JavaScript

By
Sudha Agarwal

INDEX

WHAT IS JAVASCRIPT

- JavaScript is:

- Is a lightweight, **interpreted** programming language.
- used to create interactive effects within web browsers.
- Designed for creating network-centric applications
- Complementary to and integrated with Java
- Complementary to and integrated with HTML
- Open and cross-platform

JAVASCRIPT SYNTAX

- A JavaScript consists of JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

- The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>
  JavaScript code
</script>
```

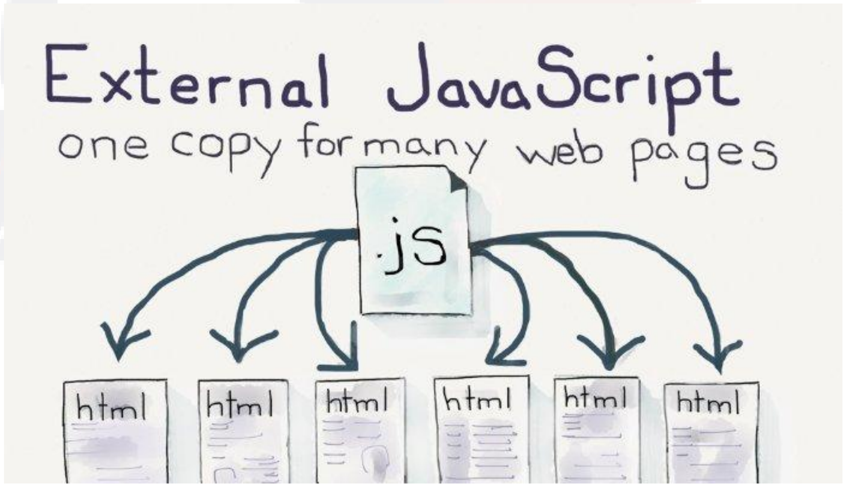
JAVASCRIPT IN <HEAD> OR <BODY>

- Scripts can be placed in the **<body>,** or in the **<head>** section of an HTML page, or in both.
- External JavaScript:
  - Scripts can also be placed in external files.
  - External scripts are practical when the same code is used in many different web pages.
  - JavaScript files have the file extension .js.

```
<!DOCTYPE html>
<html>
<head>
<script src="firstScript.js"></script>
</head>
</html>
```

External JavaScript Advantages

- Placing JavaScript in external files has some advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads



JavaScript Functions and Events

- A JavaScript **function** is a block of JavaScript code, that can be executed when "asked" for.

- For example, a function can be executed when an event occurs, like when the user clicks a button.

SEMICOLONS ARE OPTIONAL

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java.

- JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line. For example, the following code could be written without semicolons

```
<script language="javascrip”>
 <!–
var1 = 10
var2 =
 20
//--> </script>
```

CASE SENSITIVITY

- JavaScript is a **case-sensitive** language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

- So identifiers Time, TIme and TIME will have different meanings in JavaScript.

- NOTE: Care should be taken while writing your variable and function names in JavaScript.

COMMENTS IN JAVASCRIPT

- JavaScript supports both C-style and C++-style comments, Thus any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.

- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.

- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.

- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

COMMENTS IN JAVASCRIPT

```
<script>
 <!-
// This is a comment. It is similar to comments in C++

/* This is a multiline comment in JavaScript
* It is very similar to comments in C Programming
*/


//-->


</script>
```

JAVASCRIPT DATATYPES

- JavaScript allows you to work with three primitive data types:
- **Numbers** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.
- JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value.
- JavaScript supports a composite data type known as **object**.

JAVASCRIPT VARIABLES

- Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows:

```
<script>

    var money;
    var name;

</script>
```

JavaScript Variable Scope

- The **scope** of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes:
  - **Global Variables**: has global scope which means it is defined everywhere in your JavaScript code.
  - **Local Variables**: will be visible only within a function where it is defined.
  - Function parameters are always local to that function.

JAVASCRIPT IDENTIFIER

- 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).

JAVASCRIPT IDENTIFIER

- 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 Code Blocks

- JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
- The purpose of code blocks this is to define statements to be executed together.
- One place you will find statements grouped together in blocks, are in JavaScript functions:

```
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello Dolly.";
    document.getElementById("myDIV").innerHTML = "How are you?";
}
```

JavaScript Reserved Words

- The following are reserved words in JavaScript.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
doublet	in	super	

JavaScript Display Possibilities

- JavaScript can "display" data in different ways:
  - Writing into an alert box, using window.alert().
  - Writing into the HTML output using document.write().
  - Writing into an HTML element, using innerHTML.
  - Writing into the browser console, using console.log().

USING WINDOW.ALERT()

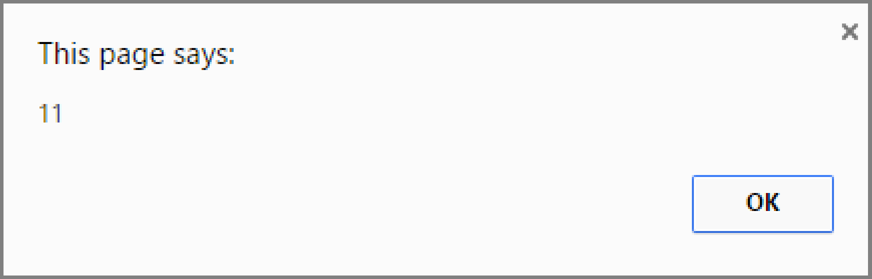
- You can use an alert box to display data

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

USING WINDOW.ALERT()

- You can use an alert box to display data

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```



This page says:

11

USING DOCUMENT.WRITE()

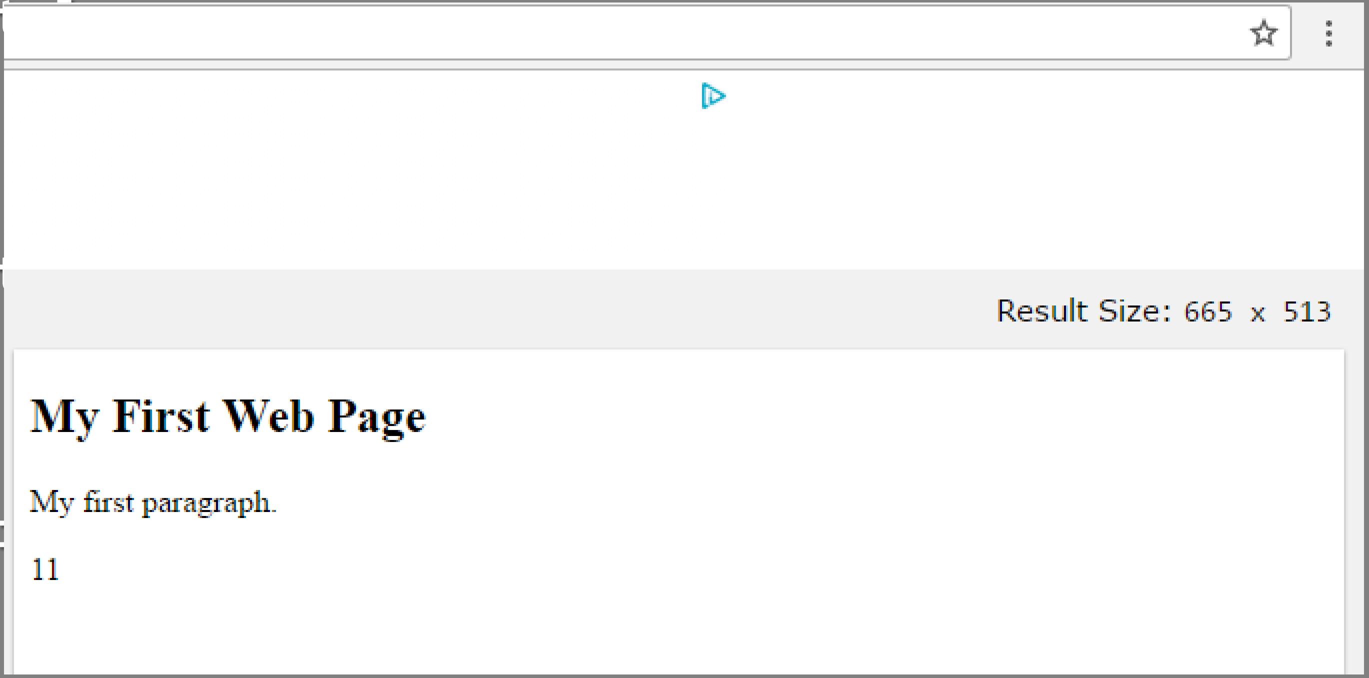
- For testing purposes, it is convenient to use document.write():

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

USING DOCUMENT.WRITE()

- For testing purposes, it is convenient to use document.write():

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

<h1>My First
<p>My first

<script>
document.wr
</script>

</body>
</html>
```



Result Size: 665 x 513

My First Web Page

My first paragraph.

11

USING INNERHTML

- To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

- The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

```
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

Confirmation Dialog Box

- A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

```
<head>
<script>

var retVal = confirm("Do you want to continue ?");
}

</script>

</head>
```

CONSOLE.LOG

- For debugging purposes, you can use the console.log() method to display data.

```
<body>

<script>
console.log(5 + 6);
</script>

</body>
```

PROMPT DIALOG BOX

- You can use prompt dialog box as follows

```
<head>
<script>

var retVal = prompt("Enter your name : ", "your name 
here");
   alert("You have entered : " +  retVal );

</script>
</head>
```

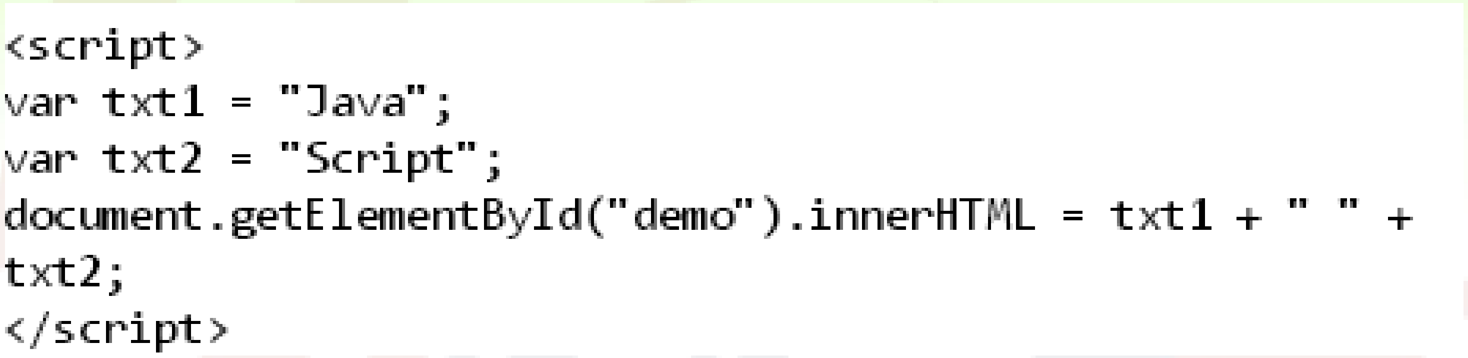
ARITHMETIC OPERATORS

- Arithmetic operators are used to perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

STRING OPERATORS

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



```
<script>
var txt1 = "Java";
var txt2 = "Script";
document.getElementById("demo").innerHTML = txt1 + " " +
txt2;
</script>
```

Output: Java Script

Adding Strings and Numbers

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

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

- **If you add a number and a string, the result will be a string!**

JAVASCRIPT ARITHMETIC

- The numbers (in an arithmetic operation) are called operands.
- The operation (to be performed between the two operands) is defined by an operator.

Operand	**Operator**	**Operand**
100	+	50

JAVASCRIPT ARITHMETIC

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

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

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

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

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

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

OPERATOR PRECEDENCE

- Operator precedence describes the order in which operations are performed in an arithmetic expression.

```
var x = 100 + 50 * 3;
```

- Is the result of example above the same as 150 * 3, or is it the same as 100 + 150?
- Is the addition or the multiplication done first?
- As in traditional school mathematics, the multiplication is done first.
- Multiplication (*) and division (/) have higher precedence than addition (+) and subraction (-).

OPERATOR PRECEDENCE

- And (as in school mathematics) the precedence can be changed by using parentheses:

```
var x = (100 + 50) * 3;
```

- When using parentheses, the operations inside the parentheses are computed first.

- When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

```
var x = 100 + 50 - 3;
```

Arithmetic Operator Precedence

- The following table lists JavaScript arithmetic operators, ordered from highest to lowest precedence:

Operator	**Precedence**
( )	Expression grouping
++ --	Increment and decrement
* / %	Multiplication, division, and modulo division
+ -	Addition and subtraction

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

COMPARISON AND LOGICAL OPERATORS

- Assignment operators assign values to JavaScript variables.

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

COMPARISON AND LOGICAL OPERATORS

- Comparison and Logical operators are used to test for true or false.
- Comparison operators are used in logical statements to determine equality or difference between variables or values.
- Given that x = 5, the table below explains the comparison operators:

Operator	**Description**	**Comparing**	**Returns**
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true

COMPARISON OPERATORS

!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	*true*

HOW CAN IT BE USED

- Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age < 18) text = "Too young";
```

LOGICAL OPERATORS

- Logical operators are used to determine the logic between variables or values.
- Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	**Description**	**Example**
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Conditional (Ternary) Operator

- JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
variablename = (condition) ? value1:value2
```

```
var voteable = (age < 18) ? "Too young":"Old enough";
```

- If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Comparing Different Types

- Comparing data of different types may give unexpected results.
- When comparing a string with a numeric constant, JavaScript will treat the number as a string when doing the comparison. The result of this is commonly not the same as a numeric comparison.
- When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

COMPARING DIFFERENT TYPES

- To secure a proper result, variables should be converted to the proper type before comparison:

```
age = Number(age);
if (isNaN(age)) {
    voteable = "Error in input";
} else {
    voteable = (age < 18) ? "Too young" : "Old enough";
}
```

The Number() function converts an empty string to 0, and a non numeric string to NaN.

JavaScript Data Types

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

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var cars = ["Saab", "Volvo", "BMW"];                // Array
var x = {firstName:"John", lastName:"Doe"};         // Object
```

JAVASCRIPT HAS DYNAMIC TYPES

- JavaScript has dynamic types. This means that the same variable can be used as different types

```
var x;                  // Now x is undefined
var x = 5;              // Now x is a Number
var 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:

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

JavaScript Booleans

- Booleans can only have two values: true or false.

```
var x = true;
var y = false;
```

JAVASCRIPT ARRAYS

- JavaScript arrays are written with square brackets.
- Array items are separated by commas.

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

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

JAVASCRIPT OBJECTS

- JavaScript objects are written with curly braces.
- Object properties are written as name:value pairs, separated by commas.

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

- The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

The typeof Operator

- You can use the JavaScript typeof operator to find the type of a JavaScript variable

```
typeof "John"                       // Returns string
typeof 3.14                         // Returns number
typeof false                        // Returns boolean
typeof [1,2,3,4]                    // Returns object
typeof {name:'John', age:34}        // Returns object
```

- In JavaScript, an array is a special type of object. Therefore typeof [1,2,3,4] returns object.

UNDEFINED

- In JavaScript, a variable without a value, has the value undefined. The typeof is also undefined.

```
var person;                    // Value is undefined, type is undefined
```

- Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

```
person = undefined;          // Value is undefined, type is undefined
```

Empty Values

- An empty value has nothing to do with undefined.
- An empty string variable has both a value and a type.

```
var car = "";                    // The value is "", the typeof is string
```

NULL

- In JavaScript null is "nothing". It is supposed to be something that doesn't exist.
- Unfortunately, in JavaScript, the data type of null is an object.

```
var car = "";                // The value is "", the typeof is string
```

- You can empty an object by setting it to null

```
var person = null;           // Value is null, but type is still an object
```

- You can also empty an object by setting it to undefined

```
var person = undefined;     // Value is undefined, type is undefined
```

Difference between undefined and null

```
typeof undefined              // undefined
typeof null                   // object
null === undefined            // false
null == undefined             // true
```

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).

```
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
}
```

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":

```
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
}
```

JAVASCRIPT OBJECTS

- In real life, a car is an object.
- A car has properties like weight and color, and methods like start and stop

Properties	Methods
car.name = Fiat	car.start()
car.model = 500	car.drive()
car.weight = 850kg	car.brake()
car.color = white	car.stop()

JavaScript Objects

- Objects are variables too. But objects can contain many values.
- This code assigns many values (Fiat, 500, white) to a variable named car:

```
var car = {type:"Fiat", model:500, color:"white"};
```

- The values are written as **name:value** pairs (name and value separated by a colon).

OBJECT PROPERTIES

- The **name:values pairs** (in JavaScript objects) are called properties.

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Property	**Property Value**
firstName	John
lastName	Doe
age	50
eyeColor	blue

OBJECT METHODS

- Methods are actions that can be performed on objects.
- Methods are stored in properties as function definitions.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

OBJECT DEFINITION

- You define (and create) a JavaScript object with an object literal:

```
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};
```

USER-DEFINED OBJECTS

- All user-defined objects and built-in objects are descendants of an object called Object.
- The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.
- The Object() Constructor - A constructor is a function that creates and initializes an object.
- The return value of the Object() constructor is assigned to a variable.
- The variable contains a reference to the new object.

USER-DEFINED OBJECTS

```
<head>
    <title>User-defined objects</title>
    <script type="text/javascript">
      var book = new Object();   // Create the object
      book.subject = "Perl"; // Assign properties to the object
      book.author  = "Mohtashim";
    </script>
    </head>

  <body>
    <script type="text/javascript">
      document.write("Book name is : " + book.subject + "<br>");
      document.write("Book author is : " + book.author + "<br>");
    </script>
  </body>
```

USER-DEFINED OBJECTS

```
<script type="text/javascript">
      function book(title, author){
        this.title = title;
        this.author  = author;
      }
    </script>
  </head>
  <body>
    <script type="text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
    </script>

  </body>
```

Accessing Object Properties

- You can access object properties in two ways:
  - objectName.propertyName
  - objectName[propertyName]

```
person.lastName;
```

```
person["lastName"];
```

Accessing Object Methods

- You access an object method with the following syntax:
  - objectName.methodName()

```
name = person.fullName();
```

LOCAL JAVASCRIPT VARIABLES

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables have local scope: They can only be accessed within the function.

```
function myFunction() {
    var carName = "Volvo";
}
```

- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
- Local variables are created when a function starts, and deleted when the function is completed.

Global JavaScript Variables

- A variable declared outside a function, becomes GLOBAL.
- A global variable has global scope: All scripts and functions on a web page can access it.

```
var carName = " Volvo";
// code here can use carName
function myFunction() {
    // code here can use carName
}
```

Automatically Global

- If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.
- This code example will declare carName as a global variable, even if it is executed inside a function.

```
// code here can use carName
function myFunction() {
    carName = "Volvo“;
    // code here can use carName
}
```

The Lifetime of JavaScript Variables

- The lifetime of a JavaScript variable starts when it is declared.
- Local variables are deleted when the function is completed.
- Global variables are deleted when you close the page.

HTML EVENTS

- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something.
- JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.
- <some-HTML-element some-event='some JavaScript'>

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

JavaScript Strings

- JavaScript strings are used for storing and manipulating text
- A JavaScript string simply stores a series of characters like "John Doe".
- A string can be any text inside quotes. You can use single or double quotes:

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

String Length

- The length of a string is found in the built in property length:

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

SPECIAL CHARACTERS

- Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var y = "We are the so-called "Vikings" from the north."
```

- The string will be chopped to "We are the so-called ".
- The solution to avoid this problem, is to use the \ escape character.
- The backslash escape character turns special characters into string characters:

```
var x = 'It\'s alright';
var y = "We are the so-called \"Vikings\" from the north."
```

BREAKING LONG CODE LINES

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

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

- The safest (but a little slower) way to break a long string is to use string addition:

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

Strings Can be Objects

- Normally, JavaScript strings are primitive values, created from literals: **var firstName = "John“**
- But strings can also be defined as objects with the keyword new: **var firstName = new String("John")**

```
var x = "John";
var y = new String("John");

// typeof x will return string
// typeof y will return object
```

String Properties and Methods

- Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).
- But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

STRING PROPERTIES

Property	Description
constructor	Returns the function that created the String object's prototype
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

STRING METHODS

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches

STRING METHODS

replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position through a number of characters
substring()	Extracts a part of a string between two specified positions

STRING METHODS

toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

JAVASCRIPT STRING METHODS

- **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:

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

- **Output:** 7

JAVASCRIPT STRING METHODS

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

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

- **Output:** 21

•Both the indexOf(), and the lastIndexOf() methods return -1 if the text is not found.
•Both methods accept a second parameter as the starting position for the search.

Searching for a String in a String

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

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

- **Output:** 7

indexOf() vs search()

- The two methods, indexOf() and search(), are equal.
- They accept the same arguments (parameters), and they return the same value.
- The two methods are equal, but the search() method can take much more powerful search values such as regular expression.

EXTRACTING STRING PARTS

- There are 3 methods for extracting a part of a string:
  - slice(start, end)
  - substring(start, end)
  - substr(start, length)

THE SLICE( ) METHOD

- slice() extracts a part of a string and returns the extracted part in a new string.
- The method takes 2 parameters: the starting index (position), and the ending index (position).

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
```

- **Output:** Banana

THE SLICE( ) METHOD

- If a parameter is negative, the position is counted from the end of the string.
- This example slices out a portion of a string from position -12 to position -6:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12,-6);
```

- **Output:** Banana

THE SLICE( ) METHOD

- If you omit the second parameter, the method will slice out the rest of the string:

```
var res = str.slice(7);
```

- **Output:** Banana, Kiwi
- or, counting from the end

```
var res = str.slice(-12);
```

- **Output:** Banana, Kiwi

The substring() Method

- substring() is similar to slice().
- The difference is that substring() cannot accept negative indexes.

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7,13);
```

- **Output:** Banana

- If you omit the second parameter, substring() will slice out the rest of the string.

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7);
```

- **Output:** Banana, Kiwi

The substr() Method

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

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7,6);
```

- **Output:** Banana
- If the first parameter is negative, the position counts from the end of the string.
- The second parameter can not be negative, because it defines the length.
- If you omit the second parameter, substr() will slice out the rest of the string.

REPLACING STRING CONTENT

- The **replace()** method replaces a specified value with another value in a string:

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft","W3Schools");
```

- **Output:** Please visit W3Schools!

- The replace() method can also take a regular expression as the search value.

Converting to Upper and Lower Case

- A string is converted to upper case with **toUpperCase():**

```
var text1 = "Hello World!";           // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to
upper
```

- **Output:** HELLO WORLD!
- A string is converted to lower case with **toLowerCase():**

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to
lower
```

- **Output:** hello world!

THE CONCAT() METHOD

- concat() joins two or more strings

```
var text1 = "Hello";
var text2 = "World";
text3 = text1.concat(" ",text2);
```

- **Output:** Hello World!

EXTRACTING STRING CHARACTERS

- There are 2 safe methods for extracting string characters:
- charAt(position)
- charCodeAt(position)

THE CHARAT() METHOD

- The charAt() method returns the character at a specified index (position) in a string

```
var str = "HELLO WORLD";
str.charAt(0);
```

- **Output:** H

THE CHARCODEAT() METHOD

- The charCodeAt() method returns the unicode of the character at a specified index in a string

```
var str = "HELLO WORLD“;
str.charCodeAt(0);
```

- **Output:** 72

Converting a String to an Array

- A string can be converted to an array with the split() method:

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

- If the separator is omitted, the returned array will contain the whole string in index [0].
- If the separator is "", the returned array will be an array of single characters:

```
var txt = "Hello";       // String
txt.split("");
```

TRIM( ) METHOD

- Remove whitespace from both sides of a string

```
var str = "       Hello World!        ";
alert(str.trim());
```

- **Output:** Hello World

MATCH() METHOD

- The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

```
var str = "The rain in SPAIN stays mainly in the plain";
var res = str.match(/ain/g);
```

- **Output:** ain,ain,ain

JavaScript Numbers

- JavaScript has only one type of number.
- Numbers can be written with, or without, decimals

```
var x = 34.00;    // A number with decimals
var y = 34;       // A number without decimals
```

NaN - Not a Number

- NaN is a JavaScript reserved word indicating that a value is not a number.
- Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

- However, if the string contains a numeric value , the result will be a number

```
var x = 100 / "10";     // x will be 10
```

Numbers Can Be Objects

- Normally JavaScript numbers are primitive values created from literals: **var x = 123**
- But numbers can also be defined as objects with the keyword new: **var y = new Number(123)**

```
var x = 123;
var y = new Number(123);

// typeof x returns number
// typeof y returns object
```

NUMBERS CAN BE OBJECTS

- When using the == equality operator, equal numbers looks equal:

```
var x = 500;
var y = new Number(500);
// (x == y) is true because x and y have equal values
```

- When using the === equality operator, equal numbers are not equal, because the === operator expects equality in both type

```
var x = 500;
var y = new Number(500);
// (x === y) is false because x and y have different types
```

Numbers Can be Objects

- Objects cannot be compared:

```
var x = new Number(500);
var y = new Number(500);

// (x == y) is false because objects cannot be compared
```

Number Methods: toString()

- **toString()** returns a number as a string.
- All number methods can be used on any type of numbers (literals, variables, or expressions):

```
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:

```
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 parameter is optional. If you don't specify it, JavaScript will not round the number.

THE TOFIXED( ) METHOD

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

```
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
```

Converting Variables to Numbers

- There are 3 JavaScript functions that can be used to convert variables to numbers:
- The Number() method
- The parseInt() method
- The parseFloat() method

THE NUMBER() METHOD

- Number(), can be used to convert JavaScript variables to numbers:

```
x = true;
Number(x);        // returns 1
x = false;
Number(x);        // returns 0
x = new Date();
Number(x);        // returns 1404568027739
x = "10"
Number(x);        // returns 10
x = "10 20"
Number(x);        // returns NaN
```

The parseInt() Method

- parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

```
parseInt("10");         // returns 10
parseInt("10.33");      // returns 10
parseInt("10 20 30");   // returns 10
parseInt("10 years");   // returns 10
parseInt("years 10");   // returns NaN
```

If the number cannot be converted, NaN (Not a Number) is returned

The parseFloat() Method

- parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned

```
parseFloat("10");        // returns 10
parseFloat("10.33");     // returns 10.33
parseFloat("10 20 30");  // returns 10
parseFloat("10 years");  // returns 10
parseFloat("years 10");  // returns NaN
```

JAVASCRIPT MATH OBJECT

- The Math object allows you to perform mathematical tasks on numbers.
- Math has no constructor. No methods have to create a Math object first.

Math.min() and Math.max()

- 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);      // returns -8

Math.max(0, 150, 30, 20, -8);      // returns 150
```

Math.random()

- Math.random() returns a random number between 0 and 1:

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

Math.round()

- Math.round() rounds a number to the nearest integer:

```
Math.round(4.7);            // returns 5
Math.round(4.4);            // returns 4
```

Math.ceil()

- Math.ceil() rounds a number up to the nearest integer:

```
Math.ceil(4.4);            // returns 5
```

Math.floor()

- Math.floor() rounds a number down to the nearest integer:

```
Math.floor(4.7);              // returns 4
```

Math.floor() and Math.random() can be used together to return a random number between 0 and 10:

```
Math.floor(Math.random() * 10) + 1);   // returns a random
number between 0 and 10
```

JavaScript Dates

- The Date object is a datatype built into the JavaScript language.
- The Date object lets us work with dates (years, months, days, hours, minutes, seconds, and milliseconds).
- Once a Date object is created, a number of methods allow us to operate on it.

JavaScript Date Formats

- A JavaScript date can be written as a string:
- Thu Apr 16 2015 06:08:06 GMT+0530 (India Standard Time)
- or as a number:
- 1429144686841
- Dates written as numbers, specifies the number of milliseconds since January 1, 1970, 00:00:00.

```
<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Creating Date Objects

- The Date object lets us work with dates.
- A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.
- Date objects are created with the new Date() constructor.
- There are 4 ways of initiating a date:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds,
milliseconds)
```

Creating Date Objects

- Using new Date(), creates a new date object with the current date and time

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

```
Thu Apr 16 2015 06:19:31 GMT+0530 (India Standard Time)
```

Creating Date Objects

- Using new Date(date string), creates a new date object from the specified date and time:

```
<script>
var d = new Date(“January 24, 2016 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

```
Sun Jan 24 2016 11:13:00 GMT+0530 (India Standard Time)
```

CREATING DATE OBJECTS

- Using new Date(number), creates a new date object as zero time plus the number.
- Zero time is 01 January 1970 00:00:00 UTC. The number is specified in milliseconds

```
<script>
var d = new Date(86400000);
document.getElementById("demo").innerHTML = d;
</script>
```

```
Fri Jan 02 1970 05:30:00 GMT+0530 (India Standard Time)
```

CREATING DATE OBJECTS

- Using new Date(7 numbers), creates a new date object with the specified date and time:
- The 7 numbers specify the year, month, day, hour, minute, second, and millisecond, in that order

```
<script>
var d = new Date(99,5,24,11,33,30,0);
document.getElementById("demo").innerHTML = d;
</script>
```

```
Thu Jun 24 1999 11:33:30 GMT+0530 (India Standard Time)
```

- JavaScript counts months from 0 to 11. January is 0. December is 11.

Creating Date Objects

- Variants of the previous example let us omit any of the last 4 parameters:

```
<script>
var d = new Date(2016,5,24);
document.getElementById("demo").innerHTML = d;
</script>
```

```
Fri Jun 24 2016 00:00:00 GMT+0530 (India Standard Time)
```

JAVASCRIPT DATE FORMATS

- There are generally 4 types of JavaScript date formats:
  - ISO Dates
  - Long Dates
  - Short Dates
  - Full Format

JAVASCRIPT DATE FORMATS

- **JavaScript ISO Dates**:
- The ISO 8601 syntax (YYYY-MM-DD) is also the preferred JavaScript date format:

```
var d = new Date("2015-03-25");
```

```
var d = new Date("2015-03");
```

```
var d = new Date("2015");
```

- It can be written with added hours, minutes, and seconds (YYY-MM-DDTHH:MM:SS):

```
var d = new Date("2015-03-25T12:00:00");
```

JAVASCRIPT DATE FORMATS

- **Long Dates**:
- Long dates are most often written with a "MMM DD YYYY" syntax

```
var d = new Date("Mar 25 2015");
```

- Month and day can be in any order:

```
var d = new Date("25 Mar 2015");
```

- And, month can be written in full (January), or abbreviated (Jan):

```
var d = new Date("January 25 2015");
```

JAVASCRIPT DATE FORMATS

- **Short Dates**:
- Short dates are most often written with an "MM/DD/YYYY" syntax

```
var d = new Date("03/25/2015");
```

- JavaScript will also accept "YYYY/MM/DD":

```
var d = new Date("2015/03/25");
```

JavaScript Date Formats

- **Full Date Format** :
- Short dates are most often written with an "MM/DD/YYYY" syntax

```
var d = new Date("Wed Mar 25 2015 00:00:00 GMT+0530 (India Standard Time)");
```

DATE PROPERTIES

- Date object has following properties:
- **Constructor** - Specifies the function that creates an object's prototype.

```
function Date() { [native code] }
```

- **Prototype** - The prototype property allows us to add properties and methods to an object.
- Use the following syntax to use Prototype.

```
object.prototype.name = value
```

DATE METHODS

- When a Date object is created, a number of methods allow us to operate on it.
- Date methods allow us to get and set the year, month, day, hour, minute, second, and millisecond of objects, using either local time or UTC (universal, or GMT) time.

Date Get Methods

- **The getTime() Method**
- getTime() returns the number of milliseconds since 01.01.1970

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>
```

The getFullYear() Method

- **getFullYear()** returns the year of a date as a four digit number.
- The value returned by getFullYear() is an absolute number

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML =
d.getFullYear();
</script>
```

THE GETDAY() METHOD

- **getDay()** returns the weekday as a number (0-6):

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getDay();
</script>
```

- The first of the week (0) means "Sunday"

THE GETDATE() METHOD

- **getDate()** Returns the day of the month for the specified date according to local time.
- The value returned by getDate() is an integer between 1 and 31

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getDate();
</script>
```

OTHER DATE GET METHODS

- **getHours()** - Returns the hour in the specified date according to local time.
- **getMilliseconds()** -Returns the milliseconds in the specified date according to local time.
- **getMinutes()** - Returns the minutes in the specified date according to local time
- **getMonth()** - Returns the month in the specified date according to local time.
- **getSeconds()** - Returns the seconds in the specified date according to local time.

Date Set Methods

- Set methods are used for setting a part of a date. Here are the most common

setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

The setFullYear() Method

- **setFullYear()** sets a date object to a specific date. In this example, to March 24, 2018

```
<script>
var d = new Date();
d.setFullYear(2018, 2, 24);
document.getElementById("demo").innerHTML = d;
</script>
```

```
Sat Mar 24 2018 01:28:38 GMT+0530 (India Standard Time)
```

The setDate() Method

- **setDate()** sets the day of the month (1-31)

```
<script>
var d = new Date();
d.setDate(20);
document.getElementById("demo").innerHTML = d;
</script>
```

```
Fri Jan 15 2016 01:29:10 GMT+0530 (India Standard Time)
```

Compare Dates

- Dates can easily be compared.
- The following example compares today's date with January 14, 2100

```
var today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);
if (someday > today) {
    text = "Today is before January 14, 2100.";
} else {
    text = "Today is after January 14, 2100.";
}
document.getElementById("demo").innerHTML = text;
```

JavaScript If...Else Statements

- Conditional statements are used to perform different actions based on different conditions

- In JavaScript we have the following conditional statements:
- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed
-

The if Statement

- Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
    block of code to be executed if the condition is true
}
```

```
if (hour < 18) {
    greeting = "Good day";
}
```

THE ELSE STATEMENT

- Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
    block of code to be executed if the condition is true
} else {
    block of code to be executed if the condition is false
}
```

```
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

The else if Statement

- Use the else if statement to specify a new condition if the first condition is false
- Syntax

```
if (condition1) {
   block of code to be executed if condition1 is true
} else if (condition2) {
   block of code to be executed if the condition1 is false and condition2 is true
} else {
   block of code to be executed if the condition1 is false and condition2 is false
}
```

JavaScript Switch Statement

- The switch statement is used to perform different actions based on different conditions
- Syntax

```
switch(expression) {
  case n:
    code block
    break;
  case n:
    code block
    break;
  default:
    default code block }
```

JavaScript Switch Statement

- This is how it works:
- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

The Break Keyword

- When the JavaScript code interpreter reaches a break keyword, it breaks out of the switch block.
- This will stop the execution of more execution of code and/or case testing inside the block.

The default keyword

- The default keyword specifies the code to run if there is no case match

JAVASCRIPT FOR LOOP

- Loops can execute a block of code a number of times
- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- Often this is the case when working with arrays:
- Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
```

- You can write:

```
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>"; }
```

Different Kinds of Loops

- JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

THE FOR LOOP

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

for (statement 1; statement 2; statement 3) {

code block to be executed

}

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

THE FOR LOOP

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

STATEMENT 1

Normally you will use statement 1 to initiate the variable used in the loop (var i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Statement 1

- And you can omit statement 1 (like when your values are set before the loop starts)

```
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Statement 2

- Often statement 2 is used to evaluate the condition of the initial variable.
- This is not always the case, JavaScript doesn't care. Statement 2 is also optional.
- If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.
- If you omit statement 2, you must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser

STATEMENT 3

- Often statement 3 increases the initial variable.
- This is not always the case, JavaScript doesn't care, and statement 3 is optional.
- Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.
- Statement 3 can also be omitted (like when you increment your values inside the loop):

```
var i = 0;
var len = cars.length;
for (; i < len; ) {
  text += cars[i] + "<br>";   i++;
}
```

THE FOR/IN LOOP

- The JavaScript for/in statement loops through the properties of an object

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

Example Example

The While Loop

- The while loop loops through a block of code as long as a specified condition is true.
- *Syntax*

```
while (condition) {
   code block to be executed
}
```

```
while (i < 10) {
   text += "The number is " + i;
   i++;
}
```

The do/While Loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- *Syntax*

do {

code block to be executed

}

- *while (condition);*

```
do {
   text += "The number is " + i;
   i++;
}
while (i < 10);
```

JavaScript Arrays

- JavaScript arrays are used to store multiple values in a single variable

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

<script>
var fruits= [“Apple", “Orange", “Mango"];
document.getElementById("demo").innerHTML = fuits;
</script>
```

The first line (in the script) creates an array named fruits.
The second line "displays" the array in the "innerHTML" of it.

WHAT IS AN ARRAY?

- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of fruit names, for example), storing the cars in single variables could look like this:
- **var fruit1 = “Apple";**
- **var fruit2 = “Orange";**
- **var fruit3 = “Mango**";
- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 fruits, but 300?
- The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

- Using an array literal is the easiest way to create a JavaScript Array.
- Syntax:
- *var array-name = [item1, item2, ...];*

- Example:
- var fruits= [“Apple", “Orange", “Mango"];

USING THE KEYWORD NEW

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

```
var fruits= new Array(“Apple", “Orange", “Mango");
```

ACCESS ELEMENTS OF AN ARRAY

- We refer to an array element by referring to the **index number**.
- This statement accesses the value of the first element in fruits:
- var name = fruits[0];
- This statement modifies the first element in fruits:

fruits[0] = “Banana";

Different Objects in One Array

- JavaScript variables can be objects. Arrays are special kinds of objects.
- Because of this, we can have variables of different types in the same Array.
- We can have objects in an Array. We can have functions in an Array. We can have arrays in an Array:
- myArray[0] = Date.now;
- myArray[1] = myFunction;
- myArray[2] = myCars;

Array Properties and Methods

- var x = fruits.length;   // The length property returns the number of elements in fruits
- var y = fruits.sort();   // The sort() method sort fruits in alphabetical order

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

Adding Array Elements

- The easiest way to add a new element to an array is to use the length property:
- Example

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

Adding Array Elements

- The easiest way to add a new element to an array is to use the length property:

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

- New element can also be added to an array using the push method:

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

Adding Array Elements

- Adding elements with high indexes can create undefined "holes" in an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[10] = "Lemon";                    // adds a new element (Lemon) to fruits
```

Looping Array Elements

- The best way to loop through an array, is using a "for" loop:

```
var index;
var fruits = ["Banana", "Orange", "Apple", "Mango"];
for      (index = 0; index < fruits.length; index++) {
    text += fruits[index];
}
```

JavaScript Array Methods

- **Converting Arrays to Strings -** The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

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

```
Banana,Orange,Apple,Mango
```

JAVASCRIPT ARRAY METHODS

- The **join()** method also joins all array elements into a string.
- It behaves just like toString(), but in addition we can specify the separator:

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

```
Banana * Orange * Apple * Mango
```

Popping and Pushing

- When we work with arrays, it is easy to remove elements and add new elements.
- This is what **popping** and **pushing** is:
- Popping items out of an array, or pushing items into an array.
- **Popping -** The **pop()** method removes the last element from an array:

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

- The pop() method returns the value that was "popped out":

Popping and Pushing

- **Pushing -** The **push()** method adds a new element to an array (at the end):

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

- The push() method returns the new array length

SHIFTING ELEMENTS

- Shifting is equivalent to popping, working on the first element instead of the last.
- The **shift()** method removes the first element of an array, and "shifts" all other elements one place up.

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

- The shift() method returns the string that was "shifted out".

SHIFTING ELEMENTS

- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements

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

- The unshift() method returns the new array length.

DELETING ELEMENTS

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**.

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

- Using delete on array elements leaves undefined holes in the array. Use pop() or shift() instead.

SPLICING AN ARRAY

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

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

- The first parameter (2) defines the position where new elements should be added (spliced in).
- The second parameter (0) defines how many elements should be removed.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

```
Banana,Orange,Lemon,Kiwi,Apple,Mango
```

Using splice() to Remove Elements

- Wecan use splice() to remove elements without leaving "holes" in the array:

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

- The first parameter (0) defines the position where new elements should be added (spliced in).
- The second parameter (1) defines how many elements should be removed.
- The rest of the parameters are omitted. No new elements will be added.

```
Orange,Apple,Mango
```

SORTING AN ARRAY

- Javascript array sort() method sorts the elements of an array.

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

```
Apple,Banana,Mango,Orange
```

- **Reversing an Array -** The **reverse()** method reverses the elements in an array.
- We can use it to sort an array in descending order:

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

JOINING ARRAYS

- The concat() method creates a new array by concatenating two arrays:

```
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
```

```
a,b,c,1,2,3
```

Slicing an Array

- The slice() method slices out a piece of an array into a new array.
- This example slices out a part of an array starting from array element 1 ("Orange"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

```
fruits: Banana,Orange,Lemon,Apple,Mango
citrus: Orange,Lemon,Apple,Mango
```

SLICING AN ARRAY

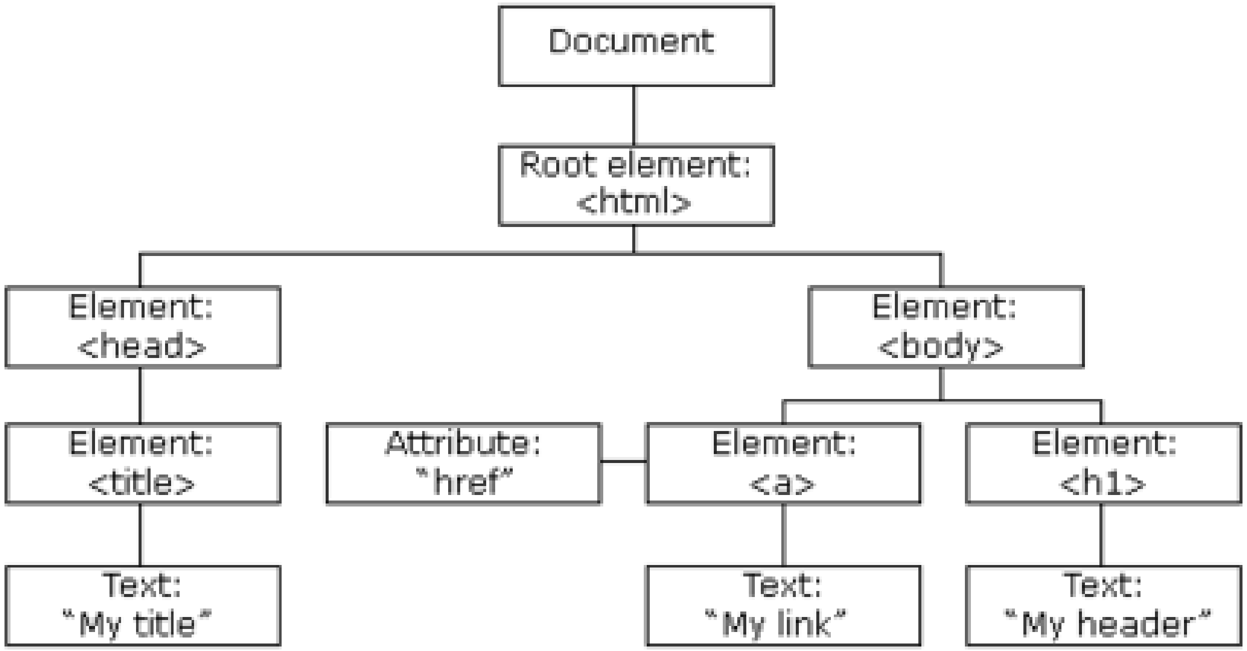
- The slice() method can take two arguments like slice(1,3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

```
fruits: Banana,Orange,Lemon,Apple,Mango
citrus: Orange,Lemon
```

JAVASCRIPT HTML DOM

- 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



JAVASCRIPT HTML DOM

- The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.
- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

JavaScript HTML DOM

- 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

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
- 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 (like changing the content of an HTML element).

The getElementById Method

- The most common way to access an HTML element is to use the id of the element.

- **The innerHTML Property**
- The innerHTML property is useful for getting or replacing the content of HTML elements

HTML DOM Methods

- getElementById is a method, while innerHTML is a property

```
<html>
<body>
<p id="demo"></p>

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

</body>
</html>
```

HTML DOM Document

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

FINDING HTML ELEMENTS

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

CHANGING HTML ELEMENTS

Method	**Description**
element.innerHTML=	Change the inner HTML of an element
element.*attribute=*	Change the attribute of an HTML element
element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property=*	Change the style of an HTML element

Adding Events Handlers

Method	**Description**
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

FINDING HTML ELEMENTS

- Often, with JavaScript, you want to manipulate HTML elements.
- To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

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

```
var x = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).
If the element is not found, x will contain null

Finding HTML Elements by Tag Name

- If you want to find all HTML elements with the tag name, use getElementsByTagName().

```
var x = document.getElementsByTagName("p");
```

This example finds the element with id="main", and then finds all <p> elements inside "main":

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Example

Finding HTML Elements by Class Name

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

```
var x = document.getElementsByClassName("intro");
```

Finding elements by class name does not work in Internet Explorer 8 and earlier versions.

Example

Finding HTML Elements by CSS Selectors

- If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

```
var x = document.querySelectorAll("p.intro");
```

The querySelectorAll() method does not work in Internet Explorer 8 and earlier versions.

Example

Finding HTML Elements by HTML Object Collections

- This example finds the form element with id="frm1", in the forms collection, and displays all element values:

```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i < x.length; i++) {
    text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Example

Changing HTML

- The HTML DOM allows JavaScript to change the content of HTML elements.
- **Changing the HTML Output Stream -** JavaScript can create dynamic HTML content.
- In JavaScript, document.write() can be used to write directly to the HTML output stream

Changing HTML Content

- The easiest way to modify the content of an HTML element is by using the innerHTML property.
- To change the content of an HTML element, use this syntax

```
document.getElementById(id).innerHTML = new HTML
```

Changing HTML Content

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

<h1 id="header">Old Header</h1>

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

</body>
</html>
```

Changing the Value of an Attribute

- To change the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute=new value
```

This example changes the value of the src attribute of an <img> element

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

Changing CSS - Changing HTML Style

- To change the style of an HTML element, use this syntax

```
document.getElementById(id).style.property=new style
```

The following example changes the style of a <p> element:

```
<script>
document.getElementById("p2").style.color = "blue";
</script>
```

Example

USING EVENTS

- The HTML DOM allows you to execute code when an event occurs.
- Events are generated by the browser when "things happen" to HTML elements:
- An element is clicked on
- The page has loaded
- Input fields are changed

Using Events

- This example changes the style of the HTML element with id="id1", when the user clicks a button

```
<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>
```

Example

HTML DOM Events

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

```
onclick=JavaScript
```

HTML DOM EVENTS

- Examples of HTML events:
  - When a user clicks the mouse
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key

HTML Event Attributes

- To assign events to HTML elements we can use event attributes

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

Assign Events Using the HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

- In the example above, a function named displayDate is assigned to an HTML element with the id="myBtn“.
- The function will be executed when the button is clicked.

Example

EVENT ATTRIBUTE

```
<!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>
```

HTML DOM EVENT

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

<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;
function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
</script>
</body>
</html>
```

The onload and onunload Events

- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload and onunload events can be used to deal with cookies.

```
<body onload="checkCookies()">
```

THE ONCHANGE EVENT

- The onchange event are often used in combination with validation of input fields.
- Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" onchange="upperCase()">
```

Example

The onmouseover and onmouseout Events

- The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)“>
Mouse Over Me</div>
```

Example

onmousedown, onmouseup and onclick events

- The onmousedown, onmouseup, and onclick events are all parts of a mouse-click.
- First when a mouse-button is clicked, the onmousedown event is triggered, then,
- when the mouse-button is released, the onmouseup event is triggered,
- finally, when the mouse-click is completed, the onclick event is triggered.

```
<div onmousedown="mDown(this)" onmouseup="mUp(this)“>
Click Me</div>
```

Example

HTML DOM EventListener

- The **addEventListener()** method - attaches an event handler to the specified element.
- It attaches an event handler to an element without overwriting existing event handlers.
- We can add many event handlers to one element.
- We can add many event handlers of the same type to one element, i.e two "click" events.
- We can add event listeners to any DOM object not only HTML elements. i.e the window object.

HTML DOM EventListener

- It makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- We can easily remove an event listener by using the removeEventListener() method.

```
document.getElementById("myBtn").addEventListener("click",
displayDate);
```

Example

HTML DOM EventListener

- Syntax:

```
element.addEventListener(event, function, useCapture);
```

- The first parameter is the type of the event (like "click" or "mousedown").
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a Boolean value specifying whether to use **event bubbling** or **event capturing**. This parameter is optional.

Add many Event Handlers to the same Element

- The addEventListener() method allows us to add many events to the same element, without overwriting existing events:

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

- You can add events of different types to the same element:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

Example

Event Handler to the Window Object

- The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object.
- Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
   document.getElementById("demo").innerHTML = sometext;
});
```

Example

PASSING PARAMETERS

- When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

```
<script>
var p1 = 5;
var p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
    myFunction(p1, p2);
});
function myFunction(a, b) {
    var result = a * b;
    document.getElementById("demo").innerHTML = result;
} </script>
```

Event Bubbling or Event Capturing?

- There are two ways of event propagation in the HTML DOM, bubbling and capturing.
- 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.

```
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.

The removeEventListener() Method

- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

```
element.removeEventListener("mousemove", myFunction);
```

```
<script>
document.getElementById("myDIV").addEventListener("mousemove",
myFunction);

function myFunction() {
    document.getElementById("demo").innerHTML = Math.random();
}
function removeHandler() {
    document.getElementById("myDIV").removeEventListener("mousemove",
myFunction);
}
```

Example

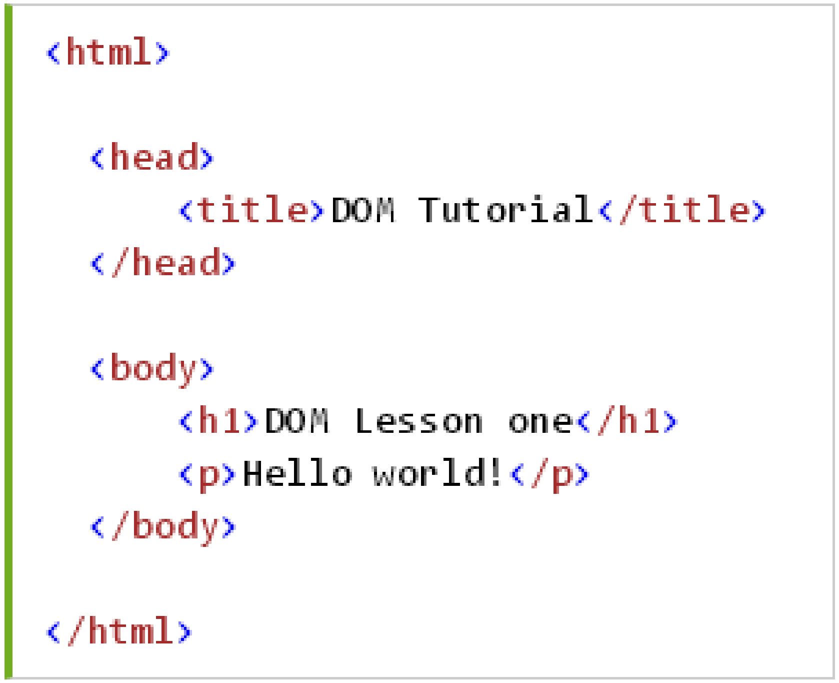
HTML DOM Navigation

- With the HTML DOM, you can navigate the node tree using node relationships.
- According to the W3C HTML DOM standard, everything in an HTML document is a node:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text inside HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - All comments are comment nodes

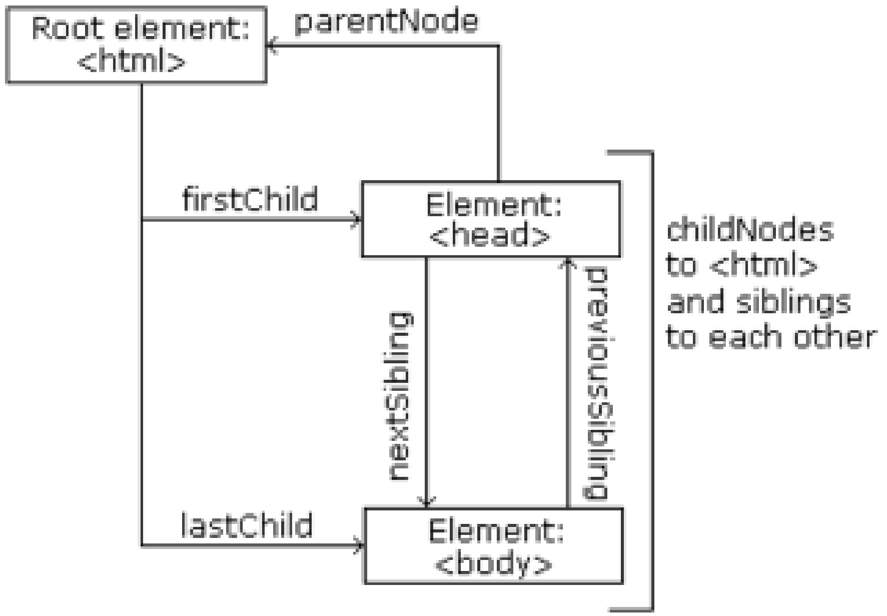
NODE RELATIONSHIPS

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

NODE RELATIONSHIPS



```
<html>

  <head>
      <title>DOM Tutorial</title>
  </head>

  <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
  </body>

</html>
```



NODE RELATIONSHIPS

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

Navigating Between Nodes

- We can use the following node properties to navigate between nodes with JavaScript:
  - parentNode
  - childNodes[nodenumber]
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling

Child Nodes and Node Values

- In addition to the innerHTML property, you can also use the childNodes and nodeValue properties to get the content of an element.

```
<h1 id="intro">My First Page</h1>
<p id="demo">Hello!</p>

<script>
var myText = document.getElementById("intro").childNodes[0].nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>
```

Example

Child Nodes and Node Values

- Using the firstChild property is the same as using childNodes[0]:

```
<h1 id="intro">My First Page</h1>

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

<script>
myText = document.getElementById("intro").firstChild.nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>
```

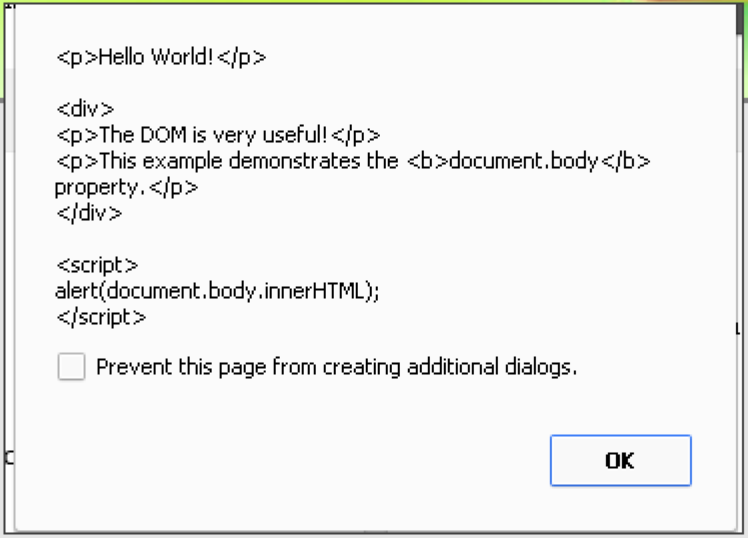
Example

DOM Root Nodes

- There are two special properties that allow access to the full document:
    - document.body - The body of the document
    - document.documentElement - The full document

DOM Root Nodes

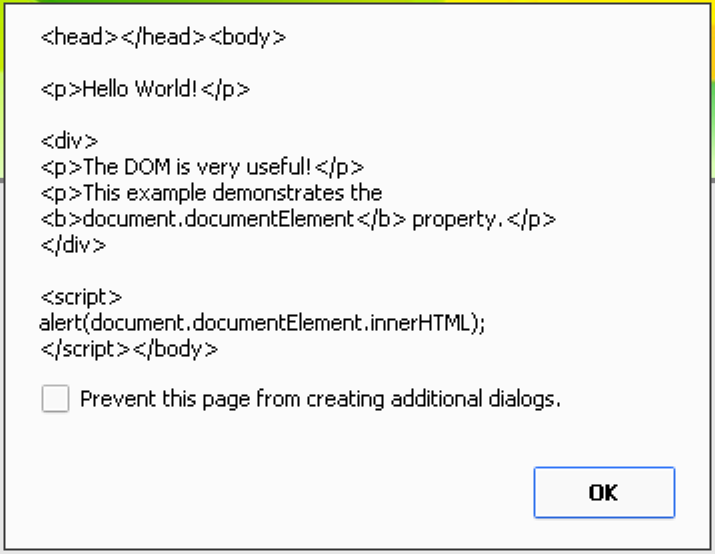
```
<!DOCTYPE html>
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.body</b> property.</p>
</div>

<script>
alert(document.body.innerHTML);
</script>

</body>
</html>
```



```
<p>Hello World!</p>

<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.body</b>
property.</p>
</div>

<script>
alert(document.body.innerHTML);
</script>
```

DOM Root Nodes

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

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.documentElement</b>
property.</p>
</div>


<script>
alert(document.documentElement.innerHTML);
</script>
</body>
</html>
```



```
<head></head><body>

<p>Hello World!</p>

<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the
<b>document.documentElement</b> property.</p>
</div>

<script>
alert(document.documentElement.innerHTML);
</script></body>
```

Creating New HTML Elements (Nodes)

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

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

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Creating new HTML Elements (Nodes)

- This code creates a new <p> element:
  - **var para = document.createElement("p");**
- To add text to the <p> element, we must create a text node first. This code creates a text node:
  - **var node = document.createTextNode("This is a new paragraph.");**
- Then we must append the text node to the <p> element:
  - **para.appendChild(node);**

Creating New HTML Elements (Nodes)

- Finally we must append the new element to an existing element.
- This code finds an existing element:
  - **var element = document.getElementById("div1");**
- This code appends the new element to the existing element:
  - **element.appendChild(para);**

Example

INSERTBEFORE()

- The appendChild() method in the previous example, appended the new element as the last child of the parent.
- If you don't want that you can use the insertBefore() method:

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

Example

Removing Existing HTML Elements

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

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

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

Example

REPLACING HTML ELEMENTS

- To replace an element to the HTML DOM, use the **replaceChild()** method:

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

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

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para,child);
</script>
```

Example

The Browser Object Model

- There are no official standards for the Browser Object Model (BOM).
- Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.
- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

The Window Object

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

```
window.document.getElementById("header");
```

Window methods

- Window methods are:
- window.open() - open a new window
- window.close() - close the current window
- window.moveTo() -move the current window
- window.resizeTo() -resize the current window

Window open() Method

- The open() method opens a new browser window.
- Syntax

  window.open(URL,name,specs,replace)

```
var myWindow = window.open("", "", "width=200, height=100");
```

- Open an about:blank page in a new window

Window open() Method

Parameter	**Description**
URL	Optional. Specifies the URL of the page to open. If no URL is specified, a new window with about:blank is opened
name	•Optional. Specifies the target attribute or the name of the window. The following values are supported:_blank - URL is loaded into a new window. This is default •_parent - URL is loaded into the parent frame •_self - URL replaces the current page •_top - URL replaces any framesets that may be loaded •*name* - The name of the window (**Note:** the *name* does not specify the title of the new window)

Window open( ) Method

specs	**Optional. A comma-separated list of items. The following values are supported:**
height=pixels	The height of the window. Min. value is 100
left=pixels	The left position of the window. Negative values not allowed
menubar=yes|no|1|0	Whether or not to display the menu bar
resizable=yes|no|1|0	Whether or not the window is resizable. IE only
scrollbars=yes|no|1|0	Whether or not to display scroll bars. IE, Firefox & Opera only
status=yes|no|1|0	Whether or not to add a status bar
top=pixels	The top position of the window. Negative values not allowed
width=pixels	The width of the window. Min. value is 100

WINDOW OPEN( ) METHOD

replace	•Optional. Specifies whether the URL creates a new entry or replaces the current entry in the history list. The following values are supported:t •true - URL replaces the current document in the history list •false - URL creates a new entry in the history list

Window close() method

- The close() method closes the current window.

```
function openWin() {
    myWindow = window.open("", "myWindow", "width=200,
height=100");   // Opens a new window
}

function closeWin() {
    myWindow.close();   // Closes the new window
}
```

Window moveTo() Method

- The moveTo() method moves a window's left and top edge to the specified coordinates.
- Syntax

  window.moveTo(x,y)

Parameter Values

Parameter	**Type**	**Description**
x	Number	Required. A positive or negative number that specifies the horizontal coordinate to be moved to
y	Number	Required. A positive or negative number specifies the vertical coordinate to be moved to

Window resizeTo() Method

- The resizeTo() method resizes a window to the specified width and height.
- Syntax

  window.resizeTo(width,height)

Parameter Values

Parameter	**Type**	**Description**
width	Number	Required. Sets the width of the window, in pixels
height	Number	Required. Sets the height of the window, in pixels

JAVASCRIPT WINDOW SCREEN

- The **window.screen** object contains information about the user's screen.
- Properties:
  - **screen.width**
  - **screen.height**
  - **screen.availWidth**
  - **screen.availHeight**

Example

JAVASCRIPT WINDOW LOCATION

- The **window.location** object can be used to get the current page address (URL) and to redirect the browser to a new page.
- **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:)

JAVASCRIPT WINDOW LOCATION

- The **window.location.href** property returns the URL of the current page.

```
document.getElementById("demo").innerHTML =
"Page location is " + window.location.href;
```

```
Page location is
https://www.jsTutorial.com/Js/js_window_location.asp
```

JavaScript Window Navigator

- The window.navigator object contains information about the visitor's browser.
- **Navigator Cookie Enabled -** The property cookieEnabled returns true if cookies are enabled, otherwise false:

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

<script>
document.getElementById("demo").innerHTML =
"Cookies Enabled is " + navigator.cookieEnabled;
</script>
```

Example

JAVASCRIPT WINDOW NAVIGATOR

- **The Browser Names**
- The properties **appName** and **appCodeName** return the name of the browser:

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

<script>
document.getElementById("demo").innerHTML =
"Name is " + navigator.appName + ". Code name is " +
navigator.appCodeName;
</script>
```

JAVASCRIPT WINDOW HISTORY

- The **window.history** object contains the browsers history.
- To protect the privacy of the users, there are limitations to how JavaScript can access this object.
- Some methods:
- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

Window History Back

- The **history.back()** method loads the previous URL in the history list.
- This is the same as clicking the Back button in the browser.

```
<html>
<head>
<script>
function goBack() {
    window.history.back()
}
</script>
</head>
<body>
<input type="button" value="Back" onclick="goBack()">
</body>
</html>
```

Window History Forwad

- The history **forward()** method loads the next URL in the history list.
- This is the same as clicking the Forward button in the browser.

```
<html>
<head>
<script>
function goForward() {
    window.history.forward()
}
</script>
</head>
<body>
<input type="button" value="Forward" onclick="goForward()">
</body>
</html>
```

Regular Expressions

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

- When you search for data in a text, you can use this search pattern to describe what you are searching for.

- A regular expression can be a single character, or a more complicated pattern.

REGULAR EXPRESSIONS

- Syntax

```
/pattern/modifiers;
```

```
var patt = /expression/i;
```

- /expression/i is a regular expression.
- expression is a pattern (to be used in a search).
- i is a modifier (modifies the search to be case-insensitive).

USING STRING METHODS

- In JavaScript, regular expressions are often used with the two string methods: search() and replace().
- The search() method uses an expression to search for a match, and returns the position of the match.
- The replace() method returns a modified string where the pattern is replaced.

```
var str = “Javascript Tutorial";
var n = str.search(/tutorial/i);
```

```
var str = “Javascript Tutorial";
var res = str.replace(/tutorial/i, ”Class");
```

Modifiers

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Brackets

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any digit NOT between the brackets
(x|y)	Find any of the alternatives specified

METACHARACTERS

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word

QUANTIFIERS

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X to Y n's
n{X,}	Matches any string that contains a sequence of at least X n's

QUANTIFIERS

Quantifier	Description
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

RegExp Object Methods

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

USING TEST()

- The test() method is a RegExp expression method.
- It searches a string for a pattern, and returns true or false, depending on the result.
- The following example searches a string for the character "e":

```
var patt = /e/;
patt.test("The best things in life are free!");
```

USING EXEC()

- The exec() method is a RegExp expression method.
- It searches a string for a specified pattern, and returns the found text.
- If no match is found, it returns null.
- The following example searches a string for the character "e":

```
/e/.exec("The best things in life are free!");
```

Thank you!

Contact Us

info@4kitsolutions.com