavaScript programs are executed by the web browser
- JavaScript statements are composed of:
  - Values, Operators, Expressions, Keywords, and Comments.
- Semicolons separate JavaScript statements.
- The statements are executed, one by one, in the same order as they are written
- JavaScript ignores multiple spaces.
- JavaScript statements can be grouped together in code blocks, inside curly brackets
- There are 2 types of values defined by JavaScript: variables and literals
  - Literals can be integers or string
  - *var* keyword is used to declare a variable
  - An equal sign is used to assign values to variables.
- JavaScript uses arithmetic operators (+ - * /) to compute values
- An expression is a combination of values, variables, and operators, which computes to a value
- Code after double slashes // or between /* and */ is treated as a comment.
- Identifiers are used to name variables, functions, keywords and labels. They are case sensitive
  - In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).
  - Subsequent characters may be letters, digits, underscores, or dollar signs

Variables and datatypes

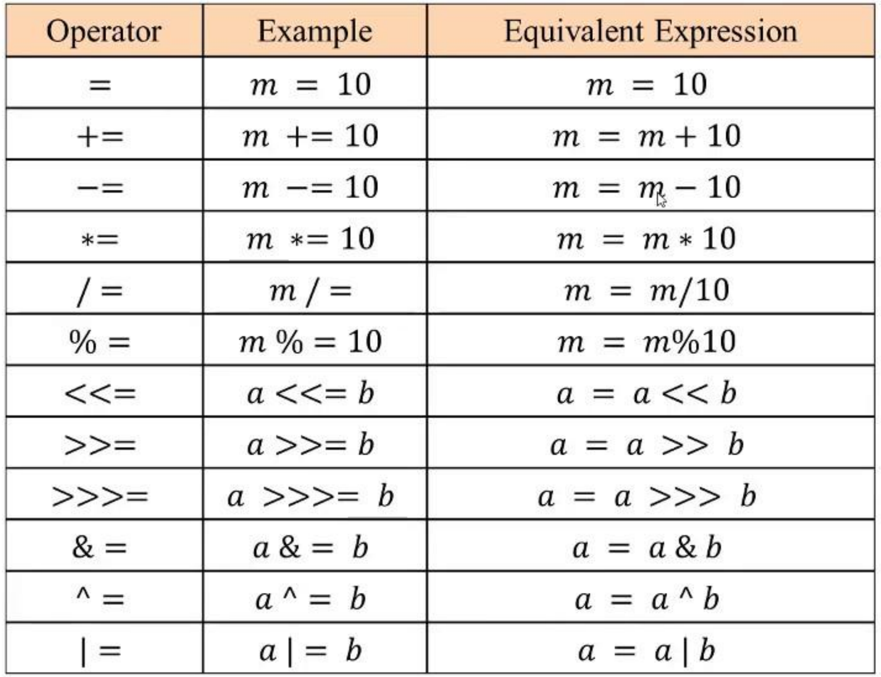
- Variables in JavaScript are dynamically typed, meaning a variable can be an integer, and then later a string, then later an object.
- JavaScript variables can hold numbers like 100 and text values like "Hello".
- Strings are written inside double or single quotes. Numbers are written without quotes.
- Many variables can be declared in one statement separated by commas
- If you re-declare a JavaScript variable, it will not lose its value
- Assignment can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment
- JavaScript evaluates expressions from left to right. Different sequences can produce different results
- The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop.
- When adding a number and a string, JavaScript will treat the number as a string.

Arithmetic Operators



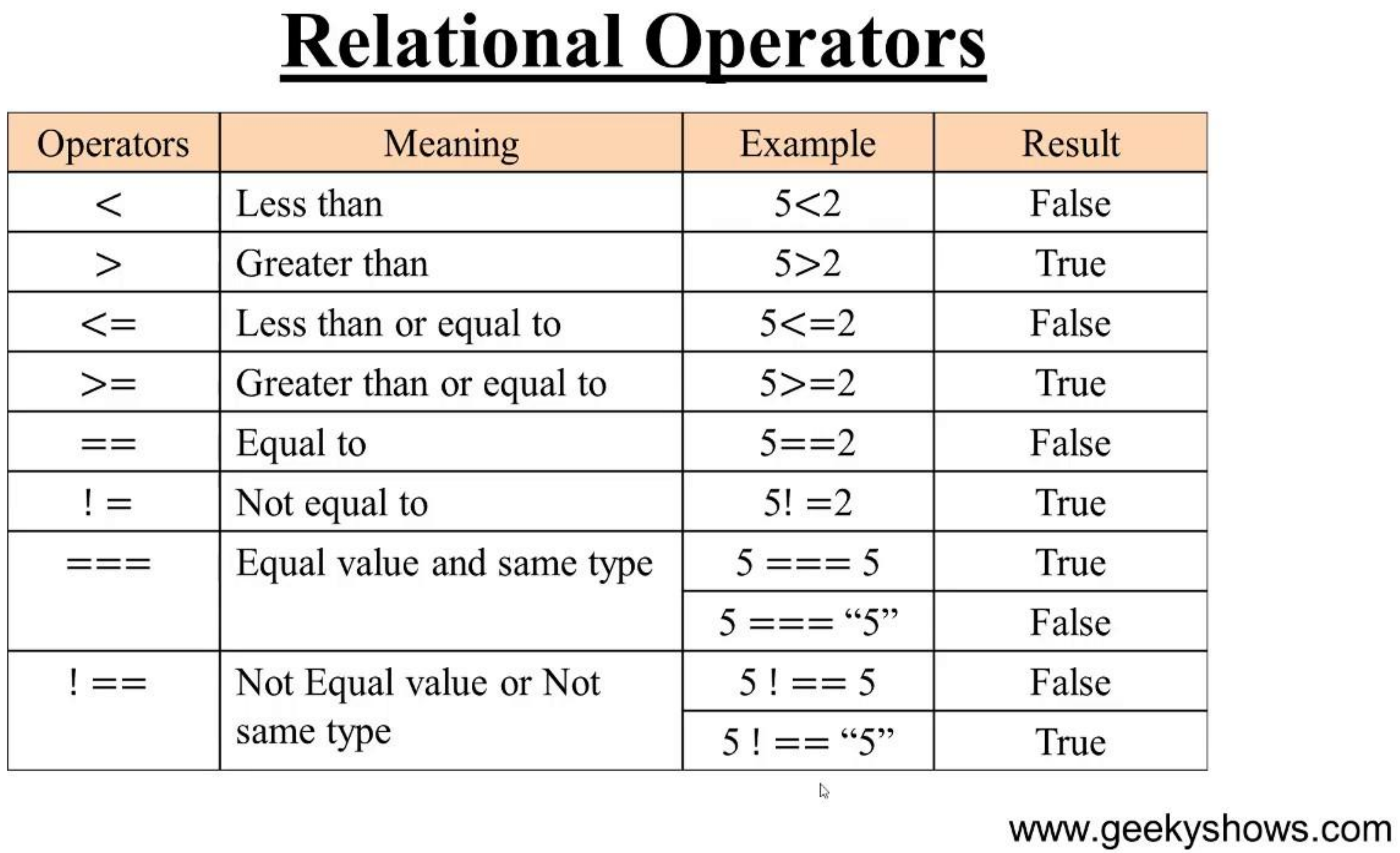
Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
− −	Decrement	A = 10; A− −	9

Assignment Operators



Operator	Example	Equivalent Expression
$=$	$m = 10$	$m = 10$
$+=$	$m += 10$	$m = m + 10$
$-=$	$m -= 10$	$m = m - 10$
$*=$	$m *= 10$	$m = m * 10$
$/=$	$m /=$	$m = m/10$
$\% =$	$m \% = 10$	$m = m\%10$
$<<=$	$a <<= b$	$a = a << b$
$>>=$	$a >>= b$	$a = a >> b$
$>>>=$	$a >>>= b$	$a = a >>> b$
$\& =$	$a \& = b$	$a = a \& b$
$\hat{} =$	$a \hat{} = b$	$a = a \hat{} b$
$\vert =$	$a \vert = b$	$a = a \vert b$

www.geekyshows.com

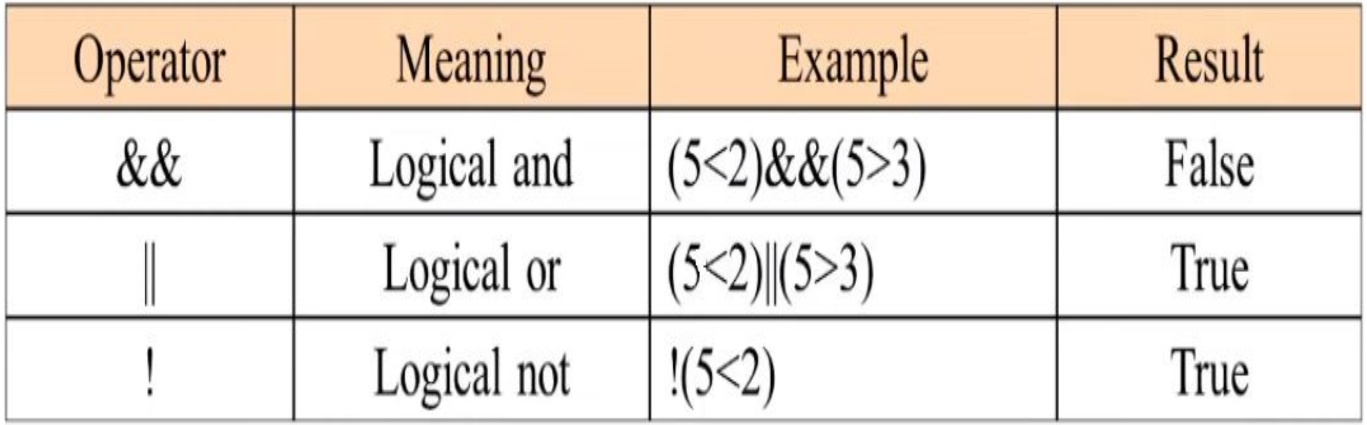


Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === “5”	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == “5”	True

www.geekyshows.com

Logical Operators



Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

Conditional and iterative statements

- Similar to conditional structures such as if and if else statements in C.
- The condition to test is contained within ( ) brackets with the body contained in { } blocks.
- Iteration is done using while and for loops.
- Loops use the ( ) and { } blocks to define the condition and the body of the loop.
  - while loops normally initialize a loop control variable before the loop, use it in the condition, and modify it within the loop.
  - A for loop combines the common components of a loop: initialization, condition, and post-loop operation into one statement.

Objects

- Though JavaScript is not a full fledged object oriented language, it do support objects
- Objects can be in built or user defined and can have constructors, properties, and methods associated with them.
- Each object might have properties that can be accessed using dot (.) operator
- Objects can also have methods, which are functions associated with an instance of an object.
- Some of the objects that are included in JavaScript are:
  - Array
  - Boolean
  - Date
  - Math
  - String
  - Dom objects
- In JavaScript variables are containers for data values. Objects are variables that can contain many values.
- The values are written as name:value pairs
  - var student = {regno:7184, slot: “XX”, univ:”VIT_AP”};
- When a JavaScript variable is declared with the keyword "new", the variable is created as an object,
- Methods are stored in properties as function definitions.
  - var student = {regno:7184, slot: “XX”, univ:”VIT_AP”, display:function(){return this.regno+” “+this.slot}};
- In a function definition,’ this’ refers to the "owner" of the function.

Dealing with functions and arrays

- A JavaScript function can be written to handle a particular event
- Functions are defined by using the reserved word *function* and then the function name and (optional) parameters.
- Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type.
- JavaScript arrays are used to store multiple values in a single variable. Array indexes start with 0
- Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays
- The following code creates a new, empty array named greetings:
    - var greetings = new Array(); or var greetings = [“Hello”,”Hai”,100];
- The length property of an array returns the number of array elements
- The easiest way to add a new element to an array is using the push() method. The **pop** method can be used to remove an item from the back of an array.
- Additional methods: concat(), slice(), join(), reverse(), shift(), and sort()
    - var alpha = ["a", "b", "c"];
    - var numeric = [1, 2, 3];
    - var alphaNumeric = alpha.concat(numeric); o/p: a,b,c,1,2,3
    - alphaNumeric.slice(1, 3); o/p: b, c
    - alphaNumeric.reverse(); o/p: 3,2,1,c,b,a

Strings

- To find the length of a string, use the built-in length property
- The backslash (\) escape character turns special characters into string characters
  - The sequence \" inserts a double quote in a string
- Strings can also be defined as objects with the keyword new:
  - var firstName = new String("John");
- *charAt*(): Returns the character at a specified index (position) in a string
- *charCodeAt*() method returns the unicode of the character at a specified index in a string
- *indexOf*(): returns the index of (the position of) the first occurrence of a specified text in a string:
- *slice*() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the start position, and the end position
- The *replace*() method replaces a specified value with another value in a string. It returns a new string
- The *concat*() method can be used instead of the plus operator to join two or more strings

Math

- Math object allows you to perform mathematical tasks on numbers.
- The Math object is static. All methods and properties can be used without creating a Math object first.
- Following are the predefined mathematical constants:
  - Math.E // returns Euler's number
  - Math.PI // returns PI
  - Math.SQRT2 // returns the square root of 2
  - Math.SQRT1_2 // returns the square root of 1/2
  - Math.LN2 // returns the natural logarithm of 2
  - Math.LN10 // returns the natural logarithm of 10
  - Math.LOG2E // returns base 2 logarithm of E
  - Math.LOG10E // returns base 10 logarithm of E
- Following are the predefined mathematical functions:
  - Math.round(x) : Returns x rounded to its nearest integer
  - Math.ceil(x) : Returns x rounded up to its nearest integer
  - Math.floor(x) : Returns x rounded down to its nearest integer
  - Math.pow(x, y) returns the value of x to the power of y:
  - Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments
    - Math.min(0, 150, 30, 20, -8, -200);

Date

- Used to calculate the current date or create date objects for particular dates
- Date objects are static and does not change with time.
- Date objects are created with the new Date() constructor.
- JavaScript counts months from 0 to 11
- JavaScript will (by default) output dates in full text string format
- When you display a date object in HTML, it is automatically converted to a string, with the toString() method

Window

- The window object in JavaScript corresponds to the browser itself.
- Using window object we can access:
  - The current page's URL
  - The browser's history
  - Content displayed in the status bar
  - Opening new browser windows
- alert() and print() are methods of window object

Defining Scope

- The 2015 version of JavaScript (ES6) allows the use of the const keyword to define a variable that cannot be reassigned, and the let keyword to define a variable with restricted scope
- Before ES2015, JavaScript had only two types of scope: Global Scope and Function Scope
    - Variables declared Globally (outside any function) have Global Scope
    - Variables declared Locally (inside a function) have Function Scope
- Variables declared with the let keyword can have Block Scope.
- Variables declared inside a block {} cannot be accessed from outside the block
- Global variables defined with the var keyword belong to the window object whereas variables defined with let keyword do not.
- Variables defined with const behave like let variables, except they cannot be reassigned
- JavaScript const variables must be assigned a value when they are declared
- We cannot change constant primitive values, but we can change the properties of constant objects.

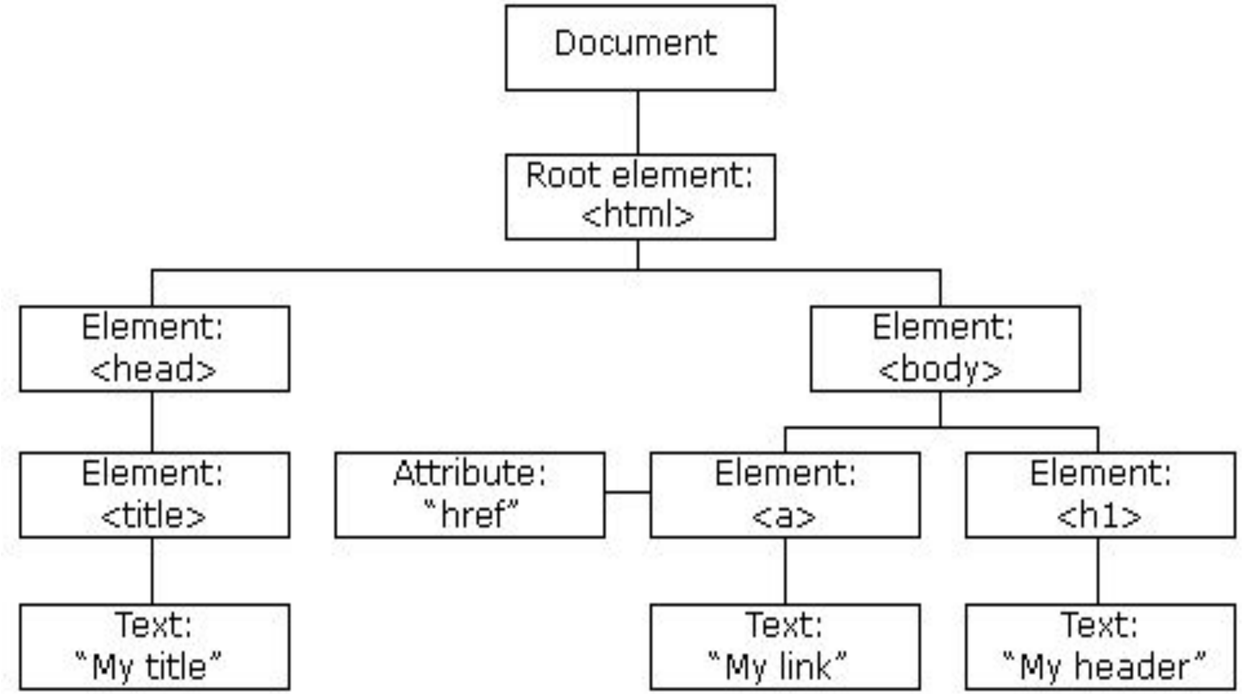
THE DOCUMENT OBJECT MODEL (DOM)

Introduction

- Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents
- The HTML DOM model is constructed as a tree of Objects
- It 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
- 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

An example

```
<!DOCTYPE html>
<head>
<title>
My  title
</title>
</head>
<html>
<body>
<a href=#href>My link</a>
<h1>My header</h1>
</body>
</html>
```



DOM nodes

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- In the DOM, all HTML elements are defined as objects
- The HTML DOM document object is the owner of all other objects in a web page.
- In the DOM, each element within the HTML document is called a **node.** If the DOM is a tree, then each node is an individual branch.
- There are:
  - element nodes,
  - text nodes, and
  - attribute nodes
- All nodes in the DOM share a common set of properties and methods

DOM elements

- To manipulate HTML elements, we have to find the elements first. There are several ways to do this:
    - Finding HTML elements by id (getElementById)
    - Finding HTML elements by tag name (document.getElementsByTagName("p");)
    - Finding HTML elements by class name (getElementsByClassName("intro");)
    - Finding HTML elements by CSS selectors such as id, class names, types, attributes, values of attributes, etc (document.querySelectorAll("p.intro");)
    - Finding HTML elements by HTML object collections like forms
- To change the value of an HTML attribute, use this syntax:
    - document.getElementById(*id*).*attribute = new value*
- To change the style of an HTML element, use this syntax:
    - document.getElementById(*id*).style.*property = new style*

JAVASCRIPT EVENTS

Introduction

- An HTML event can be something the browser does, or something a user does.
- JavaScript lets you execute code when events are detected.
- We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response
- onchange, onclick, onload and many more are events that may occur in a webpage.
- Event handlers can be used to handle and verify user input, user actions, and browser actions
    - Events could be specified right in the HTML markup with *hooks* to the JavaScript code
    - Listener approach is a refined technique to handle events
- Events can be considered as DOM event objects and event handlers are functions of these objects
    - function someHandler(**e**) {
      // e is the event that triggered this handler.
      }

Event types

There are several classes of event, with several types of events within each class:

- Mouse events: *onclick, onmousedown*
- Keyboard events: *onkeypress*
- Form events: *onsubmit, onfocus*
- Frame events: *onload, onresize*
- The *onload* and *onunload* events are triggered when the user enters or leaves the page
- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers
    - *element*.addEventListener(*event, function, useCapture*);.
- You can add many event handlers to one element.

Event propagation

- There are two ways of event propagation in the HTML DOM, *bubbling and capturing*.
- Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?
- In *bubbling* the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.
- In *capturing* the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.
- With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:
- addEventListener(*event*, *function*, *useCapture*);
- The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

Form Validation

- HTML form validation can be done by JavaScript.
- Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load.
- There are a number of common validation activities including
  - email validation
  - number validation
  - data validation.
- Using onsubmit event, validations can be done at the time of form submission
- HTML form validation can also be performed automatically by the browser (eg required attribute)

EXTENSIBLE MARKUP LANGUAGE (XML)

Introduction

- Designed to store and transport data.
- XML is a markup language like HTML
- In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.
- Most XML applications will work as expected even if new data is added (or removed)
- XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.
- Information is wrapped using tags in XML
- XML tags are NOT predefined like HTML tags

XML document structure

- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- All elements can have text content and attributes.
- A prolog defines the XML version and the character encoding. The prolog is optional and if exists it must come first in the document
    - `<?xml version="1.0" encoding="UTF-8"?>`
- The next line is the root element of the document that is the parent of all other elements
- All elements MUST HAVE a closing tag.
- XML tags are case sensitive. Opening and closing tags must be written with the same case
- XML elements can have attributes in name/value pairs just like in HTML and it must always be quoted

Entity references, comments and whitespaces

- Some characters have a special meaning in XML
- To use them inside elements we use entity references. Following are the 5 entity references in XML.

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

- The syntax for writing comments in XML is similar to that of HTML
- XML does not truncate multiple white-spaces and stores a new line as LF

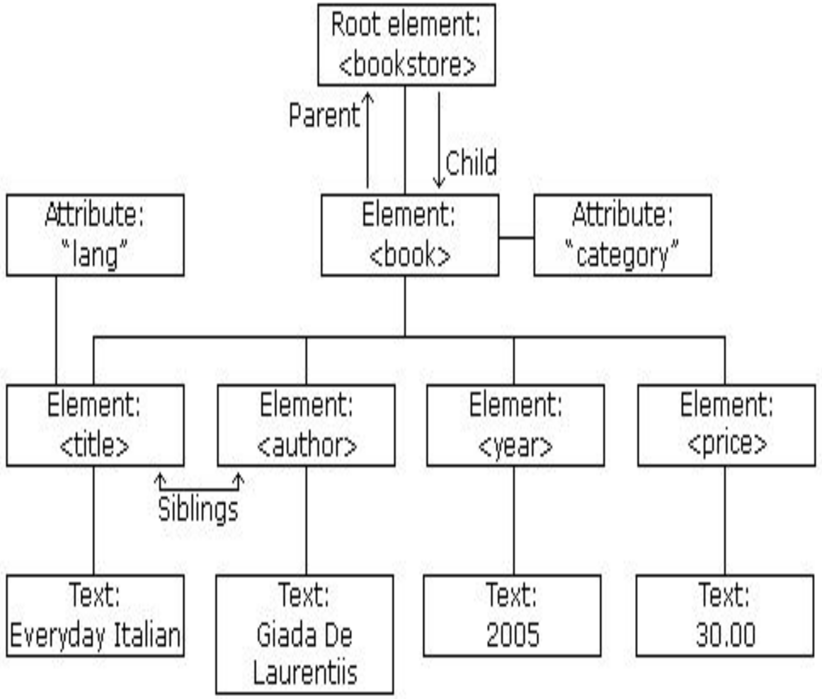
XML elements

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain:
    - text
    - attributes
    - other elements
    - or a mix of the above
- An element with no content is said to be empty. It can be indicated as <element></element> and <element />
- Empty elements can have attributes.
- Following are the naming rules for elements
    - Element names must start with a letter or underscore
    - Element names cannot start with the letters xml (or XML, or Xml, etc)
    - Element names can contain letters, digits, hyphens, underscores, and periods
    - Element names cannot contain spaces

XML DOM

- The XML DOM defines the properties and methods for accessing and manipulating XML documents.
- It presents an XML document as a tree-structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```



XML Parser

- Before an XML document can be accessed, it must be loaded into an XML DOM object
- All modern browsers have a built-in XML parser that can convert text into an XML DOM object
- An XML DOM parser can be created as:
    - parser = new DOMParser();

```
<html>
<body>
<p id="demo"></p>
<script>
var text, parser, xmlDoc;
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

VIT-AP UNIVERSITY

JAVASCRIPT OBJECT NOTATION (JSON)

Introduction

- JSON is a syntax for storing and exchanging data.
- It is text, written with JavaScript object notation
- we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

    -
      ```
      var myObj = {name: "John", reg no: 7231, branch:"AI"};
      var myJSON = JSON.stringify(myObj);
      ```
- If you receive data in JSON format, you can convert it into a JavaScript object:

    -
      ```
      var myJSON = '{"name":"John", "reg no":7231, "branch":"AI"}';
      var myObj = JSON.parse(myJSON);
      document.getElementById("demo").innerHTML = myObj.name;
      ```

JSON Syntax

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value
- K*eys* (name) must be strings, written with double quotes
- V*alues* must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null
- *String values* must be written with double quotes
- The file type for JSON files is ".json"
- A JavaScript object can be accessed in two ways: dot notation or bracket notation
  - var student= { name: "John", regNo: 7231, branch: "AI" };
  - student.name; or student[name];
- Data can be modified in two ways:
  - student.name=”Ram”;
  - student[name] = “Ram’

Parsing data in JSON

- A common use of JSON is to exchange data to/from a web server. When receiving data from a web server, the data is always a string.

- To convert it to a JavaScript object we use JSON.parse()

  -
    ```
    <script>
    var txt = '{"name":"John", "regNo”:7530, "branch":"DS"}'
    var obj = JSON.parse(txt);
    document.getElementById("demo").innerHTML = obj.name + ", " + obj.regNo;
    </script>
    ```
- When using the JSON.parse() on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

- Functions and Date objects are not allowed in JSON.

- If they need to be included then it should be written as a string and later convert it into appropriate object

  -
    ```
    var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';
    var obj = JSON.parse(text);
    obj.birth = new Date(obj.birth);
    document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
    ```

  -
    ```
    var text = '{ "name":"John", "age":"function () {return 30;}", "city":"New York"}';
    var obj = JSON.parse(text);
    obj.age = eval("(" + obj.age + ")");
    document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
    ```

Converting JavaScript objects into strings

- When sending data to a web server, the data has to be a string.
- To convert a JavaScript object into a string we use JSON.stringify()

  -
    ```
    <script>
    var obj = { name: "John", age: 30, city: "New York" };
    var myJSON = JSON.stringify(obj);
    document.getElementById("demo").innerHTML = myJSON;
    </script>
    ```
- The JSON.stringify() function will remove any functions from a JavaScript object, both the key and the value
- This can be omitted if you convert your functions into strings before running the JSON.stringify() function

  -
    ```
    var obj = { name: "John", age: function () {return 30;}, city: "New York" };
    obj.age = obj.age.toString();
    var myJSON = JSON.stringify(obj);
    document.getElementById("demo").innerHTML = myJSON;
    ```

Nested objects and arrays

- Values in a JSON object/array can be another JSON object/array
  - myObj = {
```
 "name":"John",
 "age":30,
 "cars": {
   "car1":"Ford",
   "car2":"BMW",
   "car3":"Fiat"
 } }
```
- An object can be accessed using the dot notation. Eg, myObj.cars.car2
  - myObj = {
```
 "name":"John",
 "age":30,
 "cars": [
   { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
   { "name":"BMW", "models":[ "320", "X3", "X5" ] },
   { "name":"Fiat", "models":[ "500", "Panda" ] }
 ] }
```
- The array values can be accessed using the index number
- A a for-in loop or for loop can be used to traverse through an array:
  - for (i in myObj.cars) {
```
 x += myObj.cars[i];}
```
- The delete keyword is used to delete items from an object/array

A comparison between XML and JSON

- **Similarities**
  - Can be used to receive data from a web server
  - can be fetched with an XMLHttpRequest
  - can be parsed and used by lots of programming languages
  - can be fetched with an XMLHttpRequest
- **Differences**
  - XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.
  - JSON is shorter and quicker to read and write
  - XML is much more difficult to parse than JSON.
  - JSON is parsed into a ready-to-use JavaScript object.

VIT-AP UNIVERSITY

ASYNCHRONOUS JAVASCRIPT AND XML (AJAX)

Introduction

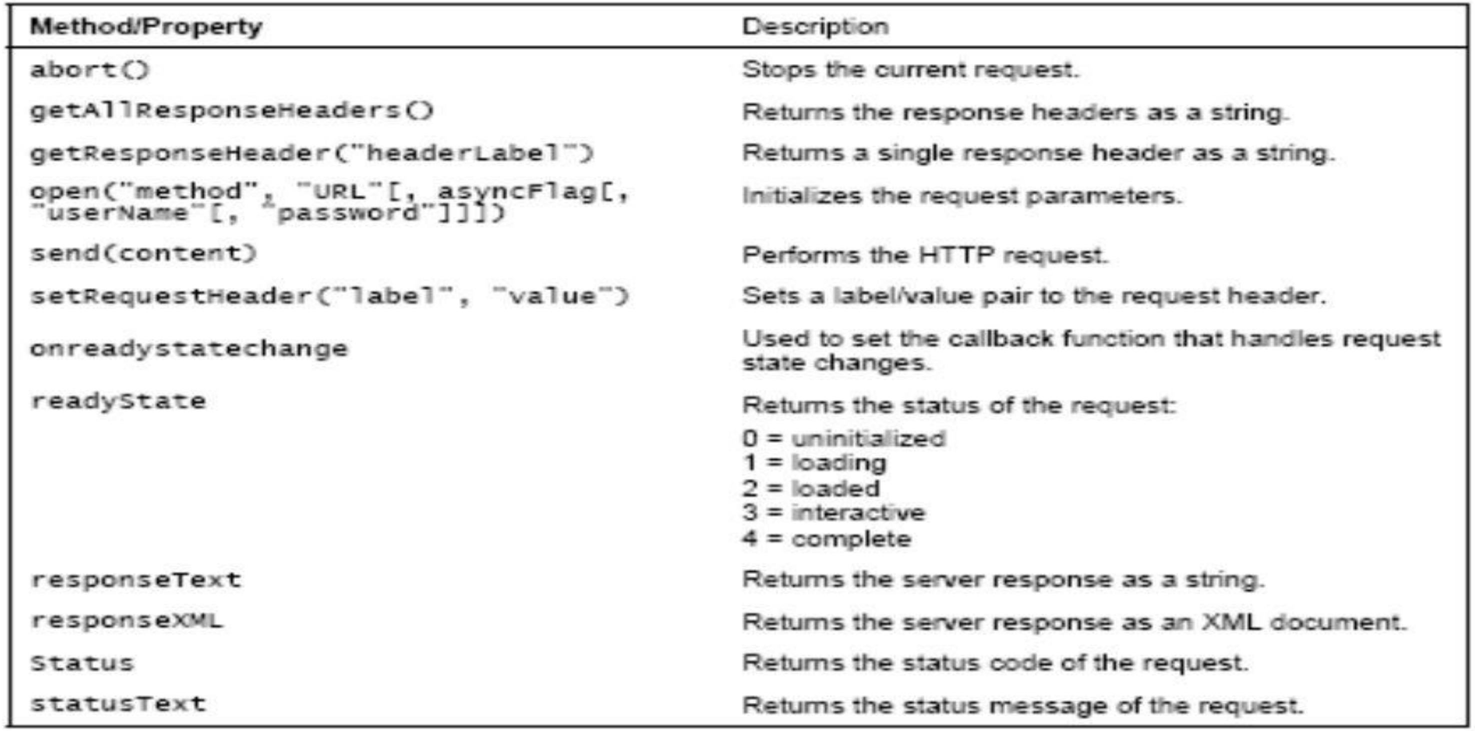
- AJAX is not a programming language.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)
- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes.
- It is possible to update parts of a web page, without reloading the whole page.

XMLHttpRequest

- The XMLHttpRequest object can be used to request data from a web server.
- It helps us to:
  - Update a web page without reloading the page
  - Request data from a server - after the page has loaded
  - Receive data from a server - after the page has loaded
  - Send data to a server - in the background
- The webpage and the XML document that is being loaded must be in the same server

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      // Typical action to be performed when the document is ready:
      document.getElementById("demo").innerHTML = xhttp.responseText;
          }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```

XHR Object’s methods and properties



Method/Property	Description
`abort()`	Stops the current request.
`getAllResponseHeaders()`	Returns the response headers as a string.
`getResponseHeader("headerLabel")`	Returns a single response header as a string.
`open("method", "URL"[, asyncFlag[, "userName"[, "password"]]])`	Initializes the request parameters.
`send(content)`	Performs the HTTP request.
`setRequestHeader("label", "value")`	Sets a label/value pair to the request header.
`onreadystatechange`	Used to set the callback function that handles request state changes.
`readyState`	Returns the status of the request: 0 = uninitialized 1 = loading 2 = loaded 3 = interactive 4 = complete
`responseText`	Returns the server response as a string.
`responseXML`	Returns the server response as an XML document.
`Status`	Returns the status code of the request.
`statusText`	Returns the status message of the request.

AJAX request

- To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object
- The url parameter of the open() method, is an address to a file on a server. The file may be of any type which can perform actions on the server before sending the response back
- GET is simpler and faster than POST, and can be used in most cases.
- POST requests should be used when:
  - A cached file is not an option (update a file or database on the server).
  - Sending a large amount of data to the server (POST has no size limitations).
  - POST is more robust and secure than GET.
- The information to be sent is added to the URL when we use the GET method
- To post data like an HTML form, add an HTTP header with setRequestHeader() and specify the data to be sent in the send() method

Ajax response

- The readyState property holds the status of the XMLHttpRequest.
- The onreadystatechange property defines a function to be executed when the readyState changes.
- The status property and the statusText property holds the status of the XMLHttpRequest object
- The onreadystatechange function is called every time the readyState changes
- The responseText property returns the server response as a JavaScript string
- If you have more than one AJAX task in a website, then a function for executing the XMLHttpRequest object should be created and one callback function for each AJAX task to be performed.
- A callback function is a function passed as a parameter to another function (here, the function creating the XMLHttpRequest object). The URL is also passed.
- The responseXML property returns the server response as an XML DOM object
- The responseText property returns the server response as a JavaScript string

Reading XML file using AJAX

- AJAX can be used for interactive communication with an XML file.

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      myFunction(this);
    }  };
 xhttp.open("GET", "http://127.0.0.1/Subject.xml", true);
 xhttp.send();
}
function myFunction(xml) {
  var i;
  var xmlDoc = xml.responseXML;
  var table="<tr><th>Subject Code</th><th>Subject Name</th></tr>";
  var x = xmlDoc.getElementsByTagName("SUB");
  for (i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("CODE")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("NAME")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
```

References

1. *Fundamentals of web development*, Randy Connolly and Ricardo Hoar.
2. https://raw.githubusercontent.com/DaphnaChacko/Web-Technologies/main/JS.html
3. https://www.w3schools.com/js/default.asp
4. https://raw.githubusercontent.com/DaphnaChacko/Web-Technologies/main/Subject.xml
5. https://raw.githubusercontent.com/DaphnaChacko/Web-Technologies/main/xmlEg.html
6. https://raw.githubusercontent.com/DaphnaChacko/Web-Technologies/main/Credentials.json
7. https://raw.githubusercontent.com/DaphnaChacko/Web-Technologies/main/Match.js