CS3201: Computer Network Technology TY Div A n B AY 2020-21

Study Material for Section-II-Part-II- JavaScript

(Resource: www.w3schools.com)

Learning JavaScript

JavaScript is the programming language of HTML and the Web.

Is JAVA and JavaScript same?

JavaScript and Java are completely different languages, both in concept and design.

JavaScript was invented by **Brendan Eich** in 1995, and became an ECMA standard in 1997. ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform.

JavaScript / ECMAScript -

JavaScript was developed for Netscape. Netscape 2 was the first browser to run JavaScript.

After Netscape the Mozilla foundation continued to develop JavaScript for the Firefox browser.

The latest JavaScript version was 1.8.5. (Identical to ECMAScript 5).

ECMAScript was developed by ECMA International after the organization adopted JavaScript.

The first edition of ECMAScript was released in 1997.

This list compares the version numbers of the different products:

Year	JavaScript	ECMA	Browser
1996	1.0		Netscape 2
1997		ECMAScript 1	IE 4
2011		ECMAScript 5	IE 9 (Except "use strict")
2011	1.8.5		Firefox 4 (Except leading zeroes in parseInt)
2015		ECMAScript 2015	Partially Supported in all Browser
2018		ECMAScript 2018	Added rest / spread properties.Asynchronous iteration. Promise.finally(). Additions to RegExp.

Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

What can JavaScript Do? - **Event Handling**

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Content that should be verified when a user inputs data
- Action that should be performed when a user clicks a button
- Things that should be done when a page to be closed
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can call JavaScript functions
- HTML event attributes can execute JavaScript code directly
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

Writing into an HTML element using innerHTML property.

Writing into the HTML output using document.write () method

Writing into an alert box, using window.alert () method

Writing into the browser console, using console.log () method

Printing on printer using window.print () method.

JavaScript Tag

The <script> Tag

In HTML, JavaScript code is inserted between <script> and </script> tags.

<script>

</script>

Points to Remember

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

JavaScript is Case Sensitive

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

Single line comments start with //

Multi-line comments start with /* and end with */.

JavaScript in <head>

```
<head>

<script>
function myFunction()

{
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</head>


<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

```
<html>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

External file: myScript1.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

Example

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

This example uses a script located in a specified folder on the current web site:

Example

```
<script src="/js/myScript1.js"></script>
```

JavaScript HTML method: `getElementById ()` ------ **This is a method**

My first JavaScript Program

```
<html>

<body>

<h2>My First JavaScript</h2>

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

<script>

var msg = "Hello World"

document.getElementById("demo").innerHTML =  msg

</script>

</body>

</html>
```

Note: The `id` attribute defines the HTML element.

The `innerHTML` property defines the HTML content.

Another Way

```
<html>

<body>

<h2>My First JavaScript</h2>

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

<script>
```

```
document.getElementById("demo").innerHTML =  "Hello World"
</script>
</body>
</html>
```

Another Way

```
<html>
<body>
<h2>My First JavaScript</h2>
<script>
document.write("Hello World!");
</script>
</body>
</html>
```

Another Way

```
<html>
<body>
   <script>
   alert( 'Hello, world!' );
  </script>
  </body>
</html>
```

JavaScript Variables and Program for addition of two numbers

```
<html>

<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

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

<script>

var x = 5;

var y = 6;

var z = x + y;

document.getElementById("demo").innerHTML =  "The value of z is: " + z;

</script>

</body>

</html>
```

JavaScript Can Change HTML Content

JavaScript HTML method: **getElementById () ------ This is a method**

```
<html>
```

<body>

<h2>I was Sitaram in India</h2>

I was Sitaram in India.

<button type="button" onclick='document.getElementById("demo").innerHTML = "Now I am Sam in US!"'>Click Me!</button>

</body>

</html>

For using images:

JavaScript Can Change HTML Attribute Values

<body>

<h2>What Can JavaScript Do?</h2>

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

<button onclick="**document.getElementById('myImage').src**='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>

</body>

JavaScript Can Change HTML Styles (CSS)

```
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>

</body>

</html>
```

Another Way

```
<html>

<body>

<p id="demo">Click the button to change the color of this paragraph.</p>

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

<script>

function myFunction() {

  var x = document.getElementById("demo");

  x.style.color = "red";

}

</script>
```

```
</body>

</html>
```

JavaScript Can Hide HTML Elements

```
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button"
onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

</body>
```

JavaScript Can Show HTML Elements

```
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button"
onclick="document.getElementById('demo').style.display='block'">Click Me!</button>

</body>
```

document.write() method

```
<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.

It will overwrite the whole document.</p>

<script>

document.write(5 + 6);

</script>

</body>

</html>
```

Using document.write() after an HTML document is loaded, will delete all existing HTML:

```
<!DOCTYPE html>

<html>

<body>

<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>
```

</body>

</html>

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

20/11/2020

JavaScript Statements

- JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

```
var x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

<script>

var a, b, c;

a = 5; b = 6; c = a + b;

document.getElementById("demo1").innerHTML = c;

</script>

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Manik";
var person="Manik";
```

A good practice is to put spaces around operators (= + - * /):

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

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Ram!";
```

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

Example

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Ram!";
  document.getElementById("demo2").innerHTML = "How are you Shyam?";
}
```

Reserved Keywords

abstract	arguments	await	boolean
break	byte	case	catch
char	class	const	continue
debugger	default	delete	do
double	else	enum	eval
export	extends	false	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface

let	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Numbers are written with or without decimals:

```
10.50
```

```
1001
```

Strings are text, written within double or single quotes:

```
"John Doe"
```

```
'John Doe'
```

JavaScript uses an **assignment operator** (`=`) to **assign** values to variables:

```
var x, y;
x = 5;
y = 6;
```

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes `//` or between `/*` and `*/` is treated as a **comment**.

Comments are ignored, and will not be executed:

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

JavaScript Identifiers

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, **the first character must be a letter**, or **an underscore (_),** or a **dollar sign ($).**

JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables `lastName` and `lastname`, are two different variables:

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

JavaScript and Camel Case

Hyphens:

first-name, last-name, master-card, inter-city.

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

JavaScript Arithmetic Operators

Arithmetic operators perform arithmetic on numbers (literals or variables).

Operator	+ - * ** / % ++ --	
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y

^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

JavaScript Objects

Objects are variables too. But objects can contain many values.

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

Example

```
var person = {

firstName:"John",

lastName:"Doe",

age:50,
```

```
eyeColor:"blue"

};
```

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
return;
}


function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
```

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
```

```
    return this.firstName + " " + this.lastName;
  }
};
```

The **this** Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "`new`", the variable is created as an object:

```
var x = new String();        // Declares x as a String object
var y = new Number();        // Declares y as a Number object
var z = new Boolean();       // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

```
<!DOCTYPE html>
<html>
<body>
<button onclick="this.innerHTML=Date()">The time is?</button>
</body>
</html>
```

HTML event attribute calls JavaScript functions.

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
</body>
```

</html>

JavaScript Strings

```
var x = "John Doe";
```

String Length

To find the length of a string, use the built-in `length` property:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the `first` occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

The `lastIndexOf()` method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Searching for a String in a String

The `search()` method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

JavaScript Number Methods

The toString() Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

```
var x = 123;
x.toString();            // returns 123 from variable x
```

```
(123).toString();         // returns 123 from literal 123
(100 + 23).toString();    // returns 123 from expression 100 + 23
```

The toExponential() Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

```
var x = 9.656;
x.toExponential(2);     // returns 9.66e+0
x.toExponential(4);     // returns 9.6560e+0
x.toExponential(6);     // returns 9.656000e+0
```

The toFixed() Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

Example

```
var x = 9.656;
x.toFixed(0);           // returns 10
x.toFixed(2);           // returns 9.66
x.toFixed(4);           // returns 9.6560
x.toFixed(6);           // returns 9.656000
```

The toPrecision() Method

`toPrecision()` returns a string, with a number written with a specified length:

Example

```
var x = 9.656;
x.toPrecision();        // returns 9.656
x.toPrecision(2);       // returns 9.7
```

```
x.toPrecision(4);       // returns 9.656
x.toPrecision(6);       // returns 9.65600
```

The valueOf() Method

`valueOf()` returns a number as a number.

Example

```
var x = 123;
x.valueOf();            // returns 123 from variable x
(123).valueOf();        // returns 123 from literal 123
(100 + 23).valueOf();   // returns 123 from expression 100 + 23
```

23/11/2020

JavaScript Arrays

```
var array_name = [item1, item2, ...];
```

```
var cars = ["Saab", "Volvo", "BMW"];
```

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

The following example also creates an Array, and assigns values to it:

Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

The two examples above do exactly the same. There is no need to use `new Array()`. For simplicity, readability and execution speed, use the first one (the array literal method).

Access the Full Array

```
<script>

var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = cars;

</script>
```

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

Arrays use **numbers** to access its "elements".

Objects use **names** to access its "members".

```
<script>

var person = {firstName:"John", lastName:"Doe", age:46};

document.getElementById("demo").innerHTML = person["firstName"];

</script>
```

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator `typeof` returns "`object`":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits;    // returns object
```

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result: Banana,Orange,Apple,Mango

The `join()` method also joins all array elements into a string with user specified separator.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result: Banana * Orange * Apple * Mango

Adding Array Elements : Pushing

The easiest way to add a new element to an array is using the `push()` method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator `delete`:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];   // Changes the first element in fruits to undefined
```

Using **delete** may leave undefined holes in the array. Use pop() or shift() instead.

Popping

The `pop()` method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();      // Removes the last element ("Mango") from fruits
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();   // Removes the first element "Banana" from fruits
```

The `unshift()` method adds a new element to an array at the beginning.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

Splicing an Array

The `splice()` method can be used to add new items to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);     // Removes the first element of fruits
```

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys); // Concatenates (joins) myGirls
and myBoys
```

Example (Merging Three Arrays)

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2
and arr3
```

Sorting an Array

The `sort()` method sorts an array alphabetically:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // Sorts the elements of fruits
```

Reversing an Array

The `reverse()` method reverses the elements in an array.

You can use it to sort an array in descending order:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // First sort the elements of fruits
fruits.reverse();     // Then reverse the order of the element
```

Numeric Sort

By default, the `sort()` function sorts values as **strings**.

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
```

Array.map()

Array.filter()

Array.reduce()

Array.reduceRight()

Array.every()

Array.some()

Array.indexOf()

Array.lastIndexOf()

Array.find()

Array.findIndex()

JavaScript Random

Math.random()

`Math.random()` returns a random number between 0 (inclusive) and 1 (exclusive):

Example

```
Math.random();              /
```

JavaScript Random Integers

`Math.random()` used with `Math.floor()` can be used to return random integers.

Example

```
Math.floor(Math.random() * 10);    // returns a random integer from 0 to 9

Math.floor(Math.random() * 11);   // returns a random integer from 0 to 10

Math.floor(Math.random() * 100);  // returns a random integer from 0 to 99
```

HomeWork -

JavaScript Booleans

Logical Operators

if else and else if

Switch Statement

For Loop

While Loop

Break and Continue

Type Conversion

24/11/2020

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a **search pattern**.

In JavaScript, regular expressions are often used with the two **string methods**: `search()` and `replace()`.

JavaScript Errors

The `try` statement lets you test a block of code for errors.

The `catch` statement lets you handle the error.

The `throw` statement lets you create custom errors.

The `finally` statement lets you execute code, after try and catch, regardless of the result.

The "use strict" Directive

The `"use strict"` directive was new in ECMAScript version 5.

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

```
<script>

"use strict";
```

```
x = 3.14;  // This will cause an error (x is not defined).

</script>
```

JavaScript Const

ECMAScript 2015

ES2015 introduced two important new JavaScript keywords: `let` and `const`.

Variables defined with `const` behave like `let` variables, except they cannot be reassigned:

Example

```
const PI = 3.141592653589793;
```

JavaScript Classes

JavaScript Classes are templates for JavaScript Objects.

Use the keyword `class` to create a class.

Always add a method named `constructor()`:

Syntax

```
class ClassName {
  constructor() { ... }
}

class Car {
  constructor(name, year) {
    this.name = name;
```

```
    this.year = year;
  }
  }
```

The HTML DOM (Document Object Model)

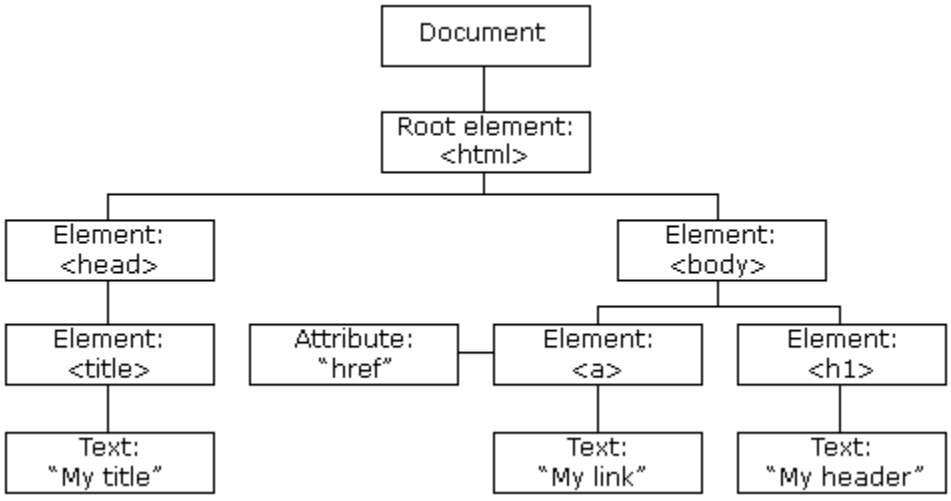
When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

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.**

The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page

- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<html>
<body>

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

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

</body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

Finding HTML Elements

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

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element

element.style.*property* = *new style*	Change the style of an HTML element

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1

document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1

document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1

document.URL	Returns the complete URL of the document	1

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

```
<h2>Finding HTML Elements by Id</h2>

<p id="intro">Hello World!</p>

<p>This example demonstrates the <b>getElementsById</b> method.</p>

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

<script>

var myElement = document.getElementById("intro");

document.getElementById("demo").innerHTML =

"The text from the intro paragraph is " + myElement.innerHTML;

</script>
```

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

```
<html>
<body>
<h2>Finding HTML Elements by Tag Name</h2>
<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
<script>
var x = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The text in first paragraph (index 0) is: ' + x[0].innerHTML;
</script>
</body>
</html>
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.

```
<p class="intro">The DOM is very useful.</p>
<p id="demo"></p>
<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
```

Finding HTML Elements by CSS Selectors

Finding HTML Elements by HTML Object Collections

The following HTML objects (and object collections) are also accessible:

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

JavaScript Form Validation

HTML form validation can be done by JavaScript.

```
<form name="loginform" onsubmit="return validateForm()" action="index.htm" method="post">

 <label for="username">Username:</label>

 <input type="text" placeholder="Enter Username" id="username" name="username">

 <label for="pwd">Password :</label>

 <input type="password"  placeholder="Enter Password" id="pwd" name="pwd">

<label for="Mobileno">Mobile Number :</label>

 <input type="number”  placeholder="Enter Mobile Number" id="mobno" name="mobno">

 <input type="submit" value="Login">

</form>
```

```
<script>

function validateForm()

{   var un = document.forms["loginform"]["username"].value;

  var pw = document.forms["loginform"]["pwd"].value;

  var mn = document.forms["loginform"]["mobno"].value;



  if (un == "") {
     alert("Username field is empty");
     return false;

     }

if (pw == "") {
     alert("Password field is empty");
     return false;

}

if (mn == "") {
     alert("Password must be filled out");
     return false;

}

if (isNaN(mn) || x < 1 || x > 10) {

alert("Mobile number is must include numeric values only");
     return false;

  }

 if ((un == "manik") && (pw == "disha"))

 {     return true;   }

 else

 { alert("Incorrect Credentials");
```

```
  return false;
 }
}
</script>
```

JavaScript Window - The Browser Object Model

There are no official standards for the **B**rowser **O**bject **M**odel (BOM). Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity,

The Window Object

The `window` object is supported by all browsers. It represents the browser's window. All global JavaScript objects, functions, and variables automatically become members of the window object.

`window.document.getElementById("header");` is the same as:
`document.getElementById("header");`

- `window.innerHeight`
- `window.innerWidth`
- `window.open()` - open a new window
- `window.close()` - close the current window
- `window.moveTo()` - move the current window
- `window.resizeTo()` - resize the current window
- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

`window.location` object can be written without the window prefix.

`window.location.href` returns the href (URL) of the current page

- `window.location.hostname` returns the domain name of the web host
- `window.location.pathname` returns the path and filename of the current page
- `window.location.protocol` returns the web protocol used (http: or https:)
- `window.location.assign()` loads a new document

AJAX

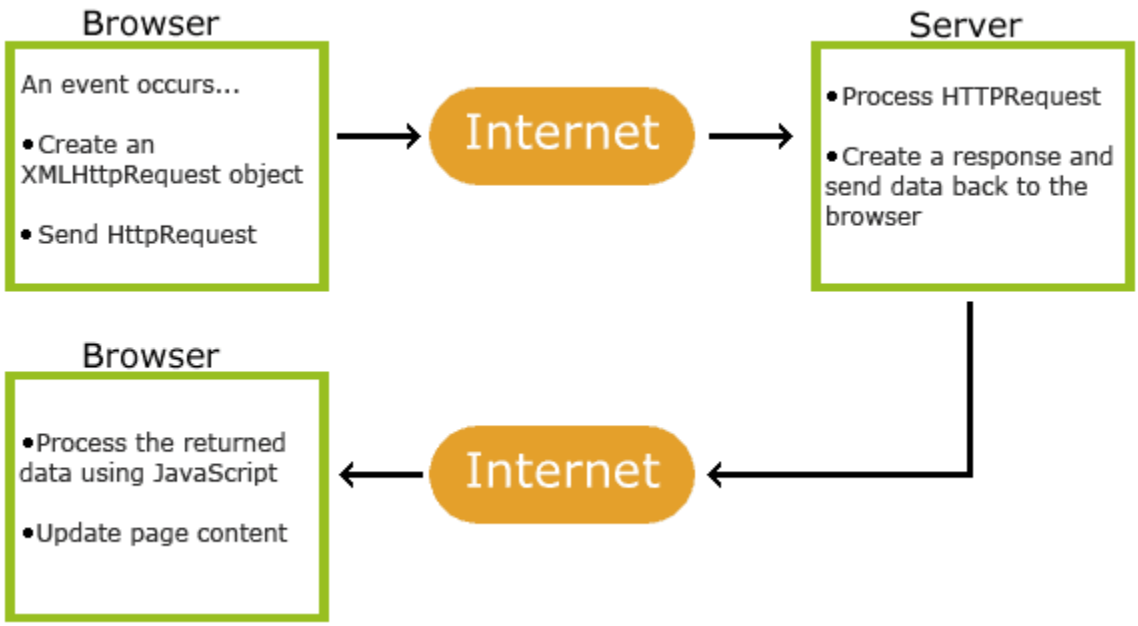
AJAX is a developer's dream, because you can:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.



Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field username is empty, the required attribute prevents this form from being submitted:

Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

```
<label for="username">Username:</label>
 <input type="text" placeholder="Enter Username" id="username" name="username" required>
 <label for="pwd">Password :</label>
 <input type="password"  placeholder="Enter Password" id="pwd" name="pwd" required>
<label for="Mobileno">Mobile Number :</label>
 <input type="number”  placeholder="Enter Mobile Number" id="mobno" name="mobno" required>
 <input type="submit" value="Login">
```

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful. Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

- Most often, the purpose of data validation is to ensure correct user input.

- Validation can be defined by many different methods, and deployed in many different ways.

- **Server side validation** is performed by a web server, after input has been sent to the server.

- **Client side validation** is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

The checkValidity() Method

```
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>

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

<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  }
```

```
}
</script>
```

Validity Properties

The **validity property** of an input element contains a number of properties related to the validity of data:

Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.

typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

JavaScript Objects

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

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

All JavaScript values, except primitives, are objects.