JAVASCRIPT FUNDAMENTALS

JavaScript often abbreviated as JS, is a programming language that conforms to the ECMA Script specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). JavaScript engines were originally used only in web browsers, but they are now embedded in server-side website deployments, usually via Node.js. They are also embedded in a variety of applications created with frameworks such as Electron and Cordova.

The "script" tag

JavaScript programs can be inserted into any part of an HTML document with the help of the `<script>` tag.

For instance:

```
<html>

<body>

  <p>Before the script...</p>

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

  <p>...After the script.</p>

</body>

</html>
```

Note: The *<script>* ***tag contains JavaScript code which is automatically executed when the browser processes the tag.***

The `<script>` tag has a few attributes that are rarely used nowadays but can still be found in old code:

- **The type attribute: <script type=…>**
  The old HTML standard, HTML4, required a script to have a type. Usually it was type="text/javascript". It's not required anymore. Also, the modern HTML standard totally changed the meaning of this attribute.
- **The language attribute: <script language=…>**
  This attribute was meant to show the language of the script. This attribute no longer makes sense because JavaScript is the default language.

External Scripts:

If we have a lot of JavaScript code, we can put it into a separate file.

Script files are attached to HTML with the *src* ***attribute:***

```
<script src="/path/to/script.js"></script>
```

Here, /path/to/script.js is an absolute path to the script from the site root. One can also provide a relative path from the current page. For instance, src="script.js" would mean a file "script.js" in the current folder.

We can give a full URL as well. For instance:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/loda.js"></script>4
```

To attach several scripts, use multiple tags:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
…
```

JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

var x = 5;
var y = 6;
var z = x + y;

- x stores the value 5
- y stores the value 6
- z stores the value 11

In programming, just like in algebra, we use variables to hold values.

JavaScript variables are containers for storing data values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

JavaScript Data Types

To be able to operate on variables, it is important to know something about the type.

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                         // Number
var lastName = "Johnson";                    // String
var x = {firstName:"John", lastName:"Doe"};    // Object
```

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

```
var x = 16 + 4 + "Volvo";
```

Result: 20Volvo

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

```
var x;           // Now x is undefined
x = 5;           // Now x is a Number
x = "John";      // Now x is a String
```

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

```
var carName1 = "Volvo XC60";   // Using double quotes
var carName2 = 'Volvo XC60';   // Using single quotes
```

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;        // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

JavaScript Booleans

Booleans can only have two values: true or false.

Example

```
var x = 5;
var y = 5;
var z = 6;
(x == y)      // Returns true
(x == z)      // Returns false
```

Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example

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

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Operators

1. **Addition** operator (+) adds numbers:

Adding

```
var x = 5;
var y = 2;
var z = x + y;
```

2. **multiplication** operator (*) multiplies numbers.

```
var x = 5;
var y = 2;
var z = x * y;
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	**Description**
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	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

The **addition assignment** operator (+=) adds a value to a variable.

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

Example

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
The result of txt3 will be:
John Doe
```

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;

The result of x, y, and z will be:
10
55
Hello55
```

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

Operator	Description
&&	logical and

||	logical or
!	logical not

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4

<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers. Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return 11111111111111111111111111111010

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

Example:

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

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
}
```

I. Function **parameters** are listed inside the parentheses () in the function definition.

II. Function **arguments** are the **values** received by the function when it is invoked.

III. Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation:

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return:

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example:

Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);   // Function is called, return value will end up in x


function myFunction(a, b) {
  return a * b;            // Function returns the product of a and b
}
```

The result in x will be:

Why Functions?

We can reuse code: Define the code once, and use it many times.

We can use the same code many times with different arguments, to produce different results.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Program: JavaScript Functions

```
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation, and returns the result:</p>

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

<script>
function myFunction(p1, p2)
{
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

Output:

```
This example calls a function which performs a calculation and returns the result:

12
```

JavaScript Methods

- **JavaScript String Methods:**
  Here is a list of each method and its description.

S. No.	Method & Description
1	charAt() Returns the character at the specified index.
2	charCodeAt() Returns a number indicating the Unicode value of the character at the given index.
3	concat() Combines the text of two strings and returns a new string.
4	indexOf() Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf() Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare() Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	length() Returns the length of the string.
8	match() Used to match a regular expression against a string.
9	replace() Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
10	search() Executes the search for a match between a regular expression and a specified string.
11	slice() Extracts a section of a string and returns a new string.
12	split()

	Splits a String object into an array of strings by separating the string into substrings.
13	substr() Returns the characters in a string beginning at the specified location through the specified number of characters.
14	substring() Returns the characters in a string between two indexes into the string.
15	toLocaleLowerCase() The characters within a string are converted to lower case while respecting the current locale.
16	toLocaleUpperCase() The characters within a string are converted to upper case while respecting the current locale.
17	toLowerCase() Returns the calling string value converted to lower case.
18	toString() Returns a string representing the specified object.
19	toUpperCase() Returns the calling string value converted to uppercase.
20	valueOf() Returns the primitive value of the specified object.

- **Array Methods:**
  Here is a list of each method and its description.

S. No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every() Returns true if every element in this array satisfies the provided testing function.
3	filter()

	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	forEach() Calls a function for each element in the array.
5	indexOf() Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	join() Joins all elements of an array into a string.
7	lastIndexOf() Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	map() Creates a new array with the results of calling a provided function on every element in this array.
9	pop() Removes the last element from an array and returns that element.
10	push() Adds one or more elements to the end of an array and returns the new length of the array.
11	reduce() Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	reduceRight() Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	reverse() Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	shift() Removes the first element from an array and returns that element.

15	slice() Extracts a section of an array and returns a new array.
16	some() Returns true if at least one element in this array satisfies the provided testing function.
17	toSource() Represents the source code of an object
18	sort() Sorts the elements of an array.
19	splice() Adds and/or removes elements from an array.
20	toString() Returns a string representing the array and its elements.
21	unshift() Adds one or more elements to the front of an array and returns the new length of the array.

- **Date Methods:**

Here is a list of each method and its description.

S. No.	Method & Description
1	Date() Returns today's date and time
2	getDate() Returns the day of the month for the specified date according to local time.
3	getDay()

	Returns the day of the week for the specified date according to local time.
4	getFullYear() Returns the year of the specified date according to local time.
5	getHours() Returns the hour in the specified date according to local time.
6	getMilliseconds() Returns the milliseconds in the specified date according to local time.
7	getMinutes() Returns the minutes in the specified date according to local time.
8	getMonth() Returns the month in the specified date according to local time.
9	getSeconds() Returns the seconds in the specified date according to local time.
10	getTime() Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	getTimezoneOffset() Returns the time-zone offset in minutes for the current locale.
12	getUTCDate() Returns the day (date) of the month in the specified date according to universal time.
13	getUTCDay() Returns the day of the week in the specified date according to universal time.
14	getUTCFullYear() Returns the year in the specified date according to universal time.
15	getUTCHours() Returns the hours in the specified date according to universal time.
16	getUTCMilliseconds() Returns the milliseconds in the specified date according to universal time.

17	getUTCMinutes() Returns the minutes in the specified date according to universal time.
18	getUTCMonth() Returns the month in the specified date according to universal time.
19	getUTCSeconds() Returns the seconds in the specified date according to universal time.
20	getYear() **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
21	setDate() Sets the day of the month for a specified date according to local time.
22	setFullYear() Sets the full year for a specified date according to local time.
23	setHours() Sets the hours for a specified date according to local time.
24	setMilliseconds() Sets the milliseconds for a specified date according to local time.
25	setMinutes() Sets the minutes for a specified date according to local time.
26	setMonth() Sets the month for a specified date according to local time.
27	setSeconds() Sets the seconds for a specified date according to local time.
28	setTime() Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	setUTCDate() Sets the day of the month for a specified date according to universal time.
30	setUTCFullYear()

	Sets the full year for a specified date according to universal time.
31	setUTCHours() Sets the hour for a specified date according to universal time.
32	setUTCMilliseconds() Sets the milliseconds for a specified date according to universal time.
33	setUTCMinutes() Sets the minutes for a specified date according to universal time.
34	setUTCMonth() Sets the month for a specified date according to universal time.
35	setUTCSeconds() Sets the seconds for a specified date according to universal time.
36	setYear() **Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
37	toDateString() Returns the "date" portion of the Date as a human-readable string.
38	toGMTString() **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	toLocaleDateString() Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	toLocaleFormat() Converts a date to a string, using a format string.
41	toLocaleString() Converts a date to a string, using the current locale's conventions.
42	toLocaleTimeString() Returns the "time" portion of the Date as a string, using the current locale's conventions.

43	toSource() Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	toString() Returns a string representing the specified Date object.
45	toTimeString() Returns the "time" portion of the Date as a human-readable string.
46	toUTCString() Converts a date to a string, using the universal time convention.
47	valueOf() Returns the primitive value of a Date object.

- **Math Methods**

Here is a list of each method and its description.

S. No.	Method & Description
1	abs() Returns the absolute value of a number.
2	acos() Returns the arccosine (in radians) of a number.
3	asin() Returns the arcsine (in radians) of a number.
4	atan() Returns the arctangent (in radians) of a number.
5	atan2() Returns the arctangent of the quotient of its arguments.
6	ceil() Returns the smallest integer greater than or equal to a number.
7	cos()

	Returns the cosine of a number.
8	exp() Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	floor() Returns the largest integer less than or equal to a number.
10	log() Returns the natural logarithm (base E) of a number.
11	max() Returns the largest of zero or more numbers.
12	min() Returns the smallest of zero or more numbers.
13	pow() Returns base to the exponent power, that is, base exponent.
14	random() Returns a pseudo-random number between 0 and 1.
15	round() Returns the value of a number rounded to the nearest integer.
16	sin() Returns the sine of a number.
17	sqrt() Returns the square root of a number.
18	tan() Returns the tangent of a number.
19	toSource() Returns the string "Math".

Controlling program flow in JavaScript

- **Switch case**

The basic syntax of the switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax

```
switch (expression) {
  case condition 1: statement(s)
                break;
  case condition 2: statement(s)
                break;
  ...
  case condition n: statement(s)
                break;
  default: statement(s)
}
```

Example

```
<script type="text/javascript">
var grade='A';
    document.write("Entering switch block<br/>");
    switch (grade) {
      case 'A': document.write("Good job<br/>");
        break;
      case 'B': document.write("Pretty good<br/>");
        break;
      case 'C': document.write("Passed<br/>");
        break;
      case 'D': document.write("Not so good<br/>");
        break;
      case 'F': document.write("Failed<br/>");
        break;
      default:  document.write("Unknown grade<br/>")
    }
    document.write("Exiting switch block");

</script>
```

- **Do while Loop**

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do{
  Statement(s) to be executed;
} while (expression);
```

Example

```
<script type="text/javascript">
    var count = 0;
    document.write("Starting Loop" + "<br/>");
    do{
      document.write("Current Count : " + count + "<br/>");
      count++;
    }while (count < 0);
    document.write("Loop stopped!");

</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Loop stopped!
```

- **While Loop**

The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.

Syntax

```
while(expression){
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
    var count = 0;
    document.write("Starting Loop" + "<br/>");
    while (count < 10){
      document.write("Current Count : " + count + "<br/>");
      count++;
    }
    document.write("Loop stopped!");

</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

- **For Loop**

The for loop is the most compact form of looping and includes the following three important parts −

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

Syntax

```
for(initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

Example

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

  var count;
  document.write("Starting Loop" + "<br/>");
  for(count = 0; count < 10; count++){
    document.write("Current Count : " + count );
    document.write("<br/>");
  }
  document.write("Loop stopped!");
```

```
</script>
```

This will produce following result which is similar to while loop −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

JavaScript Events

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

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

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

<*element event*=***'some JavaScript'***>

Example

<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

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

What can JavaScript Do?

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
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

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

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

JavaScript Object Model

JavaScript is an object-based language based on prototypes, rather than being class-based. Because of this different basis, it can be less apparent how JavaScript allows you to create hierarchies of objects and to have inheritance of properties and their values.

Document Object Model or DOM. The Document object has various properties that refer to other objects which allow access to and modification of document content. The way document content is accessed and modified is called the Document Object Model, or DOM. The Objects are organized in a hierarchy.

HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.
The HTML DOM model is constructed as a tree of Objects:

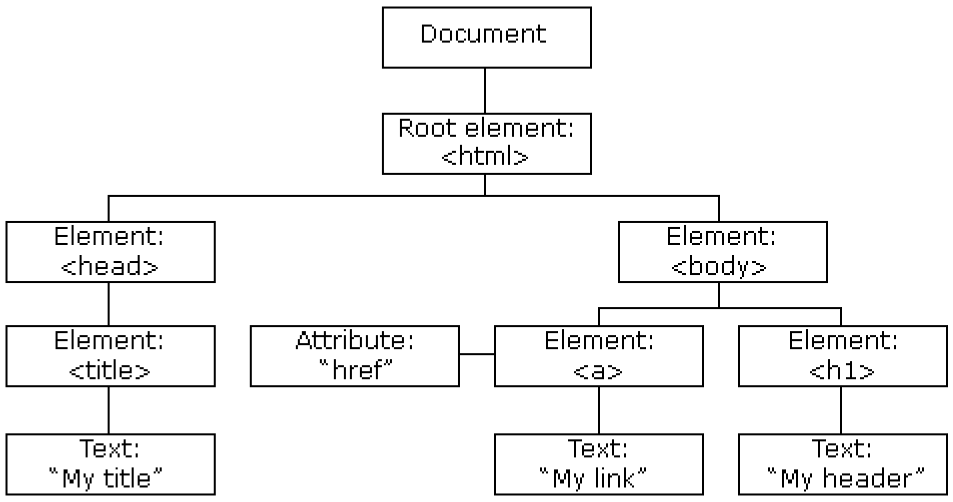


Fig: The HTML DOM Tree of Objects

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

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

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:

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 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).

Built-in Objects in JavaScript

These objects are used for simple data processing in the JavaScript.

1. Math Object

- Math object is a built-in static object.
- It is used for performing complex math operations.

1. Math Methods

Methods	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns cosine of a number.
floor()	Returns the largest integer less than or equal to a number.
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is base exponent.

Example: Simple Program on Math Object Methods

```
<html>
   <head>
      <title>JavaScript Math Object Methods</title>
   </head>
```

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

            var value = Math.abs(20);
            document.write("ABS Test Value : " + value +"<br>");

            var value = Math.acos(-1);
            document.write("ACOS Test Value : " + value +"<br>");

            var value = Math.asin(1);
            document.write("ASIN Test Value : " + value +"<br>");

            var value = Math.atan(.5);
            document.write("ATAN Test Value : " + value +"<br>");
        </script>
    </body>
</html>
```

Output

```
ABS Test Value : 20
ACOS Test Value : 3.141592653589793
ASIN Test Value : 1.5707963267948966
ATAN Test Value : 0.4636476090008061
```

2. Date Object

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.
- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

Syntax:

```
var variable_name = new Date();
```

Example:

```
var current_date = new Date();
```

Date Methods

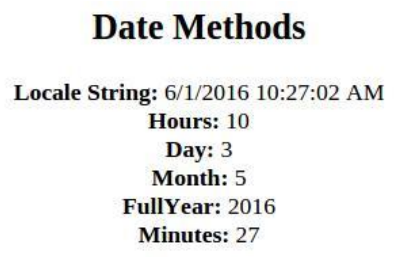
Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.
getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.
setMilliseconds()	Sets the milliseconds.
setTime()	Sets the number of milliseconds since January 1, 1970 at 12:00 AM.
setMonth()	Sets the month.
toDateString()	Returns the date portion of the Date as a human-readable string.
toLocaleString()	Returns the Date object as a string.
toGMTString()	Returns the Date object as a string in GMT timezone.
valueOf()	Returns the primitive value of a Date object.

Example : JavaScript Date() Methods Program

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

```
    <center>
        <h2>Date Methods</h2>
        <script type="text/javascript">
            var d = new Date();
            document.write("<b>Locale String:</b> " + d.toLocaleString()+"<br>");
            document.write("<b>Hours:</b> " + d.getHours()+"<br>");
            document.write("<b>Day:</b> " + d.getDay()+"<br>");
            document.write("<b>Month:</b> " + d.getMonth()+"<br>");
            document.write("<b>FullYear:</b> " + d.getFullYear()+"<br>");
            document.write("<b>Minutes:</b> " + d.getMinutes()+"<br>");
        </script>
    </center>
    </body>
</html>
```

Output:



```
          Date Methods

Locale String: 6/1/2016 10:27:02 AM
            Hours: 10
             Day: 3
            Month: 5
         FullYear: 2016
           Minutes: 27
```

3. String Object

- String objects are used to work with text.
- It works with a series of characters.

Syntax:
var variable_name = new String(string);

Example:
var s = new String(string);

String Properties

Properties	Description
length	It returns the length of the string.
prototype	It allows you to add properties and methods to an object.

constructor	It returns the reference to the String function that created the object.

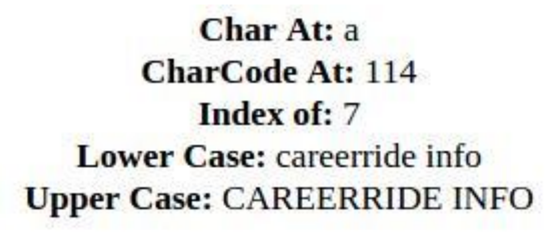
String Methods

Methods	Description
charAt()	It returns the character at the specified index.
charCodeAt()	It returns the ASCII code of the character at the specified position.
concat()	It combines the text of two strings and returns a new string.
indexOf()	It returns the index within the calling String object.
match()	It is used to match a regular expression against a string.
replace()	It is used to replace the matched substring with a new substring.
search()	It executes the search for a match between a regular expression.
slice()	It extracts a session of a string and returns a new string.
split()	It splits a string object into an array of strings by separating the string into the substrings.
toLowerCase()	It returns the calling string value converted lower case.
toUpperCase()	Returns the calling string value converted to uppercase.

Example : JavaScript String() Methods Program

```
<html>
    <body>
    <center>
        <script type="text/javascript">
            var str = "CareerRide Info";
            var s = str.split();
            document.write("<b>Char At:</b> " + str.charAt(1)+"<br>");
            document.write("<b>CharCode At:</b> " + str.charCodeAt(2)+"<br>");
            document.write("<b>Index of:</b> " + str.indexOf("ide")+"<br>");
            document.write("<b>Lower Case:</b> " + str.toLowerCase()+"<br>");
            document.write("<b>Upper Case:</b> " + str.toUpperCase()+"<br>");
        </script>
    <center>
    </body>
</html>
```

Output:



```
Char At: a
CharCode At: 114
Index of: 7
Lower Case: careerride info
Upper Case: CAREERRIDE INFO
```