Web Technologies

JavaScript

JavaScript

JavaScript

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript

JavaScript Advantages

- Javascript is the most popular programming language in the world and that makes it a programmer's great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.

- Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.

- Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.

JavaScript

JavaScript Advantages

- JavaScript usage has now extended to mobile app development, desktop app development, and game development, Artificial Intelligence etc.. This opens many opportunities for you as Javascript Programmer.

- Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

JavaScript

JavaScript Frameworks

There are many useful Javascript frameworks and libraries available:

- Angular
- React
- jQuery
- Vue.js
- Ext.js
- TensorFlowJS
- NodeRed
- Ember.js
- Meteor
- Mithril
- Node.js
- Polymer
- Aurelia
- Backbone.js

How to Write JavaScript

Two ways to write JavaScript Code

1. **Internal JavaScript**
   Is written in HTML page with the help of `<script></script>`

2. **External JavaScript**
   JavaScript code is saved in external file with .js extension and included in HTML using following tag

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

How to Write JavaScript

Steps for including external javascript

Step 1: create a javascript file with .js extension.

example **myscript.js**

and write following code in it and save it

```
document.write("Hello From External JavaScript!");
```

Step 2: Cteate HTML file with extension .html

example **mypage.html**

in the head section of html file write following tag to include java script

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

JavaScript Output

JavaScript can "display" data in different ways:

- Writing into the HTML output using **document.write().**
- Writing into an HTML element, using **innerHTML**.
- Writing into an alert box, using **window.alert().**
- Writing into the browser console, using **console.log().**

JavaScript

JavaScript Comments

Single Line Comments
Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
```

JavaScript

JavaScript Comments

Multi-line Comments

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

Any text between /* and */ will be ignored by JavaScript.

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
```

JavaScript

JavaScript Variables

Variable/Constant is a placeholder in memory. Variable is needed to store data on which operations can be performed.

it's value can vary during execution/processing.

There are 3 ways to declare a JavaScript variable:

- Using var
- Using let

JavaScript

JavaScript Variables

To declare/define a variable in JavaScript 'var' keyword is used.
The data type of a variable in JavaScript is defined based on the value stored in it.

```
int x = 10; // C Programming
var x = 10; // JavaScript
```

```
float f = 2.56; // C Programming
var f = 2.56; // JavaScript
```

JavaScript Variables/Constants

Variables are containers for storing data (values).

In this example, x, y, and z, are variables, declared with the var keyword:

Example

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

JavaScript Variables

Variables declared with the var always have Global Scope.
Variables declared with the var keyword can NOT have block scope

Example

Variables declared with `var` inside a { } block can be accessed from outside the block:

```
{
  var x = 2;
}
// x CAN be used here
```

JavaScript

JavaScript Variables

Naming Variables

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

JavaScript Variables

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

JavaScript

JavaScript Variables

Declaring (Creating) JavaScript Variables
Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var keyword:

```
var carName;
```

After the declaration, the variable has no value (technically it has the value of undefined).

To assign a value to the variable, use the equal sign:

```
carName = "Volvo";
```

JavaScript Variables

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with var and separate the variables by comma:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

JavaScript

JavaScript Variables

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

JavaScript

JavaScript Variables

JavaScript Let

The let keyword was introduced in ES6 (2015).

- Variables defined with let cannot be Redeclared.
- Variables defined with let must be Declared before use.
- Variables defined with let have Block Scope.

```
let x = "John Doe";

let y = 0;
```

JavaScript

JavaScript Variables

Block Scope
Before ES6 (2015), JavaScript had only Global Scope and Function Scope.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide Block Scope in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

Example

```
{
  let x = 2;
}
// x can NOT be used here
```

```
{
  var x = 2;
}
// x CAN be used here
```

Note: Variables declared with the var keyword can NOT have block scope.

JavaScript Constant

The **const** keyword was introduced in ES6 (2015).

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

Example

```
const PI = 3.141592653589793;
```

When to use JavaScript const?
As a general rule, always declare a variable with const unless you know that the value will change.

Use const when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

JavaScript

JavaScript Operators

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

JavaScript Operators

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

JavaScript

JavaScript Operators

JavaScript String Operators

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

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

JavaScript

JavaScript Operators

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

JavaScript

JavaScript Operators

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript

JavaScript Operators

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

JavaScript

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

JavaScript Functions

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
}
```

JavaScript

JavaScript Functions

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*

```
let 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 Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

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

JavaScript

JavaScript Conditional Statements

The if Statement

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

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

JavaScript Conditional Statements

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
}
```

JavaScript Conditional Statements

The else if Statement

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

```
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

JavaScript Conditional Statements

The JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

JavaScript Looping Statements

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
- **for/of** - loops through the values of an iterable 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

JavaScript Looping Statements

The For Loop

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

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

JavaScript Looping Statements

The For In Loop

The JavaScript for in statement loops through the properties of an Object:

```
for (key in object) {
  // code block to be executed
}
```

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

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

JavaScript Looping Statements

The For In Loop

The JavaScript for in statement loops through the properties of an Object:

```
for (key in object) {
  // code block to be executed
}
```

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

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

```
let numbers = [4, 7, 8, 3, 1];
```

```
for(let x in numbers){
    document.write(x + "<br/>");
}
```

Output of **for-in**

0
1
2
3
4

JavaScript Looping Statements

The For Of Loop

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

```
for (variable of iterable) {
  // code block to be executed
}
```

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

```
let numbers = [4, 7, 8, 3, 1];
```

```
for(let x of numbers){
    document.write(x + "<br/>");
}
```

Output of **for-of**

4
7
8
3
1

JavaScript Looping Statements

The While Loop

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

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

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

```
let numbers = [4, 7, 8, 3, 1];
```

```
let i=0;
while(i<numbers.length){
    document.write(numbers[i] + "<br/>");
    i++;
}
```

JavaScript Looping Statements

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.

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

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

```
let numbers = [4, 7, 8, 3, 1];
```

```
i=0;
do{
    document.write(numbers[i] + "<br/>");
    i++;
}while(i<numbers.length);
```

JavaScript Looping Statements

Nested Loop Examples

```
let x = [ [2, 3, 9],
          [5, 8, 1] ];

for(let i=0; i<x.length; i++){
    for(j=0; j<x[i].length; j++){
        document.write(x[i][j] + " ");
    }
    document.write("<br/>");
}
```

```
for(let row of x){
    for(let column of row){
        document.write(column + " ");
    }
    document.write("<br/>");
}
```

JavaScript Arrays

An array is a special variable, which can hold more than one value:

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

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

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

JavaScript

JavaScript Arrays

You can also create an array, and then provide the elements:

```
var cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Using the JavaScript Keyword new

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

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

JavaScript

JavaScript Arrays

Accessing Array Elements
You access an array element by referring to the index number:

```
var cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Changing an Array Element
This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

Access the Full Array
With JavaScript, the full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

JavaScript Arrays

Array Example

```
let arr = [10, "hello", 30, 3.14, true];

document.write('Fetching all array elements.<br/>');
document.write(arr);
document.write('<br/>');
document.write('Fetch individual array elements.<br/>');
document.write('Fetch second element of arrya.<br/>');
document.write("arr[1] = " + arr[1]);

arr[5] = 70;
document.write('Array elements after adding one.<br/>');
document.write(arr);
```

JavaScript Arrays

2D Array using new Array()

```
let arr = [
            [2, 9, 5],
            [7, 3, 1]
          ];
document.write(arr);
document.write("<br/>");
document.write("Rows = " + arr.length);
document.write("<br/>");
document.write("Columns = " + arr[0].length);

document.write("<br/>");
```

```
let x = new Array(2); //row

// column
x[0] = new Array(2, 9, 5);
x[1] = new Array(7, 3, 1);

document.write(arr);
document.write("<br/>");
document.write("Rows = " + arr.length);
document.write("<br/>");
document.write("Columns = " +arr[0].length);
document.write("<br/>");
document.write("Element at x[1][1] = " + x[1][1]);
```

JavaScript

JavaScript Arrays

2D Array using new Array()

```
let array2D = new Array(rows);
for (let i = 0; i < rows; i++) {
    array2D[i] = new Array(columns);
}
```

JavaScript Arrays

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

JavaScript

JavaScript Objects

An **Object** is a variable that can hold many variables.
Objects are collections of **key-value pairs**, where each key (known as **property names**) has a value.
Objects can describe anything like houses, cars, people, animals, or any other subjects.

Car Object


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

Different cars have the same **properties**, but the **property values** can differ from car to car.
Different cars have the same **methods**, but the methods can be performed **at different times**.

JavaScript Objects

JavaScript Objects

This code assigns **many values** (Fiat, 500, white) to an **object** named car:

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

Object Properties

You can access object properties in two ways:

objectName.propertyName

objectName["propertyName"]

JavaScript Objects

JavaScript Object Methods

Object methods are **actions** that can be performed on objects.
Object methods are **function definitions** stored as **property values**:

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

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

JavaScript Objects

JavaScript Object Example

```
let student = {
            "reg_no":1001,
            "name": "abc",

            "display": function(){
              document.write("Reg No: " + student.reg_no);
              document.write("<br/>");
              document.write("Name: " + student.name);
            }
};

student.display();
```

OUTPUT:

Reg No: 1001
Name: abc

JavaScript Objects

Using the new Keyword

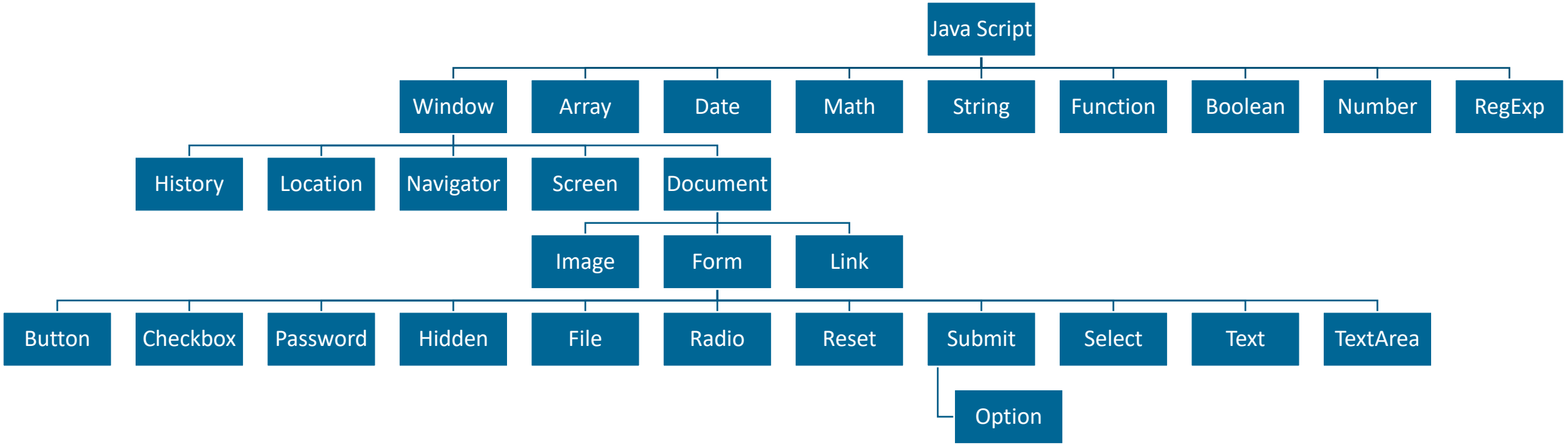
Create a new JavaScript object using `new Object()`:

```
// Create an Object
const person = new Object({
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
});
```

JavaScript

JavaScript Objects

A JavaScript object is an entity having state and behaviour (properties and method). JavaScript is an object-based language. Everything is an object in JavaScript.



JavaScript

JavaScript Objects

A window is one of the top-hierarchy objects on a browser. It is the parent of the whole DOM tree present inside a browser window (or tab), and the context reference where most of functions and event listeners will often run.

- Window
  - History
  - Location
  - Navigator
  - Screen
  - Document

History Object

The history object contains the URLs visited by the user (in the browser window). The history object is a property of the window object. The history object is accessed with:

```
window.history
history
```

Property/Method	Description
back()	Loads the previous URL (page) in the history list
forward()	Loads the next URL (page) in the history list
go()	Loads a specific URL (page) from the history list
length	Returns the number of URLs (pages) in the history list

Location Object

The location object contains information about the current URL. The location object is a property of the window object. The location object is accessed with:

```
window.location
location
```

Location Object

Location Object Properties

Property	Description
hash	Sets or returns the anchor part (#) of a URL
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the querystring part of a URL

Location Object

Location Object Methods

Method	Description
assign()	Loads a new document
reload()	Reloads the current document
replace()	Replaces the current document with a new one

The difference between assign() and replace():
replace() removes the current URL from the document history.
With replace() it is not possible to use "back" to navigate back to the original document.

Navigator Object

The navigator object contains information about the browser. The location object is a property of the window object. The navigator object is accessed with:

```
window.navigator
navigator
```

Navigator Object

The navigator object contains information about the browser. The navigator object is a property of the window object. The navigator object is accessed with:

```
window.navigator
navigator
```

JavaScript

Navigator Object

Navigator Object Properties

Property	Description
appCodeName	Returns browser code name
appName	Returns browser name
appVersion	Returns browser version
cookieEnabled	Returns true if browser cookies are enabled
geolocation	Returns a geolocation object for the user's location
language	Returns browser language
onLine	Returns true if the browser is online
platform	Returns browser platform
product	Returns browser engine name
userAgent	Returns browser user-agent header

Navigator Object

Navigator Object Methods

Method	Description
javaEnabled()	Returns true if the browser has Java enabled

Screen Object

The screen object contains information about the visitor's screen.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

Document Object

When an HTML document is loaded into a web browser, it becomes a document object. The document object is the root node of the HTML document. The document object is a property of the window object.

The document object is accessed with:

```
window.document
document
```

Document Object

The following properties can be used on HTML documents:

Property	Description
body	Sets or returns the document's body (the <body> element)
forms	Returns a collection of all <form> elements in the document
images	Returns a collection of all <img> elements in the document
links	Returns a collection of all <a> and <area> elements in the document that have a href attribute

JavaScript

Document Object

The following methods can be used on HTML documents:

Method	Description
addEventListener()	Attaches an event handler to the document
close()	Closes the output stream previously opened with document.open()
getElementById()	Returns the element that has the ID attribute with the specified value
getElementsByClassName()	Returns an HTMLCollection containing all elements with the specified class name
open()	Opens an HTML output stream to collect output from document.write()
getElementsByName()	Returns an live NodeList containing all elements with the specified name
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

Date Object

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

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

Date Object

JavaScript Date Methods

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
now()	Returns the number of milliseconds since midnight Jan 1, 1970

Date Object

JavaScript Date Methods

Method	Description
setDate()	Sets the day of the month of a date object
setFullYear()	Sets the year of a date object
setHours()	Sets the hour of a date object
setMilliseconds()	Sets the milliseconds of a date object
setMinutes()	Set the minutes of a date object
setMonth()	Sets the month of a date object
setSeconds()	Sets the seconds of a date object
setTime()	Sets a date to a specified number of milliseconds after/before January 1, 1970

JavaScript

String Object

JavaScript String Methods

```
var val = new String(string);
```

Method	Description
charAt()	Returns the character at a specified index (position)
concat()	Returns two or more joined strings
indexOf()	Returns the index (position) of the first occurrence of a value in a string
replace()	Searches a string for a value, or a regular expression, and returns a string where the values are replaced
search()	Searches a string for a value, or regular expression, and returns the index (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 number of characters from a string, from a start index (position)
toLowerCase()	Returns a string converted to lowercase letters
toUpperCase()	Returns a string converted to uppercase letters
trim()	Returns a string with removed whitespaces

Math Object

Math Object Properties

The Math object allows you to perform mathematical tasks.

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

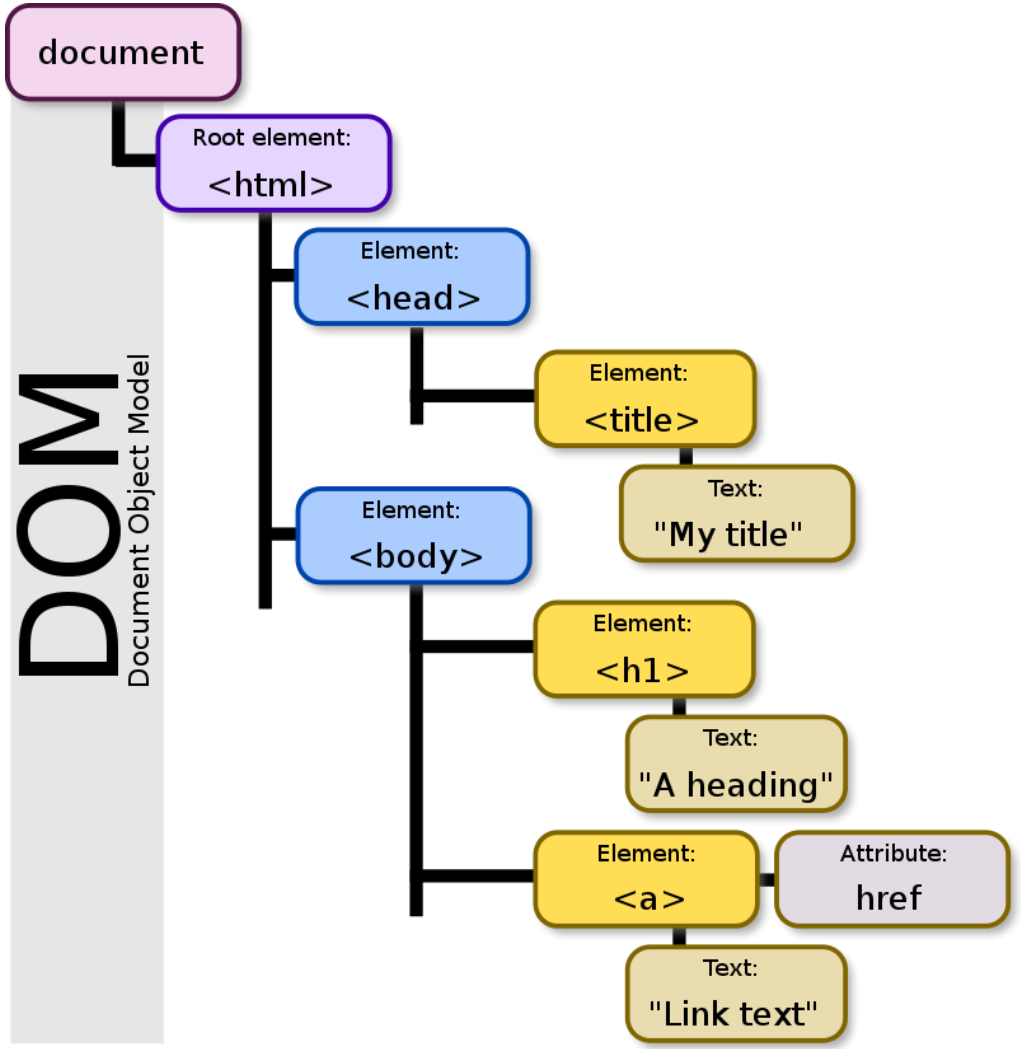
Math Object

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithmof x
log10(x)	Returns the base-10 logarithm of x
log2(x)	Returns the base-2 logarithm of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

DOM: Document Object Model

The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.



DOM: Document Object Model

DOM stands for Document Object Model. It is a programming interface that represents the structure of an HTML or XML document as a tree-like structure. In simple terms, it provides a way for web browsers to understand and manipulate web pages.

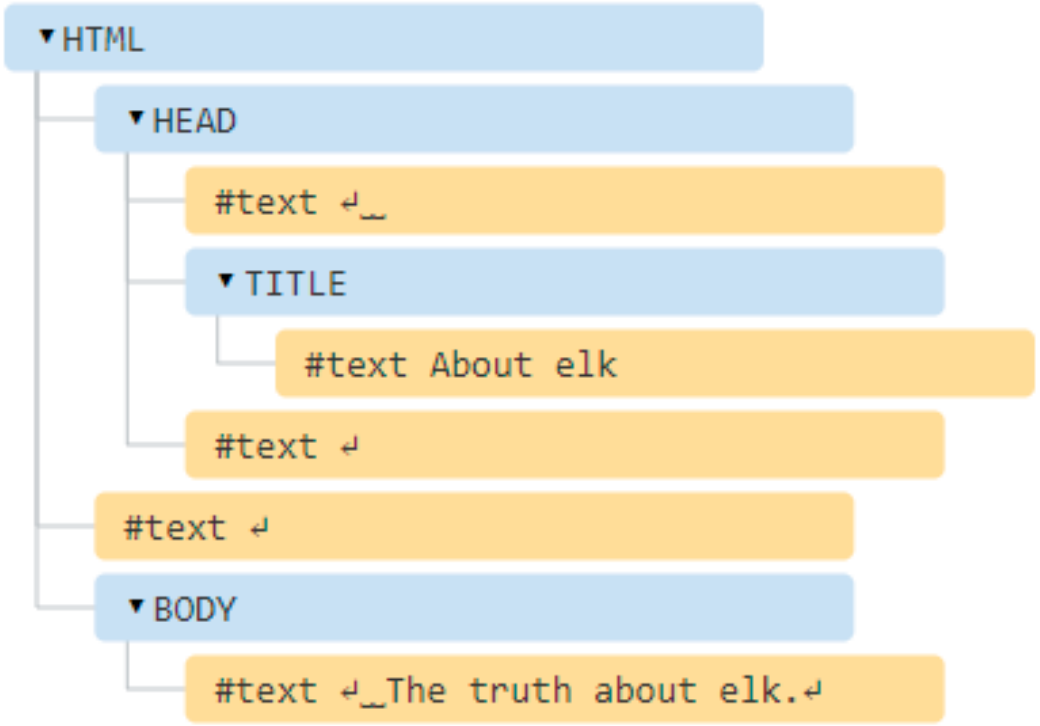
Imagine you have a web page with different elements like headings, paragraphs, images, and buttons. The DOM represents each of these elements as objects and organizes them in a hierarchical structure.

```
<html>
  <head>
    <title>My Web Page</title>
  </head>
  <body>
    <h1>Welcome to My Web Page</h1>
    <p>This is a paragraph.</p>
    <img src="image.jpg" alt="An image">
    <button>Click Me</button>
  </body>
</html>
```

DOM: Document Object Model

The DOM represents HTML as a tree structure of tags. Here's how it looks:

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4    <title>About elk</title>
5  </head>
6  <body>
7    The truth about elk.
8  </body>
9  </html>
```



DOM: Document Object Model

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

The DOM defines a standard for accessing documents:

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

JavaScript

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:

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

DHTML Examples

Demo

JavaScript

DHTML Events

An event is defined as changing the occurrence of an object.

It is compulsory to add the events in the DHTML page. Without events, there will be no dynamic content on the HTML page. The event is a term in the HTML, which triggers the actions in the web browsers.

Suppose, any user clicks an HTML element, then the JavaScript code associated with that element is executed. Actually, the event handlers catch the events performed by the user and then execute the code.

Example of events:

- Click a button.
- Submitting a form.
- An image loading or a web page loading, etc.

DHTML Events

S.No.	Event	When it occurs
1.	onabort	It occurs when the user aborts the page or media file loading.
2.	onblur	It occurs when the user leaves an HTML object.
3.	onchange	It occurs when the user changes or updates the value of an object.
4.	onclick	It occurs or triggers when any user clicks on an HTML element.
5.	ondblclick	It occurs when the user clicks on an HTML element two times together.
6.	onfocus	It occurs when the user focuses on an HTML element. This event handler works opposite to onblur.
7.	onkeydown	It triggers when a user is pressing a key on a keyboard device. This event handler works for all the keys.
8.	onkeypress	It triggers when the users press a key on a keyboard. This event handler is not triggered for all the keys.

DHTML Events

S.No.	Event	When it occurs
9.	onkeyup	It occurs when a user released a key from a keyboard after pressing on an object or element.
10.	onload	It occurs when an object is completely loaded.
11.	onmousedown	It occurs when a user presses the button of a mouse over an HTML element.
12.	onmousemove	It occurs when a user moves the cursor on an HTML object.
13.	onmouseover	It occurs when a user moves the cursor over an HTML object.
14.	onmouseout	It occurs or triggers when the mouse pointer is moved out of an HTML element.
15.	onmouseup	It occurs or triggers when the mouse button is released over an HTML element.
16.	onreset	It is used by the user to reset the form.

DHTML Events

S.No.	Event	When it occurs
17.	onselect	It occurs after selecting the content or text on a web page.
18.	onsubmit	It is triggered when the user clicks a button after the submission of a form.
19.	onunload	It is triggered when the user closes a web page.

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.
When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
window.alert("sometext");
```

The `window.alert()` method can be written without the window prefix.

Example

```
alert("I am an alert box!");
```

JavaScript Popup Boxes

Confirm Box

A confirm box is often used if you want the user to verify or accept something.
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

Syntax

```
window.confirm("sometext");
```

The `window.confirm()` method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
```

JavaScript Popup Boxes

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultText");
```

The `window.prompt()` method can be written without the window prefix.

Example

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```

THANK YOU