

# JAVA SCRIPT

# **Scripting Languages:**

- A high-level programming language that is interpreted by another program at runtime rather than compiled by the computer's processor as other programming languages (such as C and C++) are.
- Scripting language (also known as scripting, or script) is a series of commands that are able to be executed without the need for compiling.
- While all scripting languages are programming languages, not all programming languages are scripting languages.
- Scripting languages use a program known as an interpreter to translate commands and are directly interpreted from source code, not requiring a compilation step.
- Other programming languages, on the other hand, may require a compiler to translate commands into machine code before it can execute those commands.
- PHP, Perl, and Python are common examples of scripting languages.
- Scripting languages, which can be embedded within HTML, commonly are used to add functionality to a Web page, such as different menu styles or graphic displays or to serve dynamic advertisements.

# Types of Scripting Languages

- **Server Side Scripting and Client Side Scripting:**
- Server-side scripting languages run on a web server.
- When a client sends a request, the server responds by sending content via HTTP.
- In contrast, client-side scripting languages run on the client end—on their web browser.
- The benefit of client-side scripts is that they can reduce demand on the server, allowing web pages to load faster.
- Whereas, one significant benefit of server-side scripts is they are not viewable by the public like client-side scripts are.
- When trying to decide which way to go on a project, keep in mind that client-side scripting is more focused on user interface and functionality.
- Conversely, server-side scripting focuses on faster processing, access to data, and resolving errors.

# Types of Scripting Languages

## **Server-side Scripting Language**

- Can use huge resources of the server
- Complete all processing in the server and send plain pages to the client
- Reduces client-side computation overhead
- server-side scripting languages that manipulate the data, usually in a database, on the server.

## **Client-side Scripting Language**

- Does not involve server processing
- Complete application is downloaded to the client browser
- Client browser executes it locally
- Are normally used to add functionality to web pages e.g. different menu styles, graphic displays or dynamic advertisements
- client-side scripting languages, affecting the data that the end user sees in a browser window.

# Different Scripting Languages

- *Examples of Server-Side Scripting Languages*

Language	Comments
PHP	The most popular server-side language used on the web.
ASP.NET	Web-application framework developed by Microsoft.
Node.js	Can run on a multitude of platforms, including Windows, Linux, Unix, Mac, etc.
Java	Used in everything from your car stereo's Bluetooth to NASA applications.
Ruby	Dynamic. Focuses heavily on simplicity.
Perl	A bit of a mashup between C, shell script, AWK, and sed.
Python	Great for beginners to learn. Uses shorter code.

# **Client Side Scripting Languages**

## ***VBScript***

**Microsoft's scripting language**

**Client side Scripting language**

**Very easy to learn**

**Includes the functionality of Visual Basic**

## ***JavaScript***

**Client-side Scripting language**

**Easy to use programming language**

**Enhance dynamics and interactive features of a web page**

**Allows to perform calculation, write interactive games, add special effects, customize graphic selections, create security passwords**

# What is DHTML

- Generally, DHTML refers to applications that allow a Web page to change dynamically without requiring information to be passed to/from a web server.
- More specifically, DHTML refers to the interaction of HTML, CSS and Scripting language (JavaScript).
- Crucial component of DHTML is DOM (Document Object Model)
- DHTML = HTML + CSS + JavaScript + DOM

# Use of DHML

- To make Web pages interactive.
- HTML pages have static nature.
- DHTML provides us with enhanced creative control so we can manipulate any page element at any time.
- It is the easiest way to make Web pages interactive.
- It doesn't increase server workload and require special software to support.

# What is JavaScript?

JavaScript was designed to add interactivity to HTML pages.

- **JavaScript is a scripting language.**
- **JavaScript usually runs on the client-side (the browser's side).**
- **A scripting language is a lightweight programming language.**
- **JavaScript is usually embedded directly into HTML pages interpreted by browser.**
- **JavaScript is an interpreted language (means that scripts execute without preliminary compilation)**

- • *NOT Java*
  - – JavaScript was developed by Netscape
  - – Java was developed by Sun
- • *Designed to ‘plug a gap’ in the techniques available for creating web-pages*
  - – Client-side dynamic content
- • *Interpreted*

# What Can JavaScript do?

- *JavaScript gives HTML designers a programming tool*
- *JavaScript can react to events - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element*
- *JavaScript can read and write HTML elements - A JavaScript can read and change the content of an HTML element*

# What Can JavaScript do?

- *JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing*
- *JavaScript can be used to create cookies - A JavaScript can be used to store and retrieve information on the visitor's computer*

- **History:**
  - JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
  - ECMAScript is the official name of the language.
  - ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.
  - Since 2016 new versions are named by year (ECMAScript 2016 / 2017 / 2018).
  - The ECMAScript specification is a standardized specification of a scripting language developed by Brendan Eich of Netscape, initially named Mocha, then LiveScript, and finally JavaScript.
  - The first edition of **ECMA-262** was adopted by the Ecma General Assembly in June 1997.

- **ECMAScript Editions**

ES1	ECMAScript 1 (1997)	First edition
ES3	ECMAScript 3 (1999)	<p>Added regular expressions</p> <p>Added try/catch</p> <p>Added switch</p> <p>Added do-while</p>
	ECMAScript 5 (2009)	<p>Added strict mode</p> <p>Added JSON</p> <p>Added closures</p> <p>Added eval()</p> <p>Added Date.now()</p> <p>Added ArrayBuffer</p> <p>Added Symbol</p>

ES6	ECMAScript 2015	<ul style="list-style-type: none"> <li>Added let and const</li> <li>Added default parameter values</li> <li>Added Array.find()</li> <li>Added Array.findIndex()</li> </ul>
	ECMAScript 2016	<ul style="list-style-type: none"> <li>Added exponential operator (**)</li> <li>Added Array.includes()</li> </ul>
	ECMAScript 2017	<ul style="list-style-type: none"> <li>Added string padding</li> <li>Added Object.entries()</li> <li>Added Object.values()</li> <li>Added async functions</li> <li>Added shared memory</li> </ul>
	ECMAScript 2018	<ul style="list-style-type: none"> <li>Added rest / spread properties</li> <li>Added asynchronous iteration</li> <li>Added Promise.finally()</li> <li>Additions to RegExp</li> </ul>

# Difference between ES5 and ES6

- ***ECMAScript 5 (ES5P) :***
  - ES5 is also known as ECMAScript 2009 as it is released in 2009.
  - It is a function contractors focus on how the objects are instantiated.
  - For ES5 you have to write function keyword and return, to be used to define the function, like normal general JavaScript language.
- ***ECMAScript 6 (ES6) :***
  - ES6 is also known as ECMAScript 2015 as it is released in 2015.
  - Its class allows the developers to instantiate an object using the new operator, using an arrow function, in case it doesn't need to use function keyword to define the function, also return keyword can be avoided to fetch the computer value.

SR.NO.	ES5	ES6
1.	ECMA script is a trademarked scripting language specification defined by Ecma international. The fifth edition of the same is known as ES5	ECMA script is a trademarked scripting language specification defined by Ecma international. The sixth edition of the same is known as ES6
2.	It was introduced in 2009.	It was introduced in 2015.
3.	It supports primitive data types that are string, number, boolean, null, and undefined.	In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type 'symbol' for supporting unique values.
4.	There are only one way to define the variables by using the var keyword.	There are two new ways to define variables that are let and const.
5.	It has a lower performance as compared to ES6.	It has a higher performance than ES5.
6.	Object manipulation is time-consuming in ES5.	Object manipulation is less time-consuming in ES6.

7. In ES5, both function and return keywords are used to define a function.  
An arrow function is a new feature introduced in ES6 by which we don't require the function keyword to define the function.
8. It provides a larger range of community supports than that of ES6  
It provides a less range of community supports than that of ES5

# Common uses of JavaScript

- Alert messages
- Popup windows
- Form validation
- Displaying date/time

# What do I need to create JavaScript?

- *You can create JavaScript using the same softwares you use when creating HTML.*
- *Text editor.*
- *Web Browser.*

## **JavaScript Syntax:**

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

You can place the `<script>` tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the `<head>` tags.

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>
```

The script tag takes two important attributes:

language: This attribute specifies what scripting language you are using.

Typically, its value will be javascript.

type: This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

```
<script language="javascript" type="text/javascript">  
    JavaScript code  
</script>
```

## Java Script Program

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>
</body>
</html>
```

# SCRIPT tags

- *JavaScript code can be embedded in a Web page using SCRIPT tags*

```
<html>
```

```
<body>
```

```
<h1>My First Web Page</h1>
```

```
<script type="text/javascript">  
document.write("<p>" + Date() + "</p>");  
</script>
```

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

- OUTPUT

## **My First Web Page**

Sat Oct 01 2011 13:55:53 GMT+0530  
(India Standard Time)

# JavaScript Placement in HTML File

- *There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.*
- *Script in <head>...</head> section.*
- *Script in <body>...</body> section.*
- *Script in <body>...</body> and <head>...</head> sections.*
- *Script in an external file and then include in <head>...</head> section.*
- *In the following section we will see how we can put JavaScript in different ways:*

- **JavaScript in `<head>...</head>` section:**
- **If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows:**
- Example                   Output
- **JavaScript in `<body>...</body>` section:**
- **If you need a script to run as the page loads so that the script generates content in the page, the script goes in the `<body>` portion of the document. In this case you would not have any function defined using JavaScript:**
- Example                   output

- *JavaScript in <body> and <head> sections:*
- *You can put your JavaScript code in <head> and <body> section altogether as follows:*
- Example    output
- *JavaScript in External File :*
- *As you begin to work more extensively with JavaScript, you will likely find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.*
- *You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.*
- *Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute:*
- Example    output    external.js

# Scripts in <head> and <body>

- *JavaScript can be embedded in the <head>, <body>, or in both.*
- *common practice to put all functions in the head section.*
- *Function calls in body section*

# Using java script

## ● *Embedded :*

*Java script can be written in same .html file using script tag.*

[Js1.html](#) □ [output](#)

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

*External :*

*Linking to an external JavaScript file*

*Java script is in a file .js get call in a file .html  
inside script tag*

external.html □ OUTPUT  
xxx.txt

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

```
<script type="text/javascript"
src="xxx.js">
```

```
</script>
```

```
<p>
```

```
document.write
("This is External JavaScript
file.");
```

**xxx.js**

The actual script is in an external script file  
called "xxx.js" this is in ex.html.

```
</p>
```

```
</body>
```

```
</html>
```

**External.html**

```
<!DOCTYPE html>
<html><head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph
changed.";
}
</script>
</head>
<body>
<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button onclick="myFunction()">Try it</button>
</body>
</html>
```

example

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- Semicolons are Optional
- 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 But when formatted in a single line as follows, the semicolons are required:

```
<script language="javascript"  
type="text/javascript">  
<!--  
var1 = 10  
var2 = 20  
//-->  
</script>
```

```
<script language="javascript"  
type="text/javascript">  
<!--  
var1 = 10; var2 = 20;  
//-->  
</script>
```

- **JavaScript Identifiers**
- All JavaScript variables must be identified with unique names.
- These unique names are called identifiers.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
- The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with \$ and \_ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

- JavaScript 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.
- Java Script types are dynamic type.
- JavaScript Data Types:
- JavaScript allows you to work with following primitive data types:
  - Numbers eg. 123, 120.50 etc.
  - BigInt
  - Strings of text e.g. "This text string" etc.
  - Boolean e.g. true or false.
  - Null
  - Undefined
  - Symbol
- Non primitive: object,Array

# Data Types:

- Data types describe the different types or kinds of data that we're going to be working with and storing in variables.
- **Primitive data types:**
- In Java script, there are five basic, or primitive, types of data. The five most basic types of data are strings, numbers, booleans, undefined, and null.
- **Derived Data Types:**
- Objects and Arrays
- JavaScript Types are Dynamic
- JavaScript has dynamic types. This means that the same variable can be used to hold different data types:
- JavaScript variables can hold different data types: numbers, strings, objects and more:
- `let length = 16;` *// Number*
- `let lastName = "Johnson";` *// String*
- `let x = {firstName:"John", lastName:"Doe"};` *// Object*

Data Types	Description	Example
<code>String</code>	represents textual data	<code>'hello'</code> , <code>"hello world!"</code> etc
<code>Number</code>	an integer or a floating-point number	<code>3</code> , <code>3.234</code> , <code>3e-2</code> etc.
<code>BigInt</code>	an integer with arbitrary precision	<code>900719925124740999n</code> , <code>1n</code> etc.
<code>Boolean</code>	Any of two values: true or false	<code>true</code> and <code>false</code>
<code>undefined</code>	a data type whose variable is not initialized	<code>let a;</code>
<code>null</code>	denotes a <code>null</code> value	<code>let a = null;</code>
<code>Symbol</code>	data type whose instances are unique and immutable	<code>let value = Symbol('hello');</code>
<code>Object</code>	key-value pairs of collection of data	<code>let student = {};</code>

- *All data types except Object are primitive data types, whereas Object is non-primitive.*
- *The Object data type (non-primitive type) can store collections of data, whereas primitive data type can only store a single data.*

- *String is used to store text. In JavaScript, strings are surrounded by quotes:*

*Single quotes: 'Hello'*

*Double quotes: "Hello"*

*Backticks: `Hello`*

*For example,*

```
//strings example
```

```
const name = 'ram';
```

```
const name1 = "hari";
```

```
const result = `The names are ${name} and ${name1}`;
```

- *Single quotes and double quotes are practically the same and you can use either of them.*
- *Backticks are generally used when you need to include variables or expressions into a string.*
- *This is done by wrapping variables or expressions with \${variable or expression} as shown above.*

- *JavaScript Number*
- *Number represents integer and floating numbers (decimals and exponentials). For example,*

```
const number1 = 3;
```

```
const number2 = 3.433;
```

```
const number3 = 3e5 // 3 * 10^5
```

- *A number type can also be +Infinity, -Infinity, and NaN (not a number). For example,*

```
const number1 = 3/0;
```

```
console.log(number1); // Infinity
```

```
const number2 = -3/0;
```

```
console.log(number2); // -Infinity
```

```
// strings can't be divided by numbers
```

```
const number3 = "abc"/3;
```

```
console.log(number3); // NaN
```

- *JavaScript BigInt*
- *In JavaScript, Number type can only represent numbers less than  $(2^{53} - 1)$  and more than  $-(2^{53} - 1)$ .*
- *However, if you need to use a larger number than that, you can use the BigInt data type.*
- *A BigInt number is created by appending n to the end of an integer.*
- *For example,*

```
// BigInt value
const value1 = 900719925124740998n;

// Adding two big integers
const result1 = value1 + 1n;
console.log(result1); // "900719925124740999n"

const value2 = 900719925124740998n;
// Error! BitInt and number cannot be added
const result2 = value2 + 1;
console.log(result2);
```

- ***JavaScript Boolean***
- *This data type represents logical entities.*
- *Boolean represents one of two values: true or false.*
- *It is easier to think of it as a yes/no switch.*
- ***For example,***

```
const dataChecked = true;
```

```
const valueCounted = false;
```

- **JavaScript undefined**
- *The undefined data type represents value that is not assigned.*
- *If a variable is declared but the value is not assigned, then the value of that variable will be undefined.*
- *For example,*
- **`let name;`**
- **`console.log(name); // undefined`**
- *It is also possible to explicitly assign a variable value undefined. For example,*
- **`let name = undefined;`**
- **`console.log(name); // undefined`**

*It is recommended not to explicitly assign undefined to a variable.*

*Usually, null is used to assign 'unknown' or 'empty' value to a variable.*

- ***Undefined***
- In JavaScript, a variable without a value, has the value undefined.
- The type is also undefined.
- Example
- `let car; // Value is undefined, type is undefined`
- Any variable can be emptied, by setting the value to undefined. The type will also be undefined.
- Example
- `car = undefined; // Value is undefined, type is undefined`

- **JavaScript null**
  - *In JavaScript, null is a special value that represents empty or unknown value. For example,*
  - *const number = null;*
  - *The code above suggests that the number variable is empty.*
  - **Note: null is not the same as NULL or Null.**
- **JavaScript Symbol**
  - *This data type was introduced in a newer version of JavaScript (from ES2015).*
  - *A value having the data type Symbol can be referred to as a symbol value. Symbol is an immutable primitive value that is unique. For example,*
  - *// two symbols with the same description*
  - *const value1 = Symbol('hello');*
  - *const value2 = Symbol('hello');*

- ***JavaScript Symbol***
- *The JavaScript ES6 introduced a new primitive data type called Symbol. Symbols are immutable (cannot be changed) and are unique. For example,*
- *// two symbols with the same description*
- *const value1 = Symbol('hello');*
- *const value2 = Symbol('hello');*
- *console.log(value1 === value2); // false*
- *Though value1 and value2 both contain the same description, they are different.*

- ***Creating Symbol***

*You use the `Symbol()` function to create a `Symbol`. For example,*

```
// creating symbol  
const x = Symbol()  
typeof x; // symbol
```

***You can pass an optional string as its description. For example,***

```
const x = Symbol('hey');  
console.log(x); // Symbol(hey)
```

- *Access Symbol Description*
- *To access the description of a symbol, we use the . operator. For example,*
- `const x = Symbol('hey');`
- `console.log(x.description); // hey`

- *Add Symbol as an Object Key*
- *You can add symbols as a key in an object using square brackets []. For example,*

```
let id = Symbol("id");
let person = {
  name: "Jack",
  // adding symbol as a key
  [id]: 123 // not "id": 123
};
console.log(person); // {name: "Jack", Symbol(id): 123}
```

## Symbol Methods

There are various methods available with Symbol.

Method	Description
<code>for()</code>	Searches for existing symbols
<code>keyFor()</code>	Returns a shared symbol key from the global symbol registry.
<code>toSource()</code>	Returns a string containing the source of the Symbol object
<code>toString()</code>	Returns a string containing the description of the Symbol
<code>valueOf()</code>	Returns the primitive value of the Symbol object.

- ***Example: Symbol Methods***
- **// get symbol by name**
- `let sym = Symbol.for('hello');`
- `let sym1 = Symbol.for('id');`
  
- **// get name by symbol**
- `console.log( Symbol.keyFor(sym) ); // hello`
- `console.log( Symbol.keyFor(sym1) ); // id`

## Symbol Properties

Properties	Description
<code>asyncIterator</code>	Returns the default AsyncIterator for an object
<code>hasInstance</code>	Determines if a constructor object recognizes an object as its instance
<code>isConcatSpreadable</code>	Indicates if an object should be flattened to its array elements
<code>iterator</code>	Returns the default iterator for an object
<code>match</code>	Matches against a string
<code>matchAll</code>	Returns an iterator that yields matches of the regular expression against a string
<code>replace</code>	Replaces matched substrings of a string
<code>search</code>	Returns the index within a string that matches the regular
<code>species</code>	Creates derived objects
<code>toPrimitive</code>	Converts an object to a primitive value
<code>toStringTag</code>	Gives the default description of an object
<code>description</code>	Returns a string containing the description of the symbol

- *Example: Symbol Properties Example*
- `const x = Symbol('hey');`
- *// description property*
- `console.log(x.description); // hey`
- `const stringArray = ['a', 'b', 'c'];`
- `const numberArray = [1, 2, 3];`
  
- *// isConcatSpreadable property*
- `numberArray[Symbol.isConcatSpreadable] = false;`
- `let result = stringArray.concat(numberArray);`
- `console.log(result); // ["a", "b", "c", [1, 2, 3]]`

- ***JavaScript Objects***
- JavaScript objects are written with curly braces {}.
- Object properties are written as name:value pairs, separated by commas.
- Example
- `const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};`
- The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

- *JavaScript Object*
- *An object is a complex data type that allows us to store collections of data. For example,*

```
const student = {  
    firstName: 'ram',  
    lastName: null,  
    class: 10  
};
```

- *JavaScript Object Declaration*
- *The syntax to declare an object is:*

```
const object_name = {  
    key1: value1,  
    key2: value2  
}
```

- *Here, an object object\_name is defined. Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}.*

- *For example,*

```
// object creation
```

```
const person = {  
    name: 'John',  
    age: 20  
};  
  
console.log(typeof person); // object
```

- *JavaScript Object Properties*
- *In JavaScript, "key: value" pairs are called properties. For example,*

```
let person = {  
    name: 'John',  
    age: 20  
};
```

*Here, name: 'John' and age: 20 are properties.*

- *Accessing Object Properties*
- *You can access the value of a property by using its key.*

## 1. Using dot Notation

- *Here's the syntax of the dot notation.*
- *objectName.key*
- *For example,*

```
const person = {  
    name: 'John',  
    age: 20,  
};
```

- *// accessing property*

```
console.log(person.name); // John
```

- **2. Using bracket Notation**
- **Here is the syntax of the bracket notation.**
- *objectName["propertyName"]*
- **For example,**

```
const person = {  
    name: 'John',  
    age: 20,  
};
```

```
// accessing property  
console.log(person["name"]); // John
```

- **JavaScript Nested Objects**
- **An object can also contain another object. For example,**
- **// nested object**

```
const student = {  
    name: 'John',  
    age: 20,  
    marks: {  
        science: 70,  
        math: 75  
    }  
}  
  
// accessing property of student object  
console.log(student.marks); // {science: 70, math: 75}  
  
// accessing property of marks object  
console.log(student.marks.science); // 70
```

- ***JavaScript Object Methods***
- ***In JavaScript, an object can also contain a function. For example,***

```
const person = {  
    name: 'Sam',  
    age: 30,  
    // using function as a value  
    greet: function() { console.log('hello') }  
}
```

```
person.greet(); // hello
```

- ***JavaScript Arrays***
- JavaScript arrays are written with square brackets.
- Array items are separated by commas.
- The following code declares (creates) an array called cars, containing three items (car names):
- Example
- `const cars = ["Saab", "Volvo", "BMW"];`
- ***Array indexes are zero-based, which means the first item is [0], second is [1], and so on.***

- *JavaScript Type*
- *JavaScript is a dynamically typed (loosely typed) language. JavaScript automatically determines the variables' data type for you.*
- *It also means that a variable can be of one data type and later it can be changed to another data type. For example,*

```
// data is of undefined type  
let data;  
  
// data is of integer type  
data = 5;  
  
// data is of string type  
data = "JavaScript Programming";
```

- *JavaScript typeof:*
- *To find the type of a variable, you can use the typeof operator. For example,*

*const name = 'ram';*

*typeof(name); // returns "string"*

*const number = 4;*

*typeof(number); //returns "number"*

*const valueChecked = true;*

*typeof(valueChecked); //returns "boolean"*

*const a = null;*

*typeof(a); // returns "object"*

- *console.log("hello")*
- *let a=10*
- *console.log('a="'+a)*
- *console.log(typeof(a))*
- *var b=20*

- *Data types*
- *A value in JavaScript is always of a certain type. For example, a string or a number.*
- *There are eight basic data types in JavaScript. Here, we'll cover them in general and in the next chapters we'll talk about each of them in detail.*
- *We can put any type in a variable.*
- *For example, a variable can at one moment be a string and then store a number:*
- `// no error`
- `let message = "hello";`
- `message = 123456;`
- *Programming languages that allow such things, such as JavaScript, are called “dynamically typed”, meaning that there exist data types, but variables are not bound to any of them.*

- **Number:**
- *let n = 123;*
- *n = 12.345;*
- *The number type represents both integer and floating point numbers.*
- *There are many operations for numbers, e.g. multiplication \*, division /, addition +, subtraction -, and so on.*
- *Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: Infinity, -Infinity and NaN.*
- *Infinity represents the mathematical Infinity  $\infty$ . It is a special value that's greater than any number.*
- *We can get it as a result of division by zero:*
- *alert( 1 / 0 ); // Infinity*

- *alert( 1 / 0 ); // Infinity*
- *alert( Infinity ); // Infinity*
- *NAN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:*
- *alert( "not a number" / 2 ); // NaN, such division is erroneous*
- *NAN is sticky. Any further mathematical operation on NAN returns NAN:*
- *alert( NaN + 1 ); // NaN*
- *alert( 3 \* NaN ); // NaN*
- *alert( "not a number" / 2 - 1 ); // NaN*
- *So, if there's a NAN somewhere in a mathematical expression, it propagates to the whole result (there's only one exception to that: NaN \*\* 0 is 1).*

- *In JavaScript, the “number” type cannot safely represent integer values larger than  $(2^{53}-1)$  (that’s 9007199254740991), or less than  $-(2^{53}-1)$  for negatives.*
- *To be really precise, the “number” type can store larger integers (up to 1.7976931348623157 \* 10308), but outside of the safe integer range  $\pm(2^{53}-1)$  there’ll be a precision error, because not all digits fit into the fixed 64-bit storage.*
- *So an “approximate” value may be stored.*
- *For example, these two numbers (right above the safe range) are the same:*
- `console.log(9007199254740991 + 1); // 9007199254740992`
- `console.log(9007199254740991 + 2); // 9007199254740992`

- *A string in JavaScript must be surrounded by quotes.*
- *let str = "Hello";*
- *let str2 = 'Single quotes are ok too';*
- *let phrase = `can embed another \${str}`;*
- *In JavaScript, there are 3 types of quotes.*
- *Double quotes: "Hello".*
- *Single quotes: 'Hello'.*
- *Backticks: `Hello`.*

- *Double and single quotes are “simple” quotes.*
- *There’s practically no difference between them in JavaScript.*
- *Backticks are “extended functionality” quotes.*
- *They allow us to embed variables and expressions into a string by wrapping them in \${...}, for example:*
- *let name = "John";*
- *// embed a variable*
- *alert(`Hello, \${name}!`); // Hello, John!*
- *// embed an expression*
- *alert(`the result is \${1 + 2}`); // the result is 3*
- *The expression inside \${...} is evaluated and the result becomes a part of the string.*
- *We can put anything in there: a variable like name or an arithmetical expression like 1 + 2 or something more complex.*

- *The expression inside \${...} is evaluated and the result becomes a part of the string. We can put anything in there: a variable like name or an arithmetical expression like 1 + 2 or something more complex.*
- *Please note that this can only be done in backticks. Other quotes don't have this embedding functionality!*
- *alert( "the result is \${1 + 2}" ); // the result is \${1 + 2} (double quotes do nothing)*

- *The boolean type has only two values: true and false.*
- *This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.*
- *For instance:*
- *let nameFieldChecked = true; // yes, name field is checked*
- *let ageFieldChecked = false; // no, age field is not checked*
- *Boolean values also come as a result of comparisons:*
- *let isGreater = 4 > 1;*
- *alert( isGreater ); // true (the comparison result is "yes")*
- *We'll cover booleans more deeply in the chapter Logical operators.*

- *The special null value does not belong to any of the types described above.*
- *It forms a separate type of its own which contains only the null value:*
  
- *let age = null;*
- *In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.*
- *It’s just a special value which represents “nothing”, “empty” or “value unknown”.*
- *The code above states that age is unknown.*

- *The special value undefined also stands apart. It makes a type of its own, just like null.*
- *The meaning of undefined is “value is not assigned”.*
- *If a variable is declared, but not assigned, then its value is undefined:*
- *let age;*
- *alert(age); // shows "undefined"*
- *Technically, it is possible to explicitly assign undefined to a variable:*
- *let age = 100;*
- *// change the value to undefined*
- *age = undefined;*
- *alert(age); // "undefined"*

- *A call to `typeof` x returns a string with the type name:*
- `typeof undefined` // "undefined"
- `typeof 0` // "number"
- `typeof 10n` // "bigint"
- `typeof true` // "boolean"
- `typeof "foo"` // "string"
- `typeof Symbol("id")` // "symbol"
- `typeof Math` // "object" (1)
- `typeof null` // "object" (2)
- `typeof alert` // "function" (3)

# Summary

- *There are 8 basic data types in JavaScript.*
- *Seven primitive data types:*
- *number for numbers of any kind: integer or floating-point, integers are limited by ± (253-1).*
- *bignum for integer numbers of arbitrary length.*
- *string for strings. A string may have zero or more characters, there's no separate single-character type.*
- *boolean for true/false.*
- *null for unknown values – a standalone type that has a single value null.*
- *undefined for unassigned values – a standalone type that has a single value undefined.*
- *symbol for unique identifiers.*
- *And one non-primitive data type:*
- *object for more complex data structures.*

- **Arrays**
- *Objects allow you to store keyed collections of values.*
- *But quite often we find that we need an ordered collection, where we have a 1st, a 2nd, a 3rd element and so on.*
- *For example, we need that to store a list of something: users, goods, HTML elements etc.*
- *It is not convenient to use an object here, because it provides no methods to manage the order of elements.*
- *We can't insert a new property "between" the existing ones. Objects are just not meant for such use.*
- *There exists a special data structure named Array, to store ordered collections.*

- ***Declaration***
- *There are two syntaxes for creating an empty array:*
- *let arr = new Array();*
- *let arr = [];*
- *Almost all the time, the second syntax is used. We can supply initial elements in the brackets:*
- *let fruits = ["Apple", "Orange", "Plum"];*
- *Array elements are numbered, starting with zero.*
- *We can get an element by its number in square brackets:*
- *let fruits = ["Apple", "Orange", "Plum"];*

- `alert(fruits[0]); // Apple`
- `alert(fruits[1]); // Orange`
- `alert(fruits[2]); // Plum`
- *We can replace an element:*
- `fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]`
- *...Or add a new one to the array:*
- `fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]`

- *The total count of the elements in the array is its length:*
- `let fruits = ["Apple", "Orange", "Plum"];`
- `alert(fruits.length); // 3`
- *We can also use alert to show the whole array.*
- `let fruits = ["Apple", "Orange", "Plum"];`
- `alert(fruits); // Apple,Orange,Plum`
- *An array can store elements of any type.*
- *For instance:*
- *// mix of values*
- `let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];`
- *// get the object at index 1 and then show its name*
- `alert( arr[1].name ); // John`
- *// get the function at index 3 and run it*
- `arr[3](); // hello`

- `let fruits = ["Apple", "Orange", "Plum"];`
- `alert(fruits[fruits.length-1]); // Plum`
- `fruits[-1].`
- `fruits.at(-1):`
- `let fruits = ["Apple", "Orange", "Plum"];`
- `// same as fruits[fruits.length-1]`
- `alert(fruits.at(-1)); // Plum`

- *Methods pop/push, shift/unshift*
- *A queue is one of the most common uses of an array.*
- *Ordered collection of elements which supports two operations:*
- *push appends an element to the end.*
- *shift get an element from the beginning, advancing the queue, so that the 2nd element becomes the 1st.*

- *Arrays support both operations.*
- *In practice we need it very often. For example, a queue of messages that need to be shown on-screen.*
- *There's another use case for arrays – the data structure named stack.*
- *It supports two operations:*
- *push adds an element to the end.*
- *pop takes an element from the end.*
- *So new elements are added or taken always from the “end”.*
- *A stack is usually illustrated as a pack of cards: new cards are added to the top or taken from the top:*
- *For stacks, the latest pushed item is received first, that's also called LIFO (Last-In-First-Out) principle. For queues, we have FIFO (First-In-First-Out).*
- *Arrays in JavaScript can work both as a queue and as a stack. They allow you to add/remove elements, both to/from the beginning or the end.*

- *Methods that work with the end of the array:*
- *pop*
- *Extracts the last element of the array and returns it:*
- *let fruits = ["Apple", "Orange", "Pear"];*
- *alert(fruits.pop()); // remove "Pear" and alert it*
- *alert(fruits); // Apple, Orange*
- *Both fruits.pop() and fruits.at(-1) return the last element of the array, but fruits.pop() also modifies the array by removing it.*

- *push*
- *Append the element to the end of the array:*
- *let fruits = ["Apple", "Orange"];*
- *fruits.push("Pear");*
- *alert(fruits); // Apple, Orange, Pear*
- *The call fruits.push(...) is equal to fruits[fruits.length] = ....*

- *Methods that work with the beginning of the array:*
- *Shift:*
- *Extracts the first element of the array and returns it:*
- *let fruits = ["Apple", "Orange", "Pear"];*
- *alert(fruits.shift()); // remove Apple and alert it*
- *alert(fruits); // Orange, Pear*
- *Unshift:*
- *Add the element to the beginning of the array:*
- *let fruits = ["Orange", "Pear"];*
- *fruits.unshift('Apple');*
- *alert(fruits); // Apple, Orange, Pear*
- *Methods push and unshift can add multiple elements at once:*

# JavaScript Variables

- ***Local JavaScript Variables***
- *A variable declared within a JavaScript function becomes LOCAL and can only be accessed within that function. (the variable has local scope).*
- *Local variables are destroyed when you exit the function.*
- ***Global JavaScript Variables***
- *Variables declared outside a function become GLOBAL, and all scripts and functions on the web page can access it.*
- *Global variables are destroyed when you close the page.*

# Rules for JavaScript Variables

- *Can contain any letter of the alphabet, digits 0-9, and the underscore character.*
- *No spaces*
- *No punctuation characters (eg comma, full stop, etc)*
- *The first character of a variable name cannot be a digit.*
- *JavaScript variables' names are case-sensitive. For example `firstName` and `FirstName` are two different variables.*

- *Variables are containers for storing data (values).*
- *There are 3 ways to declare a JavaScript variable:*
- *Using var*
- *Using let*
- *Using const*

- *JavaScript Variables*
- *In a programming language, variables are used to store data values.*
- *JavaScript uses the keywords var, let and const to declare variables.*
- *An equal sign is used to assign values to variables.*
- *In this example, x is defined as a variable. Then, x is assigned (given) the value 6:*
- *let x;*
- *x = 6;*

# Different methods of declaring JavaScript variables

```
// declaring one javascript variable  
var firstName;  
// declaring multiple javascript variables  
var firstName, lastName;  
// declaring and assigning one javascript variable  
var firstName = 'Homer';  
// declaring and assigning multiple javascript variables  
var firstName = 'VESIT', lastName = 'CHEMBUR';
```

- *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.*
- *With let you can not do this:*
- *Example*
- *let x = "John Doe";*
- *let x = 0;*
- *// SyntaxError: 'x' has already been declared*

- ***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:***

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

example

```
let x = 10;  
// Here x is 10  
  
{  
let x = 2;  
// Here x is 2  
}  
  
// Here x is 10
```

## CONST:

- *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.*
- *A const variable cannot be reassigned:*

```
const PI = 3.141592653589793;
```

```
PI = 3.14; // This will give an error
```

```
PI = PI + 10; // This will also give an error
```

- *JavaScript const variables must be assigned a value when they are declared:*

- *Block Scope*
- *Declaring a variable with const is similar to let when it comes to Block Scope.*
- *The x declared in the block, in this example, is not the same as the x declared outside the block:*
- *Example*

```
const x = 10;
```

```
// Here x is 10
```

```
{
```

```
const x = 2;
```

```
// Here x is 2
```

```
}
```

```
// Here x is 10
```

# JavaScript Comments

*Single line:* //

*Multiline comment:* /\* \*/

<html>

<body>

<script type="text/javascript">

// Write a heading

document.write("<h1>This is a heading</h1>");

// Write paragraph

document.write("<p>This is a paragraph.</p>");

</script>

</body>

</html>

# JavaScript Output

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

- *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:

- *Example*

```
<!DOCTYPE html>

<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

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

```
<!DOCTYPE html>

<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

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

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

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

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

- example

- Using `window.alert()`
- You can use an alert box to display data:

- Example

- `<!DOCTYPE html>`

- `<html>`

- `<body>`

- `<h1>My First Web Page</h1>`

- `<p>My first paragraph.</p>`

- `<script>`

- `window.alert(5 + 6);`

- `</script>`

- `</body>`

- `</html>`

- alert

- ***Using console.log()***

For debugging purposes, you can call the `console.log()` method in the browser to display data.

***Example***

```
<!DOCTYPE html>

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

- *Operators:*
- *JavaScript Arithmetic Operators*
- *Arithmetic operators are used to perform arithmetic on numbers:*

<b>Operator</b>	<b>Description</b>
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation ( <a href="#">ES2016</a> )
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

# JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	<code>x = y</code>	<code>x = y</code>
+=	<code>x += y</code>	<code>x = x + y</code>
-=	<code>x -= y</code>	<code>x = x - y</code>
*=	<code>x *= y</code>	<code>x = x * y</code>
/=	<code>x /= y</code>	<code>x = x / y</code>
%=	<code>x %= y</code>	<code>x = x % y</code>
**=	<code>x **= y</code>	<code>x = x ** y</code>

# JavaScript Comparison Operators

Operator	Description
<code>==</code>	equal to
<code>===</code>	equal value and equal type
<code>!=</code>	not equal
<code>!==</code>	not equal value or not equal type
<code>&gt;</code>	greater than
<code>&lt;</code>	less than
<code>&gt;=</code>	greater than or equal to
<code>&lt;=</code>	less than or equal to
<code>?</code>	ternary operator

# JavaScript Logical Operators

<b>Operator</b>	<b>Description</b>
&&	logical and
	logical or
!	logical not

# JavaScript Type Operators

Operator	Description
<code>typeof</code>	Returns the type of a variable
<code>instanceof</code>	Returns true if an object is an instance of an object type

# 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 Assignment Operators

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

# JavaScript Logical Operators

- Given that  $x=6$  and  $y=3$ , the table below explains the logical operators:

Operator	Description	Example
<code>&amp;&amp;</code>	and	$(x < 10 \&\& y > 1)$ is true
<code>  </code>	or	$(x==5 \mid\mid y==5)$ is false
<code>!</code>	not	$!(x==y)$ is true

# JavaScript Comparison Operators

- Given that  $x=5$

Operator	Description	Example
$= =$	is equal to	$x= = 8$ is false $x= = 5$ is true
$= ==$ (strict comparision)	is exactly equal to (value and type)	$x== = 5$ is true $x== = "5"$ is false <a href="#">Example</a> <a href="#">output</a>
$!=$	is not equal	$x!= 8$ is true
$>$	is greater than	$x> 8$ is false
$<$	is less than	$x< 8$ is true
$>=$	is greater than or equal to	$x>= 8$ is false
$<=$	is less than or equal to	$x<= 8$ is true

# Conditional Operator

`variablename=(condition)?value1:value2`

# Escape sequence

Code	Outputs
\'	single quote
\"	double quote
\\"	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace

Example: [esc.html](#) □ [output](#) output

- *Why use == in JavaScript?*
- *Here are the important uses of == in JavaScript:*
- *The == operator is an equality operator.*
- *It checks whether its two operands are the same or not by changing expression from one data type to others.*
- *You can use == operator in order to compare the identity of two operands even though, they are not of a similar type.*
- ***How === Works Exactly?***
- *Strict equality === checks that two values are the same or not.*
- *Value are not implicitly converted to some other value before comparison.*
- *If the variable values are of different types, then the values are considered as unequal.*
- *If the variable are of the same type, are not numeric, and have the same value, they are considered as equal.*
- *Lastly, If both variable values are numbers, they are considered equal if both are not NaN (Not a Number) and are the same value.*

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Operators</h2>
<p>a = 2, b = 5, calculate c = a + b, and display c:</p>
<p id="demonstration"></p>
<script>
var a = 2;
var b = 5;
var c= a + b;
document.getElementById("demonstration").innerHTML = c;
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demonstration"></p>
<script>
var a = 10;
document.getElementById("demonstration").innerHTML = (a == 20);
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var x = 10;
document.getElementById("demo").innerHTML = (x === "10");
</script>
</body>
</html>
```

# JavaScript Popup Boxes

*JavaScript has three kind of popup boxes:*

- *Alert box*
- *Confirm box*
- *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

```
alert('sometext');
```

*Example: [alert.html](#) □ [output](#)*

# 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

```
confirm("sometext");
```

Example: [confirm.html](#) □ [output](#)

## 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

```
prompt("sometext","defaultvalue");
```

Example: [prompt.html](#) □ [output](#)

[Factorial](#)      [output](#)

- *Control Statements:*
- *Sequential*
- *Conditional*
  - If
  - If-else
  - Switch
- *Iterative*
  - For
  - While
  - Do while

# Conditional Statements

- *if statement*
- *if...else statement*
- *if...else if....else statement*
- *switch statement*

# If statement

*if (condition)*

*{*

*code to be executed if condition is true*

*}*

[if.html](#) □ [output](#)

# If...else Statement

*if (condition)*

{

*code to be executed if condition is true*

}

*else*

{

*code to be executed if condition is not true*

}

[If\\_else.html](#) □ [output](#)

# If...else if...else Statement

```
if (condition1)
```

```
{
```

*code to be executed if condition1 is true*

```
}
```

```
else if (condition2)
```

```
{
```

*code to be executed if condition2 is true*

```
}
```

```
else
```

```
{
```

*code to be executed if neither condition1 nor condition2 is true*

```
}
```

[If else if else.html](#) □ [output](#)

# The JavaScript Switch Statement

*switch(n)*

{

*case 1:*

*execute code block 1*

*break;*

*case 2:*

*execute code block 2*

*break;*

*default:*

*code to be executed if n is different from case 1 and 2*

}

1. [Switch.html](#)  [output](#)
2. [Switch1.html](#)  [output](#)

# For loop

```
for (var=start ; var<=end ; var=var+incr)
{
code to be executed
}
```

[For.html](#) □ [output](#)

# The while Loop

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
{
    document.write("The number is " + i);
    document.write("<br />");
    i++;
}
</script>
</body>
</html>
```

# The do...while Loop

```
<html>
<body>
<script type='text/javascript'>
var i=0;
do
{
    document.write("The number is " + i);
    document.write("<br />");
    i++;
}
while (i<=5);
</script>
</body>
</html>
```

# The break Statement

- *The break statement will break the loop and continue executing the code that follows after the loop (if any).*

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
    if (i==3)
    {
        break;
    }
    document.write("The number is " + i);
    document.write("<br />");
}
document.write("The loop break ");
</script>
</body>
</html>
```

# The continue Statement

```
<html>
  <body>
    <script type="text/javascript">
      var i=0
      for (i=0;i<=10;i++)
      {
        if (i==3)
        {
          continue;
        }
        document.write("The number is " + i);
        document.write("<br />");
      }
    </script>
  </body>
</html>
```

# For...In Statement

- *The for...in statement loops through the properties of an object.*

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

var person={fname:“VESIT”, lname:“Chembur”};

for (x in person)
{
document.write(person[x] + " ");
}

</script>
</body>
</html>
```

# Functions

- **Function Definition:**
- **Before we use a function we need to define that function.**  
*The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces. The basic syntax is shown here:*
- `<script type="text/javascript"> <!-- function  
functionname(parameter-list) { statements } //--> </script>`
- Example      output

```
<!DOCTYPE html>
<body>
<script>
function myFunction(a, b) {
    return a * b;
}
var ans=myFunction(5,6)
document.write(ans)
</script>
</body>
</html>
```

### Example

```
//return argument length  
function myFunction(a, b) {  
    return arguments.length;  
}
```

**Click on a button to call a function, which will output "Hello World" in an element with id="demo":**

```
<button onclick="myFunction()">Click me</button>  
<p id="demo"></p>  
<script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello World";  
}  
</script>
```

- *A JavaScript function can also be defined using an expression.*
- *A function expression can be stored in a variable:*

```
var x = function (a, b) {return a * b};
```

```
var x = function (a, b) {return a * b};
```

```
var z = x(4, 3);
```

```
<!DOCTYPE html>
<body>
<script>
var x = function (a, b) {return a * b;};
var z = x(4, 3);
document.write(z)
</script>
</body>
</html>
```

Example  
calculator  
cal Function

Example

calculator

- *The return Statement:*
- *A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.*
- *For example you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.*
- [Example Output](#)
- [calculatorExample output](#)

# Functions

- *We started by using the `function` keyword. This tells the browser that a function is about to be defined.*
- *function definitions are similar to C++/Java, except:*
  - no return type for the function (since variables are loosely typed)
  - no types for parameters (since variables are loosely typed)
  - by-value parameter passing only (parameter gets copy of argument)
- [Functions.html](#) [Functions.html](#)      [output](#)

# Difference between regular functions and arrow functions

- **Arrow functions** – a new feature introduced in ES6 – enable writing concise functions in JavaScript. While both regular and arrow functions work in a similar manner, yet there are certain interesting differences between them, as discussed below.
- **Syntax of regular functions:-**

```
let x = function function_name(parameters){  
    // body of the function  
};
```

```
let square = function(x){  
    return (x*x);  
};  
console.log(square(9));
```

- ***JavaScript Arrow Function***
- Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions.
- For example,
- // function expression

```
let x = function(x, y) {
```

```
    return x * y;
```

```
}
```

- can be written as
- // using arrow functions

```
let x = (x, y) => x * y;
```

- [Arrow.html](#)

- *Arrow Function Syntax*
- *The syntax of the arrow function is:*

```
let myFunction = (arg1, arg2, ...argN) => {  
    statement(s)  
}  
• myFunction is the name of the function  
• arg1, arg2, ...argN are the function arguments  
• statement(s) is the function body
```

- If the body has single statement or expression, you can write arrow function as:
- let myFunction = (arg1, arg2, ...argN) => expression

- ***Example 1: Arrow Function with No Argument***
- If a function doesn't take any argument, then you should use empty parentheses. For example,

```
let greet = () => console.log('Hello');  
greet(); // Hello
```

- ***Example 2: Arrow Function with One Argument***
- ***If a function has only one argument, you can omit the parentheses. For example,***

```
let greet = x => console.log(x);  
greet('Hello'); // Hello
```

- ***Example 3: Arrow Function as an Expression***
- ***You can also dynamically create a function and use it as an expression.***  
***For example,***

```
let age = 5;  
let welcome = (age < 18) ?  
() => console.log('Baby') :  
() => console.log('Adult');  
welcome(); // Baby
```

- *Example 4: Arrow Function with Multiline statement*
- *If a function body has multiple statements, you need to put them inside curly brackets {}. For example,*

```
let sum = (a, b) => {  
    let result = a + b;  
    return result;  
}  
  
let result1 = sum(5,7);  
console.log(result1); // 12
```

*The syntax of arrow functions:-*

```
let x = (parameters) => {  
    // body of the function  
};
```

*Example of arrow functions:-*

```
var square = (x) => {  
    return (x*x);  
};  
console.log(square(9));
```

- *// Traditional Function*

```
function (a){  
    return a + 100;  
}
```

- *// Arrow Function Break Down*

// 1. Remove the word "function" and place arrow between the argument and opening body bracket

```
(a) => {  
    return a + 100;  
}
```

- *// 2. Remove the body braces and word "return" -- the return is implied.*

```
(a) => a + 100;
```

- *// 3. Remove the argument parentheses*

```
a => a + 100;
```

- *// Traditional Function*

```
function (a, b){
```

```
    return a + b + 100;
```

```
}
```

- *// Arrow Function*

```
(a, b) => a + b + 100;
```

- *// Traditional Function (no arguments)*

```
let a = 4;
```

```
let b = 2;
```

```
function (){
```

```
    return a + b + 100;
```

```
}
```

- *// Arrow Function (no arguments)*

```
let a = 4;
```

```
let b = 2;
```

```
() => a + b + 100;
```

- ***JavaScript HTML DOM - Changing CSS***
- The HTML DOM allows JavaScript to change the style of HTML elements.
- 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:
- ***Example***

```
<html>
<body>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
</script>
</body>
</html>
```

# JavaScript Events

- ***JavaScript Events***
- The change in the state of an object is known as an Event.
- In html, there are various events which represents that some activity is performed by the user or by the browser.
- When java script code is included in HTML, java script react over these events and allow the execution.
- This process of reacting over the events is called Event Handling.
- **Thus, java script handles the HTML events via Event Handlers.**
  
- For example, when a user clicks over the browser, add java script code, which will execute the task to be performed on the event.

## Examples of events

- A mouse click
- A web page or an image loading
- Mousing over a spot on the web page
- Selecting an input box in an HTML form
- Submitting an HTML form

- *Some of the HTML events and their event handlers are:*
- **Mouse events:**

<b>Event Performed</b>	<b>Event Handler</b>	<b>Description</b>
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function over() {
          document.write ("Mouse Over");
        }
        function out() {
          document.write ("Mouse Out");
        }
      //-->
    </script>
  </head>
  <body>
    <p>Bring your mouse inside the division to see the result:</p>
    <div onmouseover = "over()" onmouseout = "out()">
      <h2> This is inside the division </h2>
    </div>
  </body>
</html>      example
```

## Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

## Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

[Keyboard Event](#) [Keyboard Event](#)  
[OnFocusEvent](#)

# *Keydown.html*

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()" />
<script>
<!--
function keydownevent()
{
    document.getElementById("input1");
    alert("Pressed a key");
}
//-->
</script>
</body>
</html>
```

## Window/Document events

<b>Event Performed</b>	<b>Event Handler</b>	<b>Description</b>
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

<b>Event handler</b>	<b>Applies to:</b>	<b>triggered when:</b>
<b>onAbort</b>	Image	The loading of the image is cancelled.
<b>onBlur</b>	Button, Checkbox, FileUpload, Layer, password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).
<b>onChange</b>	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
<b>onClick</b>	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
<b>onDbClick</b>	Document, Link	The object is double-clicked on.
<b>onDragDrop</b>	Window	An icon is dragged and dropped into the browser.
<b>onError</b>	Image, Window	A JavaScript error occurs.

<b>Event handler</b>	<b>Applies to:</b>	<b>triggered when:</b>
<b>onFocus</b>	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
<b>onKeyDown</b>	Document, Image, Link, TextArea	The user presses a key.
<b>onKeyPress</b>	Document, Image, Link, TextArea	The user presses or holds down a key.
<b>onKeyUp</b>	Document, Image, Link, TextArea	The user releases a key
<b>onLoad</b>	Image, Window	The whole page has finished loading.
<b>onMouseDown</b>	Button, Document, Link	The user presses a mouse button.
<b>onMouseMove</b>	window	The user moves the mouse.
<b>onMouseOut</b>	Image, Link	The user moves the mouse away from the object.
<b>onMouseOver</b>	Image, Link	The user moves the mouse over the object.

<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onMove</b>	Window	The user moves the browser window or frame.
<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onReset</b>	Form	The user clicks the form's Reset button.
<b>onResize</b>	Window	The user resizes the browser window or frame
<b>onSelect</b>	Text, Textarea	The user selects text within the field
<b>onSubmit</b>	Form	The user clicks the form's Submit button.
<b>onUnload</b>	Window	The user leaves the page.

# Window events

- *onUnload*
- *onLoad*
- *onResize*
- *onMove*
- *onAbort*
- *onError*

*windoweventProgram*      *Output*

# Mouse event

[mouseclickExample](#) [Output](#)

[Mouseclickexample1](#) [Output](#)

[Mousedownexample](#) [Output](#)

[MouseOuteventExample](#) [Output](#)

*Key event*

[Keydownexample](#) [Output](#)

[Keyupexample](#) [Output](#)

[OnfocusExample](#) [Output](#)

[OnblurExample](#) [Output](#)

[LoadEvent](#) [Output](#)

- **HTML DOM Events**
  - **HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.**
  - **Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).**
- **Mouse Events:**

Property	Description
<u>onclick</u>	The event occurs when the user clicks on an element
<u>ondblclick</u>	The event occurs when the user double-clicks on an element
<u>onmousedown</u>	The event occurs when a user presses a mouse button over an element
<u>onmousemove</u>	The event occurs when the pointer is moving while it is over an element
<u>onmouseover</u>	The event occurs when the pointer is moved onto an element
<u>onmouseout</u>	The event occurs when a user moves the mouse pointer out of an element
<u>onmouseup</u>	The event occurs when a user releases a mouse button over an element

# Keyboard Event:

Attribute	Description
<u>onkeydown</u>	The event occurs when the user is pressing a key
<u>onkeypress</u>	The event occurs when the user presses a key
<u>onkeyup</u>	The event occurs when the user releases a key

# Frame Event

Attribute	Description
<u>onabort</u>	The event occurs when an image is stopped from loading before completely loaded (for <object>)
<u>onerror</u>	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)
<u>onload</u>	The event occurs when a document, frameset, or <object> has been loaded
<u>onresize</u>	The event occurs when a document view is resized
<u>onscroll</u>	The event occurs when a document view is scrolled
<u>onunload</u>	The event occurs once a page has unloaded (for <body> and <frameset>)

# Form Event:

Attribute	Description
<u>onblur</u>	The event occurs when a form element loses focus
<u>onchange</u>	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
<u>onfocus</u>	The event occurs when an element gets focus (for <label>, <input>, <select>, <textarea>, and <button>)
<u>onreset</u>	The event occurs when a form is reset
<u>onselect</u>	The event occurs when a user selects some text (for <input> and <textarea>)
<u>onsubmit</u>	The event occurs when a form is submitted

# Image

Property	Description
<u>align</u>	Sets or returns the value of the align attribute of an image
<u>alt</u>	Sets or returns the value of the alt attribute of an image
<u>border</u>	Sets or returns the value of the border attribute of an image
<u>complete</u>	Returns whether or not the browser is finished loading an image
<u>height</u>	Sets or returns the value of the height attribute of an image
<u>hspace</u>	Sets or returns the value of the hspace attribute of an image
<u>longDesc</u>	Sets or returns the value of the longdesc attribute of an image
<u>lowsrc</u>	Sets or returns a URL to a low-resolution version of an image
<u>name</u>	Sets or returns the name of an image
<u>src</u>	Sets or returns the value of the src attribute of an image
<u>useMap</u>	Sets or returns the value of the usemap attribute of an image
<u>vspace</u>	Sets or returns the value of the vspace attribute of an image
<u>width</u>	Sets or returns the value of the width attribute of an image

Event	The event occurs when...
<u>onabort</u>	Loading of an image is interrupted
<u>onerror</u>	An error occurs when loading an image
<u>onload</u>	An image is finished loading

- *Write a java script program to change size of image when we click on change size button.*

### Program Output

- *Write a program to count number of elements with myinput name and display value of first element.*

### Program output

- *Write a program to replace one image by another image when we click on changeimage button.*

### Program output

- *Write a program to change contents of table cells and count number of rows in table when we click on changecontent button.*

### Program Program output

● *Program to display document title and URL*

Program    Output

● *Program to display id , type and button.*

Program Output

● *Program to count number of hyperlinks in the document and display text of first hyperlink.*

Program Output

● *Program to display date*

program Output

Program1    Output

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
function clickevent()
{
    document.write("This is JavaTpoint");
}
//-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?" />
</form>
</body>
</html>
```

**ClickEvent**

[addtwonumberform.html](#)

# What is DOM?

## The DOM is:

- The Document Object Model (DOM) is a programming interface that defines the logical structure of documents and the way a document is accessed and manipulated.
- A standard Document object model for HTML
- A standard programming interface for HTML
- Platform- and language-independent
- A W3C standard (World Wide Web Consortium) that allows programs and scripts to dynamically access and update the content, structure, and style of a document.
- The HTML DOM defines a standard way for accessing and manipulating HTML documents.
- The DOM presents an HTML document as a tree-structure.

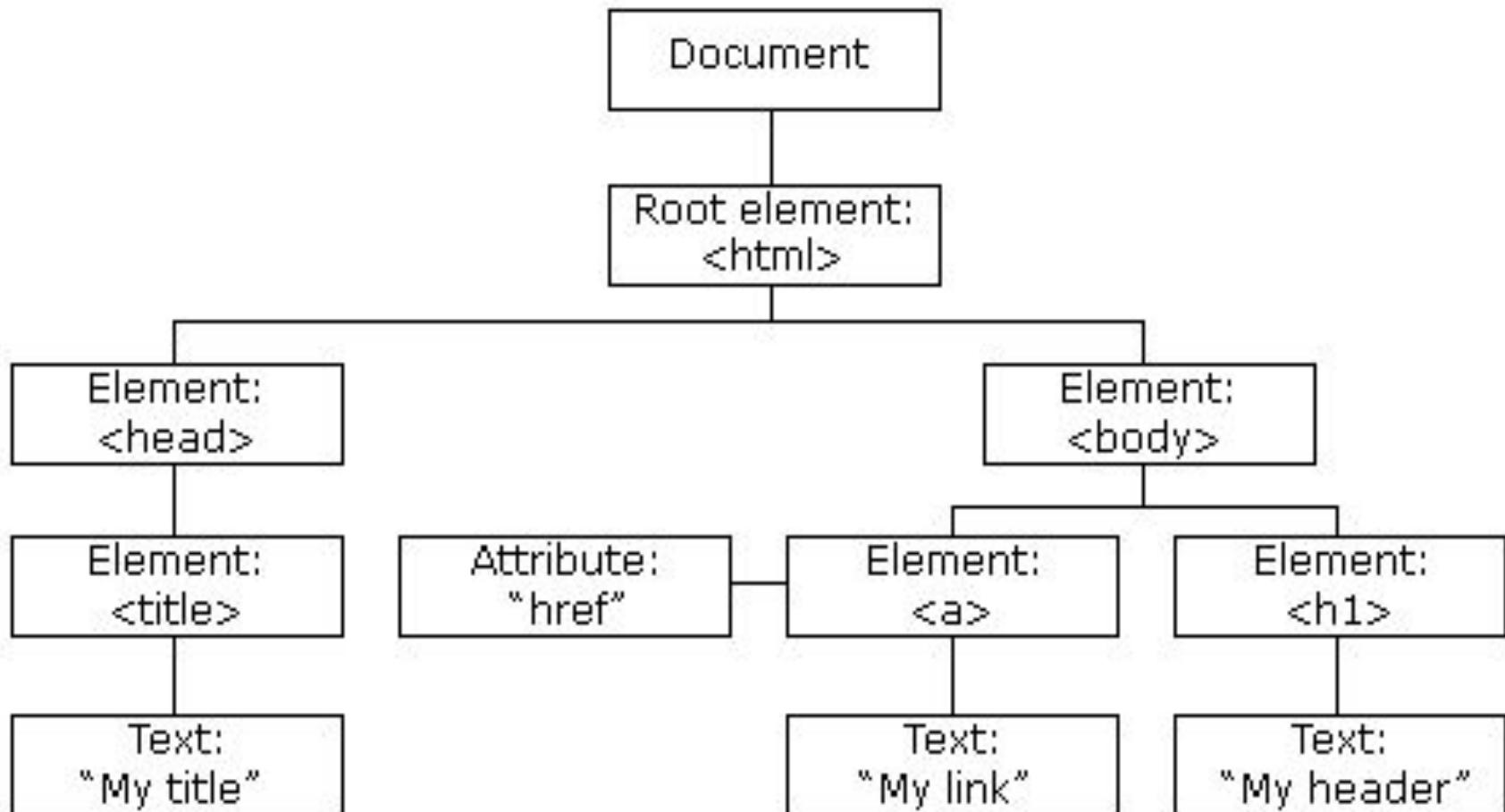
- *The HTML DOM (Document Object Model)*
- *When a web page is loaded, the browser creates a Document Object Model of the page.*
- *The HTML DOM model is constructed as a tree of Objects:*
- *The HTML DOM is a standard object model and programming interface for HTML. It defines:*
- *The HTML elements as objects*
- *The properties of all HTML elements*
- *The methods to access all HTML elements*
- *The events for all HTML element*

# Document Object Model

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
  - Your web browser builds a *model* of the web page (the *document*) that includes all the *objects* in the page (tags, text, etc)
  - All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
  - Those objects are accessible via scripting languages in modern web browsers

# The HTML DOM Node Tree (Document Tree)

- *The HTML DOM views a HTML document as a tree-structure. The tree structure is called a node-tree.*



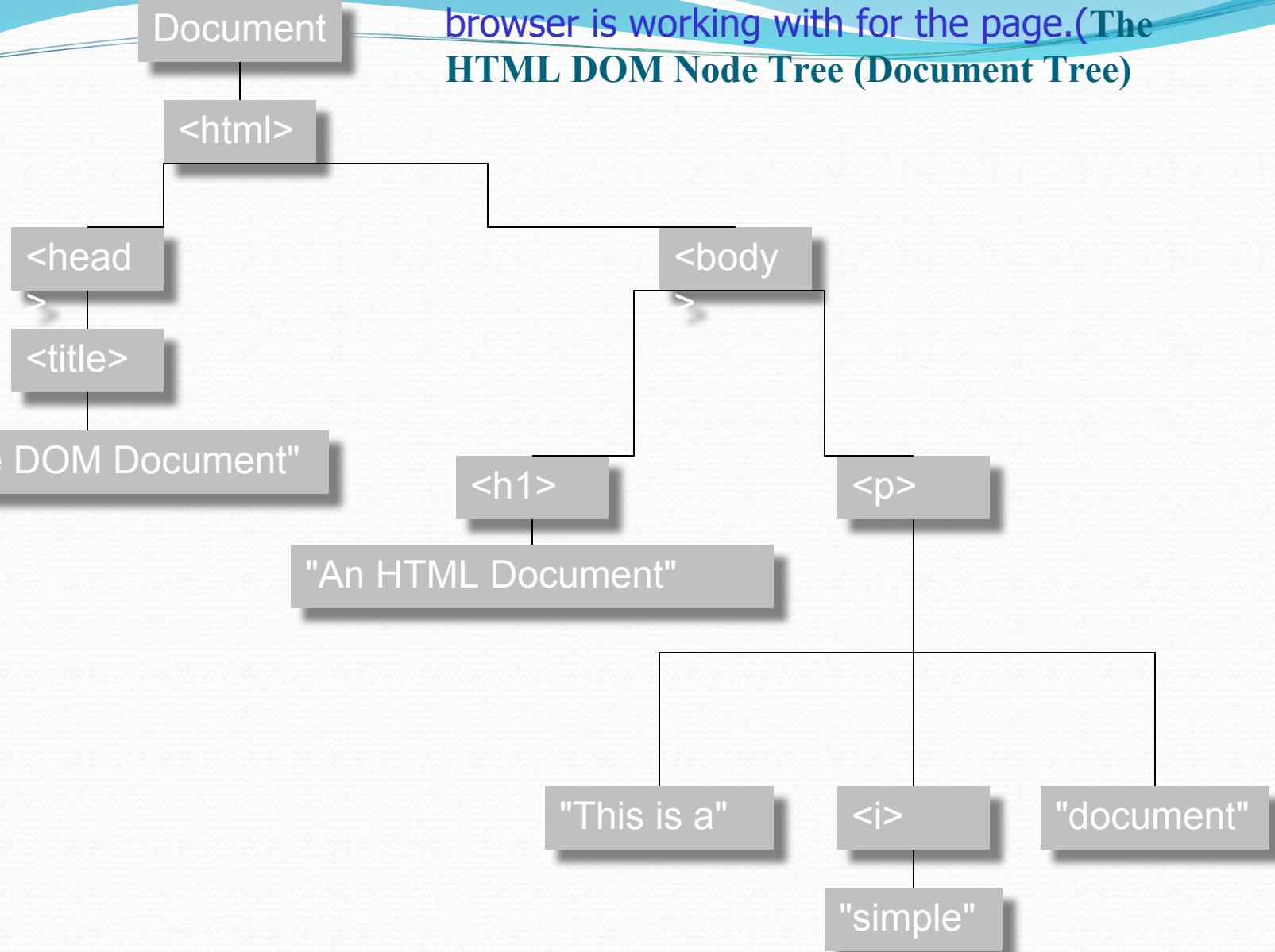
This is what the browser reads

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

This is what the browser displays on screen.



This is a drawing of the model that the browser is working with for the page.(The **HTML DOM Node Tree (Document Tree)**)



# HTML DOM

*The DOM says:*

- *The entire document is a document node*
- *Every HTML tag is an element node*
- *The text in the HTML elements are text nodes*
- *Every HTML attribute is an attribute node*
- *Comments are comment nodes*
- *In the HTML DOM the value of the text node can be accessed by the innerHTML property.*
- *Example : innerHTML □ output*

```
<html>
<body>
<p id="intro">Hi World!</p>
<p id="intro">Hello World!</p>
<p id="main">HTML DOM tutorial</p>
<script type="text/javascript">
txt=document.getElementById("intro").innerHTML;
document.write("Paragraph: " + txt);
</script>
</body>
</html>
```

**DOM references describe the properties and methods of the HTML DOM.**

HTML Document

HTML Element

HTML Attribute

HTML Events

The HTML DOM is a standard **object** model and **programming interface** for HTML.

It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

- **HTML DOM Nodes**

In the HTML DOM (Document Object Model), everything is a **node**:

- The document itself is a document node
- All HTML elements are element nodes
- All HTML attributes are attribute nodes
- Text inside HTML elements are text nodes
- Comments are comment nodes

- **The 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 and the "owner" of all other nodes:

(element nodes, text nodes, attribute nodes, and comment nodes).

The document object provides properties and methods to access all node objects, from within JavaScript

## *Example*

- *The JavaScript code to get the text from a <p> element with the id "intro" in a HTML document :*
- *txt=document.getElementById("intro").innerHTML*
- *After the execution of the statement, txt will hold the value "Hi World!"*

## *Explanation:*

- *document - the current HTML document*
- *getElementById("intro") - the <p> element with the id "intro"*
- *innerHTML - the inner text of the HTML element*
- *In the example above, getElementById is a method, while innerHTML is a property.*

# The HTML DOM Node Tree

- *The programming interface to the DOM is defined by a set of standard properties and methods.*
- *Properties are often referred to as something that is.*
  - nodename is "p"
- *Methods are often referred to as something that is done.*
  - delete "p"

## *HTML DOM Properties*

x.innerHTML - the inner text value of x (a HTML element)

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x

Note: In the list above, x is a node object (HTML element).

# HTML DOM Methods

- ***x.getElementById(id)*** - *get the element with a specified id*
- ***x.getElementsByTagName(name)*** - *get all elements with a specified tag name*
- ***x.appendChild(node)*** - *insert a child node to x*
- ***x.removeChild(node)*** - *remove a child node from x*

**Note:** In the list above, x is a node object (HTML element).

**Example :** [node.html](#) □ [output](#)

[Example1](#) [output](#)

[Example2](#) [Output](#)

# Changing an HTML Element

- *The HTML DOM and JavaScript can be used to change the inner content and attributes of HTML elements dynamically.*
- *Example 1 - Change the Background Color*

*The following example changes the background color of the <html>*

```
<body> <p> VESIT CHEMBUR !</p>
<script type="text/javascript">
document.body.bgColor='yellow';
</script>
</body>
</html>
```

# Changing the Text HTML Element - innerHTML

- ◆ *The easiest way to get or modify the content of an element is by using the innerHTML property.*
- ◆ *Example 2 – Change the Text of an Element*

```
<html>
<body>
<p id="p1">Hello World!</p>
  <script type="text/javascript">
document.getElementById("p1").innerHTML=“Visit
Chembur”; </script>
</body>
</html>
```

# Changing an HTML Element Using Events

- An event handler allows you to execute code when an event occurs.
- Events are generated by the browser when the user clicks an element, when the page loads, when a form is submitted, etc.

Example 3 - Change the Background Color Using onclick Event

```
<html> <body>
    <script type="text/javascript">
        document.body.bgColor="yellow";
    </script>
<input type="button"
onclick="document.body.bgColor='blue';" value="Click me to
change background color">
</body> </html>
```

# Change the Text font and color of an Element

- ✓ *The following example uses a function to change the style of the <p> element when the button is clicked :*
- ✓ [chgtextrfont.html](#) □ [output](#)

## ● *HTML DOM Objects Reference*

- *The references describe the properties and methods of each object, along with examples.*
- *Document object*
- *Event object*
- *HTMLElement object*
- *Anchor object*
- *Area object*
- *Base object*
- *Body object*
- *Button object*
- *Form object*
- *Frame/IFrame object*
- *Frameset object*

- *Image object*
- *Input Button object*
- *Input Checkbox object*
- *Input File object*
- *Input Hidden object*
- *Input Password object*
- *Input Radio object*
- *Input Reset object*
- *Input Submit object*
- *Input Text object*
- *Link object*
- *Meta object*
- *Object object*

- *Option object*
- *Select object*
- *Style object*
- *Table object*
- *td / th object*
- *tr object*
- *Textarea object*

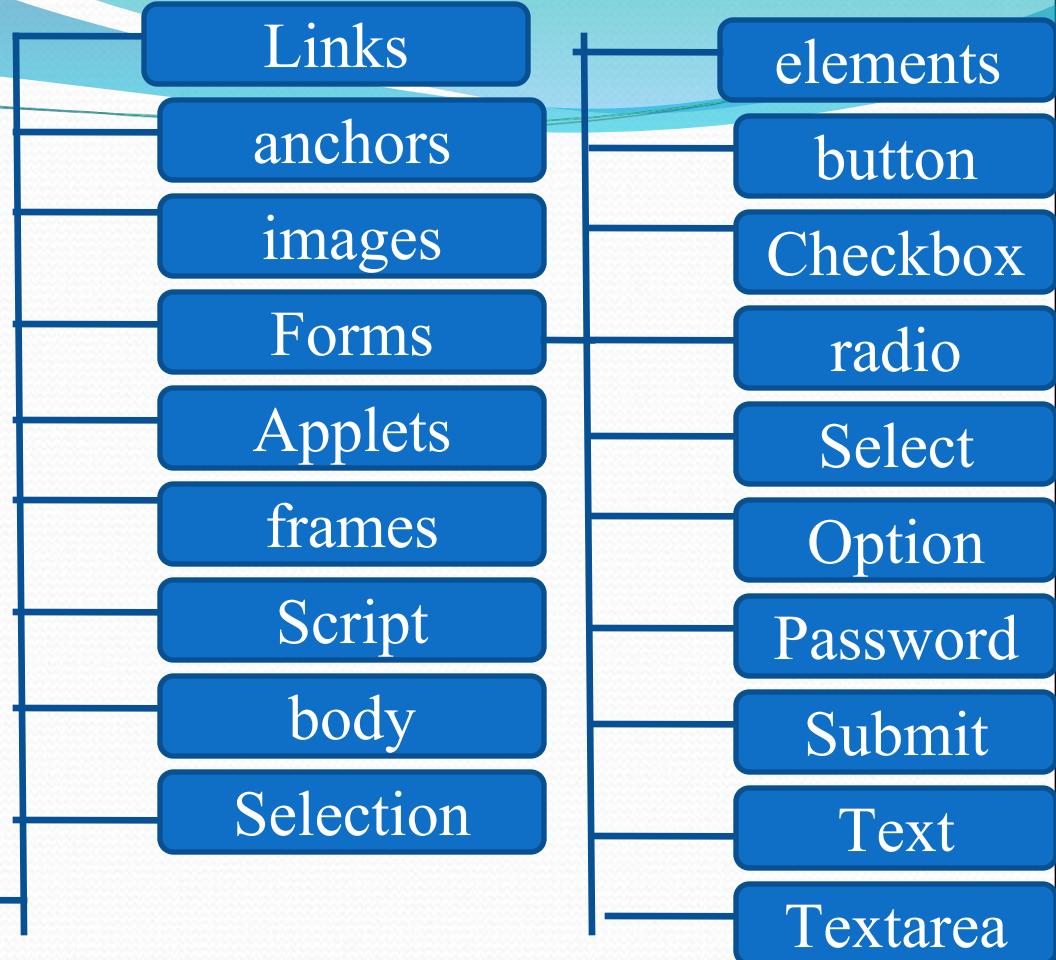
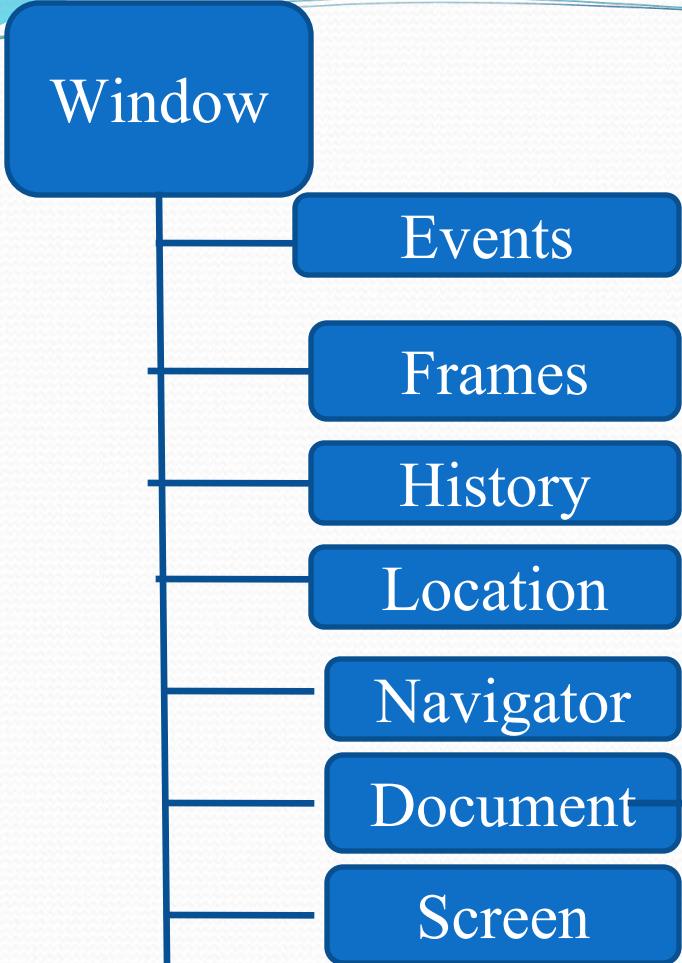
- The DOM refers to a hierarchical collection of objects that allow you to work with HTML documents.
- Java script supports various HTML DOM objects. “
- The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.”
- The W3C DOM standard is separated into 3 different parts:
- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

- *Main Purpose of Browser object is provide an application programming interface between web browser software and Java script language.*
- *Java script statements can directly access and update properties of the Browser objects and call it's methods.*
- *The parent is the window object.*
- *The window.location contains URL of the web page currently displayed in the browser window.*
- *Window.history stores list of URLs that have been stored in window.location.*
- *Window.navigator stores information about browser type,browser version etc.*
- *Window.document refers to actual web page.*
  - **Window.document.images object returns array of all images.**

# JavaScript HTML BROWSER OBJECT MODEL

## Objects

- ✓ **Object Description :**
- ✓ **Window** : *The top level object in the DOM hierarchy. The Window object represents a browser window. A Window object is created automatically with every instance of a <body> or <frameset> tag.*
- ✓ *The Window object and contains in itself several other objects, such as "document", "history" etc.*
- ✓ **Navigator** : *Contains information about the client's display screen.*
- ✓ **History** : *Contains the visited URLs in the browser window.*
- ✓ **Location** : *Contains information about the current URL.*
- ✓ **Example** : [doc.html](#) □ [output](#)



## Browser Object

[Example Output](#)

[Example1 Output](#)

Object	Description	Properties	Methods
Window	Represents the Web Browser Window	Defaultstatus,name, status,self,closed,history,location,document.	Alert(),close(),confirm(),focus(),open(),print(),prompt()
Navigator	Allows you to access various information about the user's web browser.	AppCodeName,appName,appVersion,cookieEnabled,platform	javaEnabled()
Screen	Allows you to access different information about the user screen.	AvailHeight,availWidth,colorDepth,height,width	
Location	Allows you to access information about the URL of the web page that is currently opened in the web browser	Host,hostname,href,pathname,port,protocol,search.	Assign(),reload(),replace()
Document	Represents the HTML document or web page that is currently opened in the web page.	Lastmodified,title,url	Close(),getElementById(),getElementByName(),getElementByTagName(),open(),write(),writeln()

# Document Object Methods

Method	Description
<u>close()</u>	Closes the output stream previously opened with document.open()
<u>getElementsByName()</u>	Accesses all elements <b>with</b> a specified name
<u>open()</u>	Opens an output stream to collect the output from document.write() or document.writeln()
<u>write()</u>	Writes <b>HTML</b> expressions or <b>JavaScript</b> code to a document
<u>writeln()</u>	Same as write(), but adds a <b>newline</b> character after each statement

# Document Object Properties

Property	Description
<u>anchors</u>	Returns a collection of all the anchors in the document
<u>applets</u>	Returns a collection of all the applets in the document
<u>body</u>	Returns the body element of the document
<u>cookie</u>	Returns all name/value pairs of cookies in the document
<u>documentMode</u>	Returns the mode used by the browser to render the document
<u>domain</u>	Returns the domain name of the server that loaded the document
<u>forms</u>	Returns a collection of all the forms in the document
<u>images</u>	Returns a collection of all the images in the document
<u>lastModified</u>	Returns the date and time the document was last modified
<u>links</u>	Returns a collection of all the links in the document
<u>readyState</u>	Returns the (loading) status of the document
<u>referrer</u>	Returns the URL of the document that loaded the current document
<u>title</u>	Sets or returns the title of the document
<u>URL</u>	Returns the full URL of the document

## ● *Body Object*

- *The Body object represents the HTML body element.*
- *The body element defines a document's body.*
- *The body element contains all the contents of an HTML document, such as text, hyperlinks, images, tables, lists, etc.*

## ● *Body Object Properties*

Property	Description
<u>aLink</u>	Sets or returns the value of the alink attribute of the body element
<u>background</u>	Sets or returns the value of the background attribute of the body element
<u>bgColor</u>	Sets or returns the value of the bgcolor attribute of the body element
<u>link</u>	Sets or returns the value of the link attribute of the body element
<u>text</u>	Sets or returns the value of the text attribute of the body element
<u>vLink</u>	Sets or returns the value of the vlink attribute of the body element

# Window Object

- *The window object is primary point from where most objects come.*
- document.write("HELLO") Is actually
- Windows.document.write("HELLO")

Window1.html  Output

1) Window2.html  Output

2) Window3.html  Output

Properties	Methods
Defaultstatus, name, status, self, closed, history, location, document.	Alert(), close(), confirm(), focus(), open(), print(), prompt()

# Navigator object

- *The navigator object contains information about the browser.*  
*Example : [navig.html](#) □ [output](#)*

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
platform	Returns for which platform the browser is compiled

# Location Object

- *The location object contains information about the current URL.*
- *The location object is part of the window object and is accessed through the window.location property.*
- [location.html](#) [location.html - output](#)

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

# Screen Object

- *The screen object contains information about the visitor's screen.*
- [Screen.html](#) □ [output](#)

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

# History Object

- *The history object contains the URLs visited by the user (within a browser window).*
- *The history object is part of the window object and is accessed through the window.history property.*
- *History Object Properties :*
  - Length : Returns the number of URLs in the history list
- *History Object Methods :*
  - back() : Loads the previous URL in the history list
  - forward() : Loads the next URL in the history list
  - go() : Loads a specific URL from the history list

# Events

- *By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.*
- *Every element on a web page has certain events which can trigger JavaScript functions.*

# Examples of events

- *A mouse click*
- *A web page or an image loading*
- *Mousing over a spot on the web page*
- *Selecting an input box in an HTML form*
- *Submitting an HTML form*
- *A keystroke*

<b>Event handler</b>	<b>Applies to:</b>	<b>triggered when:</b>
<b>onAbort</b>	Image	The loading of the image is cancelled.
<b>onBlur</b>	Button, Checkbox, FileUpload, Layer, password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).
<b>onChange</b>	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
<b>onClick</b>	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
<b>onDbClick</b>	Document, Link	The object is double-clicked on.
<b>onDragDrop</b>	Window	An icon is dragged and dropped into the browser.
<b>onError</b>	Image, Window	A JavaScript error occurs.

<b>Event handler</b>	<b>Applies to:</b>	<b>triggered when:</b>
<b>onFocus</b>	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
<b>onKeyDown</b>	Document, Image, Link, TextArea	The user presses a key.
<b>onKeyPress</b>	Document, Image, Link, TextArea	The user presses or holds down a key.
<b>onKeyUp</b>	Document, Image, Link, TextArea	The user releases a key
<b>onLoad</b>	Image, Window	The whole page has finished loading.
<b>onMouseDown</b>	Button, Document, Link	The user presses a mouse button.
<b>onMouseMove</b>	window	The user moves the mouse.
<b>onMouseOut</b>	Image, Link	The user moves the mouse away from the object.
<b>onMouseOver</b>	Image, Link	The user moves the mouse over the object.

<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onMove</b>	Window	The user moves the browser window or frame.
<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onReset</b>	Form	The user clicks the form's Reset button.
<b>onResize</b>	Window	The user resizes the browser window or frame
<b>onSelect</b>	Text, Textarea	The user selects text within the field
<b>onSubmit</b>	Form	The user clicks the form's Submit button.
<b>onUnload</b>	Window	The user leaves the page.

# Window events

- *onUnload*
- *onLoad*
- *onResize*
- *onMove*
- *onAbort*
- *onError*

*windoweventProgram*

*Output*

# Mouse event

[mouseclickExample](#) [Output](#)

[Mouseclickexample1](#) [Output](#)

[Mousedownexample](#) [Output](#)

[MouseOuteventExample](#) [Output](#)

## Key event

[Keydownexample](#) [Output](#)

[Keyupexample](#) [Output](#)

[OnfocusExample](#) [Output](#)

[OnblurExample](#) [Output](#)

- **HTML DOM Events**
  - **HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.**
  - **Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).**
- **Mouse Events:**

Property	Description
<u>onclick</u>	The event occurs when the user clicks on an element
<u>ondblclick</u>	The event occurs when the user double-clicks on an element
<u>onmousedown</u>	The event occurs when a user presses a mouse button over an element
<u>onmousemove</u>	The event occurs when the pointer is moving while it is over an element
<u>onmouseover</u>	The event occurs when the pointer is moved onto an element
<u>onmouseout</u>	The event occurs when a user moves the mouse pointer out of an element
<u>onmouseup</u>	The event occurs when a user releases a mouse button over an element

# Keyboard Event:

Attribute	Description
<u>onkeydown</u>	The event occurs when the user is pressing a key
<u>onkeypress</u>	The event occurs when the user presses a key
<u>onkeyup</u>	The event occurs when the user releases a key

# Frame Event

Attribute	Description
<u>onabort</u>	The event occurs when an image is stopped from loading before completely loaded (for <object>)
<u>onerror</u>	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)
<u>onload</u>	The event occurs when a document, frameset, or <object> has been loaded
<u>onresize</u>	The event occurs when a document view is resized
<u>onscroll</u>	The event occurs when a document view is scrolled
<u>onunload</u>	The event occurs once a page has unloaded (for <body> and <frameset>)

# Form Event:

Attribute	Description
<u>onblur</u>	The event occurs when a form element loses focus
<u>onchange</u>	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
<u>onfocus</u>	The event occurs when an element gets focus (for <label>, <input>, <select>, <textarea>, and <button>)
<u>onreset</u>	The event occurs when a form is reset
<u>onselect</u>	The event occurs when a user selects some text (for <input> and <textarea>)
<u>onsubmit</u>	The event occurs when a form is submitted

# Image

Property	Description
<u>align</u>	Sets or returns the value of the align attribute of an image
<u>alt</u>	Sets or returns the value of the alt attribute of an image
<u>border</u>	Sets or returns the value of the border attribute of an image
<u>complete</u>	Returns whether or not the browser is finished loading an image
<u>height</u>	Sets or returns the value of the height attribute of an image
<u>hspace</u>	Sets or returns the value of the hspace attribute of an image
<u>longDesc</u>	Sets or returns the value of the longdesc attribute of an image
<u>lowsrc</u>	Sets or returns a URL to a low-resolution version of an image
<u>name</u>	Sets or returns the name of an image
<u>src</u>	Sets or returns the value of the src attribute of an image
<u>useMap</u>	Sets or returns the value of the usemap attribute of an image
<u>vspace</u>	Sets or returns the value of the vspace attribute of an image
<u>width</u>	Sets or returns the value of the width attribute of an image

Event	The event occurs when...
<u>onabort</u>	Loading of an image is interrupted
<u>onerror</u>	An error occurs when loading an image
<u>onload</u>	An image is finished loading

- *Write a java script program to change size of image when we click on change size button.*

### Program Output

- *Write a program to count number of elements with myinput name and display value of first element.*

### Program output

- *Write a program to replace one image by another image when we click on changeimage button.*

### Program output

- *Write a program to change contents of table cells and count number of rows in table when we click on changecontent button.*

### Program Program output

● *Program to display document title and URL*

Program    Output

● *Program to display id , type and button.*

Program Output

● *Program to count number of hyperlinks in the document and display text of first hyperlink.*

Program Output

● *Program to display date*

program Output

Program1    Output

- *JavaScript Form Validation*
- *Example of JavaScript validation*
- *JavaScript email validation*
- *It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.*
- *JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation.*
- *Most of the web developers prefer JavaScript form validation.*
- *Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.*

- **JavaScript Form Validation Example**
- *In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.*
- *Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.*

```
<script>  
function validateform(){  
var name=document.myform.name.value;  
var password=document.myform.password.value;  
if(name==null || name==""){  
alert("Name can't be blank");  
return false;  
}else if(password.length<6){  
alert("Password must be at least 6 characters long.");  
return false;  
}  
}  
</script>
```

```
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return
validateform()">
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

## ~~JavaScript Retype Password Validation~~

```
<script type="text/javascript">  
function matchpass(){  
var firstpassword=document.f1.password.value;  
var secondpassword=document.f1.password2.value;  
  
if(firstpassword==secondpassword){  
return true;  
}  
else{  
alert("password must be same!");  
return false;  
}  
}  
</script>
```

```
<form name="f1" action="register.jsp" onsubmit="return matchpass()">  
Password:<input type="password" name="password" /><br/>  
Re-enter Password:<input type="password" name="password2"/><br/>  
<input type="submit">  
</form>
```

- ***JavaScript Number Validation***
- ***Let's validate the textfield for numeric value only. Here, we are using isNaN() function.***

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
document.getElementById("numloc").innerHTML="Enter Numeric value
only";
return false;
} else{
return true;
}
}
</script>
```

- ***JavaScript email validation***
- *We can validate the email by the help of JavaScript.*
- *There are many criteria that need to be follow to validate the email id such as:*
- *email id must contain the @ and . character*
- *There must be at least one character before and after the @.*
- *There must be at least two characters after . (dot).*

```
<form name="myform" onsubmit="return validate()">  
Number: <input type="text" name="num"><span  
id="numloc"></span><br/>  
<input type="submit" value="submit">  
</form>
```

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
    alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n
dotposition:"+dotposition);
    return false;
}
</script>
```

```
<body>
<form name="myform" method="post" action="#" onsubmit="return
validateemail();">
Email: <input type="text" name="email"><br/>
<input type="submit" value="register">
</form>
```

- ***Phone Number validation***
- *The validating phone number is an important point while validating an HTML form.*
- *In this page we have discussed how to validate a phone number (in different format) using JavaScript :*
- *At first, we validate a phone number of 10 digits with no comma, no spaces, no punctuation and there will be no + sign in front the number.*
- *Simply the validation will remove all non-digits and permit only phone numbers with 10 digits. Here is the function.*

```
function phonenumber(inputtxt)
{
var phoneno = /^d{10}$/;
if((inputtxt.value.match(phoneno))
{
return true;
}
else
{
alert("message");
return false;
}
}
```

```
function phonenumber(inputtxt)
{
var phoneno = /^(\d{3})\d{3}(\d{4})$/;
if((inputtxt.value.match(phoneno))
{
return true;
}
else
{
alert("message");
return false;
}
}
```

- *Validate a 10 digit phone number*
- *At first we validate a phone number of 10 digit. For example 1234567890, 0999990011, 8888855555 etc.*
- *HTML Code*

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript form validation - checking non-empty</title>
<link rel='stylesheet' href='form-style.css' type='text/css' />
</head><body onload='document.form1.text1.focus()'>
<div class="mail">
<h2>Input an Phone No.[xxxxxxxxxx] and Submit</h2>
<form name="form1" action="#">
<ul>
<li><input type='text' name='text1' /></li>
<li>&nbsp;</li>
<li class="submit"><input type="submit" name="submit" value="Submit"
onclick="phonenumber(document.form1.text1)"/></li>
<li>&nbsp;</li>
</ul>
</form>
```

```
</div>  
<script src="phoneno-all-numeric-validation.js"></script>  
</body>  
</html>
```

```
function phonenumber(inputtxt)
{
    var phoneno = /^d{10}$/;
    if(inputtxt.value.match(phoneno))
    {
        return true;
    }
    else
    {
        alert("Not a valid Phone Number");
        return false;
    }
}
```

- *JavaScript provides in-built functions to check whether the array is empty or not. Following are the method offered by JavaScript programming to check an empty array:*
- *length*
- *isArray(array)*
- *The Array.isArray() function checks the array type (passed parameter is an array or not) and array.length find the length of the array.*

- ***.length property***
- *The length property returns the length of the array by which you can determine whether the array is empty or not.*
- *This property is directly used with the name of array concatenated by dot (.) operator, e.g., arr1.length.*
- **Syntax**  
*array.length*
- *If the length returned by this property is 0, it refers to true means the array is empty. Otherwise, the array is not empty if it returns a non-zero value.*

```
<script>
```

```
var arr1 = [15, 78, 24, 89, 23];
```

```
var arr2 = [];
```

```
//check second array (arr2) length
```

```
if(arr1.length == 0)
```

```
    document.write("arr1 is empty <br>");
```

```
else
```

```
    document.write("arr1 is not empty <br>");
```

```
//check second array (arr2) length
```

```
if(arr2.length == 0)
```

```
    document.write("arr2 is empty <br>");
```

```
else
```

```
    document.write("arr2 is not empty <br>");
```

```
</script>
```

- *In JavaScript, arrays are objects.*
- *So, if you check the type of array using typeof property, it will return value as an object.*
- *But now we have Array.isArray() function to check the type of array, which can be used with .length property to check empty array.*
- *Syntax*
- *Array.isArray(arr1)*
- *It returns a Boolean value, either true or false.*

```
<html>
<head>
<script type="text/javascript">
function Focus_In (event) {
    event.srcElement.style.color = "red";
}
function Focus_Out (event) {
    event.srcElement.style.color = "blue";
}
</script>
</head>
```

```
<body>
<center>  <h3>Click on the textbox and observe </h3><br><br>
<form onfocusin="Focus_In (event)" onfocusout="Focus_Out (event)">
<b>  User name: <input type="text value="Username"/></b><br/>
<br>
<b>  Password <input type="password" value="*****"/></b>
</center>
</form>
</body>
</html>
```

- ***JavaScript eval() function***
- *The eval() function in JavaScript is used to evaluate the expression.*
- *It is JavaScript's global function, which evaluates the specified string as JavaScript code and executes it.*
- *The parameter of the eval() function is a string.*
- *If the parameter represents the statements, eval() evaluates the statements.*
- *If the parameter is an expression, eval() evaluates the expression.*
- *If the parameter of eval() is not a string, the function returns the parameter unchanged.*
- *There are some limitations of using the eval() function, such as the eval() function is not recommended to use because of the security reasons.*
- *It is not suggested to use because it is slower and makes code unreadable.*

- **Syntax**
- *eval(string)*
- *Values*
- *It accepts a single parameter, which is defined as follows.*
- *string: It represents a JavaScript expression, single statement, or the sequence of statements.*
- *It can be a variable, statement, or a JavaScript expression.*
- *Let's understand the JavaScript eval() function by using illustrations.*

```
<html>
<head>
<script>
var a = 10, b = 20, c = 30, sum, mul, sub;
sum = eval("a + b + c");
mul = eval("a * b * c");
sub = eval("a - b");
document.write(sum + "<br>");
document.write(mul + "<br>");
document.write(sub);
</script>
</head>
<body>
</body>
</html>
```

## ***• Exception Handling in JavaScript***

- *An exception signifies the presence of an abnormal condition which requires special operable techniques.*
- *In programming terms, an exception is the anomalous code that breaks the normal flow of the code.*
- *Such exceptions require specialized programming constructs for its execution.*
  
- ***What is Exception Handling***
- *In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them.*
- *It also enables to handle the flow control of the code/program.*
- *For handling the code, various handlers are used that process the exception and execute the code.*
- *For example, the Division of a non-zero value with zero will result into infinity always, and it is an exception.*
- *Thus, with the help of exception handling, it can be executed and handled.*

- ***In exception handling:***
- ***A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.***
- ***The thrown exception is handled by wrapping the code into the try...catch block. If an error is present, the catch block will execute, else only the try block statements will get executed.***
- ***Types of Errors***
- ***While coding, there can be three types of errors in the code:***
- ***Syntax Error:*** *When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.*
- ***Runtime Error:*** *When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.*
- ***Logical Error:*** *An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.*

- **Error Object**
- *When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:*
- *name: This is an object property that sets or returns an error name.*
- *message: This property returns an error message in the string form.*
- *Although Error is a generic constructor, there are following standard built-in error types or error*

- ***Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:***
- ***EvalError: It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.***
- ***InternalError: It creates an instance when the js engine throws an internal error.***
- ***RangeError: It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.***
- ***ReferenceError: It creates an instance for the error that occurs when an invalid reference is de-referenced.***
- ***SyntaxError: An instance is created for the syntax error that may occur while parsing the eval().***
- ***TypeError: When a variable is not a valid type, an instance is created for such an error.***
- ***URIError: An instance is created for the error that occurs when invalid parameters are passed in encodeURI() or decodeURI().***

- ***Exception Handling Statements***
- *There are following statements that handle if any exception occurs:*
- *throw statements*
- *try...catch statements*
- *try...catch...finally statements.*

- **JavaScript try...catch**
- *A try...catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code.*
- *It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur).*
- *A good programming approach is to keep the complex code within the try...catch statements.*
- *try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error.*
- *Otherwise, it executes the code written within.*
- *catch{} statement: This block handles the error of the code by executing the set of statements written within the block.*
- *This block contains either the user-defined exception handler or the built-in handler.*
- *This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.*

- **Syntax:**

*try{  
expression; } //code to be written.*

*catch(error){  
expression; } // code for handling the error.*

*try...catch example*

*<html>*

*<head> Exception Handling</br></head>*

*<body>*

*<script>*

```
try{  
var a= ["34","32","5","31","24","44","67"]; //a is an array  
document.write(a); // displays elements of a  
document.write(b); //b is undefined but still trying to fetch its value. Thus  
catch block will be invoked  
}catch(e){  
alert("There is error which shows "+e.message); //Handling error  
}  
</script>  
</body>  
</html>
```

- **Throw Statement**
- *Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors.*
- *When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.*
- **Syntax:**
- ***throw exception;***

*try...catch...throw syntax*

*try{*

*throw exception; // user can define their own exception*

*}*

*catch(error){*

*expression; } // code for handling exception.*

- ***throw example with try...catch***

```
<html>
<head>Exception Handling</head>
<body>
<script>
try {
  throw new Error('This is the throw keyword'); //user-defined throw
  statement.
}
catch (e) {
  document.write(e.message); // This will generate an error message
}
</script>
</body>
</html>
```

- ***try...catch...finally statements***
- *Finally is an optional block of statements which is executed after the execution of try and catch statements.*
- *Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute.*
- *It does not care for the output too.*
- ***Syntax:***

```
try{  
    expression;  
}  
catch(error){  
    expression;  
}  
finally{  
    expression; } //Executable code
```

- *try...catch...finally example*

```
<html>
<head>Exception Handling</head>
<body>
<script>
try{
var a=2;
if(a==2)
document.write("ok");
}
catch(Error){
document.write("Error found"+e.message);
}
finally{
document.write("Value of a is 2 ");
}
</script>
</body>
</html>
```

```
<html>
<head>
<script>
function ageCalculator()
{
    var userinput = document.getElementById("DOB").value;
    var dob = new Date(userinput);
    if(userinput==null || userinput=='')
    {
        document.getElementById("message").innerHTML = "***Choose a date
please!";
        return false;
    }
    else
    {
        //calculate month difference from current date in time
        var month_diff = Date.now() - dob.getTime();
    }
}
```

```
//convert the calculated difference in date format  
var age_dt = new Date(month_diff);  
  
//extract year from date  
var year = age_dt.getUTCFullYear();  
  
//now calculate the age of the user  
var age = Math.abs(year - 1970);  
  
//display the calculated age  
return document.getElementById("result").innerHTML =  
    "Age is: " + age + "years. ";  
}  
}
```

```
</script>
</head>
<body>
<center>
<h2 style="color: 32A80F" align="center"> Calculate Age from Date of Birth <br> <br> </h2>
<!-- Choose a date and enter in input field -->
<b> Enter Date of Birth: <input type=date id = DOB> </b>
<span id = "message" style="color:red"> </span> <br><br>
<!-- Choose a date and enter in input field -->
<button type="submit" onclick = "ageCalculator()"> Calculate age
</button> <br><br>
<h3 style="color:32A80F" id="result" align="center"></h3>
</center>
</body>
</html>
```

- ***JavaScript Classes***
- *In JavaScript, classes are the special type of functions.*
- *We can define the class just like function declarations and function expressions.*
- *The JavaScript class contains various class members within a body including methods or constructor.*
- *The class is executed in strict mode.*
- *So, the code containing the silent error or mistake throws an error.*
  
- ***The class syntax contains two components:***
- *Class declarations*
- *Class expressions*

- ***Class Declarations***
- *A class can be defined by using a class declaration.*
- *A class keyword is used to declare a class with any particular name.*
- *According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.*

- *Class Declarations Example*

```
<script>  
//Declaring class  
class Employee  
{  
//Initializing an object  
constructor(id,name)  
{  
this.id=id;  
this.name=name;  
}  
//Declaring method  
detail()  
{  
document.writeln(this.id+" "+this.name+"<br>")  
}  
}
```

**//passing object to a variable**

```
var e1=new Employee(101,"Martin Roy");
var e2=new Employee(102,"Duke William");
e1.detail(); //calling method
e2.detail();
</script>
```

- ***JavaScript Objects***
- *A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.*
- *JavaScript is an object-based language. Everything is an object in JavaScript.*
- ***Creating Objects in JavaScript***
- ***There are 3 ways to create objects.***
- *By object literal*
- *By creating instance of Object directly (using new keyword)*
- *By using an object constructor (using new keyword)*

## **1) JavaScript Object by object literal**

- The syntax of creating object using object literal is given below:
- $\text{object} = \{\text{property1:value1}, \text{property2:value2}, \dots, \text{propertyN:valueN}\}$
- As you can see, property and value is separated by : (colon).
- Let's see the simple example of creating object in JavaScript.

```
<script>
```

```
emp={id:102,name:"Shyam Kumar",salary:40000}
```

```
document.write(emp.id+" "+emp.name+" "+emp.salary);
```

```
</script>
```

## **2) By creating instance of Object**

*The syntax of creating object directly is given below:*

```
var objectname=new Object();
```

*/\*Here, new keyword is used to create object.*

*Let's see the example of creating object directly. \*/*

```
<script>
```

```
var emp=new Object();
```

```
emp.id=101;
```

```
emp.name="Ravi Malik";
```

```
emp.salary=50000;
```

```
document.write(emp.id+" "+emp.name+" "+emp.salary);
```

```
</script>
```

- **3) By using an Object constructor**
- *Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.*
- *The this keyword refers to the current object.*
- *The example of creating object by object constructor is given below.*

```
<script>  
function emp(id,name,salary){  
this.id=id;  
this.name=name;  
this.salary=salary;  
}  
e=new emp(103,"Vimal Jaiswal",30000);  
document.write(e.id+" "+e.name+" "+e.salary);  
</script>
```

- **Defining method in JavaScript Object**
- We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.
- The example of defining method in object is given below.

```
<script>  
function emp(id,name,salary){  
    this.id=id;  
    this.name=name;  
    this.salary=salary;  
    this.changeSalary=changeSalary;  
    function changeSalary(otherSalary){  
        this.salary=otherSalary;  
    }  
}
```

```
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

```
<script>
function Employee(firstName,lastName)
{
    this.firstName=firstName;
    this.lastName=lastName;
}
Employee.prototype.company="Javatpoint"
var employee1=new Employee("Martin","Roy");
var employee2=new Employee("Duke", "William");
document.writeln(employee1.firstName+" "+employee1.lastName+
"+employee1.company+"<br>");
document.writeln(employee2.firstName+" "+employee2.lastName+
"+employee2.company);
</script>
```

- ***JavaScript Encapsulation***
- *The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data.*
- *It allows us to control the data and validate it.*
- ***To achieve an encapsulation in JavaScript: -***
  - *Use var keyword to make data members private.*
  - *Use setter methods to set the data and getter methods to get that data.*

```
<script>  
class Student  
{  
    constructor()  
    {  
        var name;  
        var marks;  
    }  
    getName()  
    {  
        return this.name;  
    }  
    setName(name)  
    {  
        this.name=name;  
    }  
}
```

```
getMarks()  
{  
    return this.marks;  
}  
  
setMarks(marks)  
{  
    this.marks=marks;  
}  
  
}  
  
var stud=new Student();
```

- ***JavaScript Inheritance***
- *The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes.*
- *It provides flexibility to the child class to reuse the methods and variables of a parent class.*
- *The JavaScript extends keyword is used to create a child class on the basis of a parent class.*
- *It facilitates child class to acquire all the properties and behavior of its parent class.*

- *It maintains an IS-A relationship.*
- *The extends keyword is used in class expressions or class declarations.*
- *Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.*
- *We can also use a prototype-based approach to achieve inheritance.*

```
<script>

class Moment extends Date

{
    constructor() {
        super();
    }
}

var m=new Moment();
document.writeln("Current date:")
document.writeln(m.getDate()+"-"+(m.getMonth()+1)+"-"+m.getFullYear());
</script>
```

```
<script>
class Bike {
    constructor()
    {
        this.company="Honda";
    }
}

class Vehicle extends Bike {
    constructor(name,price) {
        super();
        this.name=name;
        this.price=price;
    }
}
var v = new Vehicle("Shine","70000");
document.writeln(v.company+" "+v.name+" "+v.price);
</script>
```

- *Access Modifiers in JavaScript*
- *Access modifiers are keywords used to specify the declared accessibility of a member or a type.*
- *The followings are the access modifiers in most of the object oriented programs.*
- *Private – Access is limited.*
- *Public – Can be access from anywhere, access is not restricted.*
- *Privileged/Protected – Access is limited to the containing class or types derived from the containing class.*

- ***JavaScript Polymorphism***
- *The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms.*
- *It provides an ability to call the same method on different JavaScript objects.*
- *As JavaScript is not a type-safe language, we can pass any type of data members with the methods.*

- *JavaScript Polymorphism Example 1*
- *Let's see an example where a child class object invokes the parent class method.*

```
<script>  
class A  
{  
    display()  
    {  
        document.writeln("A is invoked");  
    }  
}  
  
class B extends A  
{  
}  
  
var b=new B();  
b.display();  
</script>
```

```
<script>
class A
{
    display()
    {
        document.writeln("A is invoked<br>");
    }
}

class B extends A
{
    display()
    {
        document.writeln("B is invoked");
    }
}
```

```
var a=[new A(), new B()]
a.forEach(function(msg)
{
msg.display();
});
</script>
```

***Output:***

*A is invoked*

*B is invoked*

- ***Creating New HTML Elements (Nodes)***
- ***To add a new element to the HTML DOM, you 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>  
const para = document.createElement("p");  
const node = document.createTextNode("This is new.");  
para.appendChild(node);  
const element = document.getElementById("div1");  
element.appendChild(para);  
</script>
```

- ***JavaScript Inheritance***
- *The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes.*
- *It provides flexibility to the child class to reuse the methods and variables of a parent class.*
- *The JavaScript extends keyword is used to create a child class on the basis of a parent class.*
- *It facilitates child class to acquire all the properties and behavior of its parent class.*

- As JavaScript is widely used in Web Development, in this article we would explore some of the Object Oriented mechanism supported by JavaScript to get most out of it.
- Some of the common interview question in JavaScript on OOPS includes,-  
“How Object-Oriented Programming is implemented in JavaScript?
- How they differ from other languages? Can you implement Inheritance in JavaScript and so on...”
- There are certain features or mechanisms which makes a Language Object-Oriented like:
- Object
- Classes
- Encapsulation
- Inheritance

- ***Object–***
- *An Object is a unique entity which contains property and methods. For example “car” is a real life Object, which have some characteristics like color, type, model, horsepower and performs certain action like drive.*
- *The characteristics of an Object are called as Property, in Object Oriented Programming and the actions are called methods.*
- *An Object is an instance of a class. Objects are everywhere in JavaScript almost every element is an Object whether it is a function, arrays and string.*
- *Note: A Method in javascript is a property of an object whose value is a function.*

- *Object can be created in two ways in JavaScript:*

- *Using an Object Literal*

*//Defining object*

```
let person = {  
    first_name:'Mukul',  
    last_name: 'Latiyan',  
    //method  
    getFunction : function(){  
        return ('The name of the person is  
        ${person.first_name} ${person.last_name}')  
    },  
    //object within object  
    phone_number : {  
        mobile:'12345',  
        landline:'6789'  
    }  
}
```

- *console.log(person.getFunction());*
- *console.log(person.phone\_number.landline);*

- *Using an Object Constructor:*

*//using a constructor*

```
function person(first_name,last_name){  
    this.first_name = first_name;  
    this.last_name = last_name;  
}
```

*//creating new instances of person object*

```
let person1 = new person('Mukul','Latiyan');  
let person2 = new person('Rahul','Avasthi');
```

*console.log(person1.first\_name);*

*console.log(`\${person2.first\_name} \${person2.last\_name}`);*

- **Classes-**
- *Classes are blueprint of an Object.*
- *A class can have many Object, because class is a template while Object are instances of the class or the concrete implementation.*
- *Before we move further into implementation, we should know unlike other Object Oriented Language there is no classes in JavaScript we have only Object.*
- *To be more precise, JavaScript is a prototype based object oriented language, which means it doesn't have classes rather it define behaviors using constructor function and then reuse it using the prototype.*

- Lets use ES6 classes then we will look into traditional way of defining Object and simulate them as classes.

// Defining class using es6

```
class Vehicle {  
    constructor(name, maker, engine) {  
        this.name = name;  
        this.maker = maker;  
        this.engine = engine;  
    }  
    getDetails(){  
        return (`The name of the bike is ${this.name}.`)  
    }  
}
```

```
// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name); // Hayabusa
console.log(bike2.maker); // Kawasaki
console.log(bike1.getDetails());
```

- *Encapsulation – The process of wrapping property and function within a single unit is known as encapsulation.*

```
class person{  
    constructor(name,id){  
        this.name = name;  
        this.id = id;  
    }  
    add_Address(add){  
        this.add = add;  
    }  
    getDetails(){  
        console.log(`Name is ${this.name},Address is: ${this.add}`);  
    }  
}
```

```
let person1 = new person('Mukul',21);  
person1.add_Address('Delhi');  
person1.getDetails();
```

- *Encapsulation – The process of wrapping property and function within a single unit is known as encapsulation.*

//encapsulation example

```
class person{  
    constructor(name,id){  
        this.name = name;  
        this.id = id;  
    }  
    add_Address(add){  
        this.add = add;  
    }  
    getDetails(){  
        console.log(`Name is ${this.name},Address is: ${this.add}`);  
    }  
}
```

```
let person1 = new person('Mukul',21);  
person1.add_Address('Delhi');  
person1.getDetails();
```

- *JavaScript Class Syntax*
- *Use the keyword class to create a class.*
- *Always add a method named constructor():*
- *Syntax*

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

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

- *The example above creates a class named "Car".*
- *The class has two initial properties: "name" and "year".*
- *A JavaScript class is not an object.*
- *It is a template for JavaScript objects.*

- *Using a Class*
- *When you have a class, you can use the class to create objects:*
- *Example*
- *let myCar1 = new Car("Ford", 2014);*
- *let myCar2 = new Car("Audi", 2019);*

- ***The Constructor Method***
- ***The constructor method is a special method:***
- *It has to have the exact name "constructor"*
- *It is executed automatically when a new object is created*
- *It is used to initialize object properties*
- *If you do not define a constructor method, JavaScript will add an empty constructor method.*

- *Class Methods*
- *Class methods are created with the same syntax as object methods.*
- *Use the keyword class to create a class.*
- *Always add a constructor() method.*
- *Then add any number of methods.*

*Syntax*

*class ClassName*

*{*

*constructor() { ... }*

*method\_1() { ... }*

*method\_2() { ... }*

*method\_3() { ... }*

*}*

```
class Car {  
    constructor(name, year) {  
        this.name = name;  
        this.year = year;  
    }  
    age() {  
        let date = new Date();  
        return date.getFullYear() - this.year;  
    }  
}  
  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML =  
    "My car is " + myCar.age() + " years old.";
```

```
class Car {  
    constructor(name, year) {  
        this.name = name;  
        this.year = year;  
    }  
    age(x) {  
        return x - this.year;  
    }  
}  
  
let date = new Date();  
let year = date.getFullYear();  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML=  
"My car is " + myCar.age(year) + " years old.";
```

- ***Class Inheritance***
- *To create a class inheritance, use the extends keyword.*
- *A class created with a class inheritance inherits all the methods from another class:*

```
class Car {  
    constructor(brand) {  
        this.carname = brand;  
    }  
    present() {  
        return 'I have a ' + this.carname;  
    }  
}  
  
class Model extends Car {  
    constructor(brand, mod) {  
        super(brand);  
        this.model = mod;  
    }  
    show() {  
        return this.present() + ', it is a ' + this.model;  
    }  
}
```

- *The super() method refers to the parent class.*
- *By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.*

```
class Animal {  
    constructor(legs) {  
        this.legs = legs;  
    }  
    walk() {  
        console.log('walking on ' + this.legs + ' legs');  
    }  
}
```

```
class Bird extends Animal {  
    constructor(legs) {  
        super(legs);  
    }  
    fly() {  
        console.log('flying');  
    }  
}
```

```
let bird = new Bird(2);  
bird.walk();  
bird.fly();
```

## Iterators and generators:

- *Iterators are a new way to loop over any collection in JavaScript.*
- *An iterable is a data structure that wants to make its elements accessible to the public.*
- ***It does so by implementing a method whose key is Symbol.iterator.***
- *It will create iterators.*
- *An iterator is a pointer for traversing the elements of a data structure.*

- ***Iterable values in Javascript***
- *The following values are iterable:*
- *Arrays*
- *Strings*
- *Maps*
- *Sets*

- *The idea of iterability is as follows.*
- **Data consumers:**
- *JavaScript has language constructs that consume data.*
- *For example, for-of loops over values and the spread operator (...) inserts values into Arrays or function calls.*
- **Data sources:**
- *The data consumers could get their values from a variety of sources.*
- *For example, you may want to iterate over the elements of an Array, the key-value entries in a Map or the characters of a string.*

- *An iterator is an object that can access one item at a time from a collection while keeping track of its current position*
- *It just requires that you have a method called next() to move to the next item to be a valid iterator*
- *The result of next() is always an object with two properties –*
- *Value: The value in the iteration sequence*
- *Done: true | false*

**Data consumers**

**Interface**

**Data sources**

for-of loop

spread operator

Iterable

Arrays

Maps

Strings



- ***Symbol.iterator***
- *The Symbol.iterator is a special-purpose symbol made especially for accessing an object's internal iterator.*
- *So, you could use it to retrieve a function that iterates over an array object, like so –*

```
> var numbers = [1,2,4,6,8];
var iterators = numbers[Symbol.iterator]();
iterators.next();
< ▶ {value: 1, done: false}
> iterators.next();
< ▶ {value: 2, done: false}
> iterators.next();
< ▶ {value: 4, done: false}
> iterators.next();
< ▶ {value: 6, done: false}
> iterators.next();
< ▶ {value: 8, done: false}
> iterators.next();
< ▶ {value: undefined, done: true}
```

- **Making objects iterable**
- *So as we learnt in the previous section, we need to implement a method called `Symbol.iterator`.*
- *We will use computed property syntax to set this key.*

```
const array = ['a', 'b', 'c', 'd', 'e'];
```

```
const iterator = array[Symbol.iterator]();
```

```
const first = iterator.next().value
```

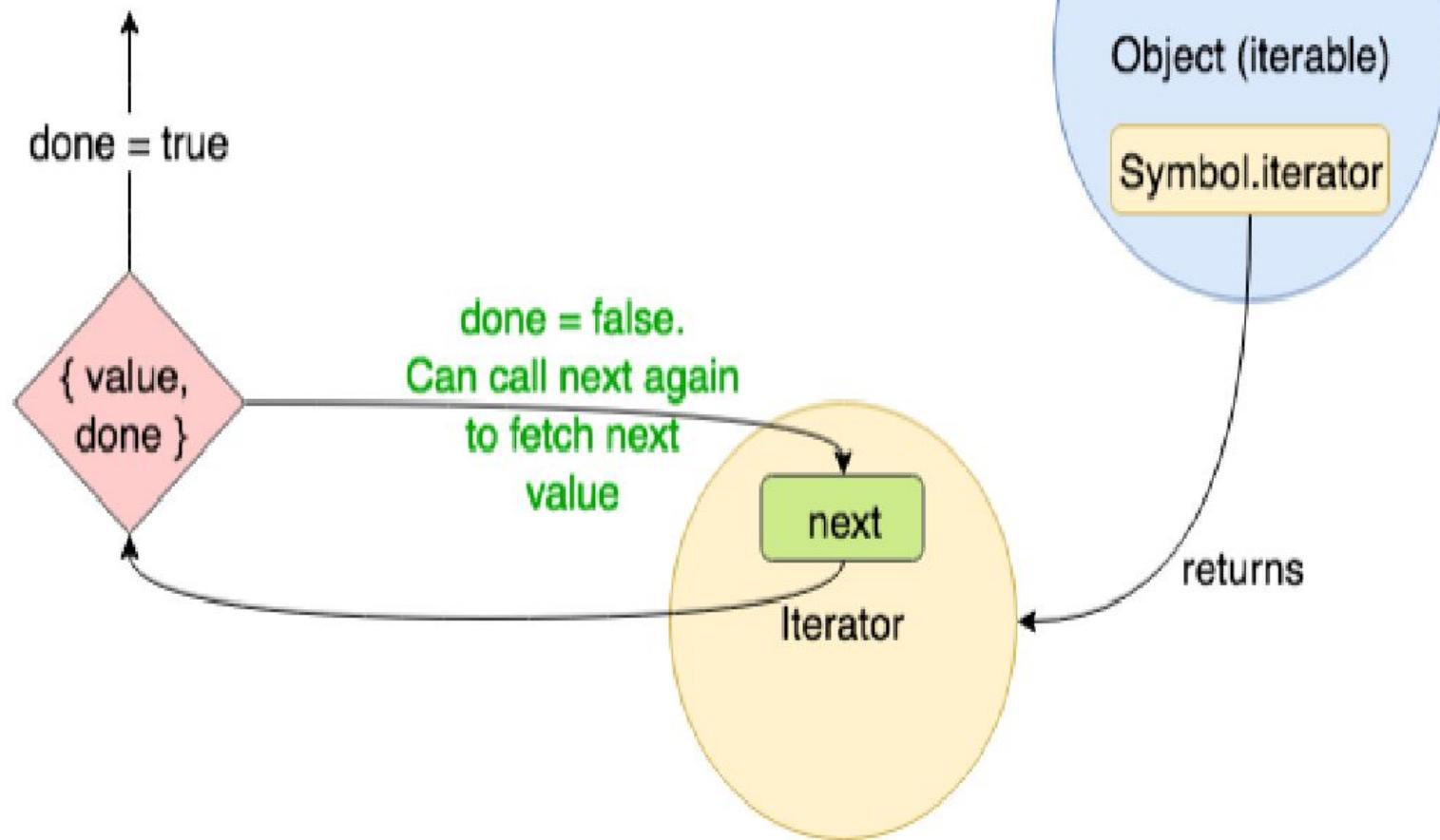
```
iterator.next().value // Since it was skipped, so it's not assigned
```

```
const third = iterator.next().value
```

```
iterator.next().value // Since it was skipped, so it's not assigned
```

```
const last = iterator.next().value
```

The iterator can not give  
more values



- ***What is an Iterator and how does it work?***
- *There are many ways to loop through a data structure in JavaScript.*
- *For example, using a for loop or using a while loop.*
- *An Iterator has similar functionality but with a significant difference.*
- *An iterator only needs to know the current position in the collection as opposed to other loops where they require to load the entire collection upfront in order to loop through it.*
- *Iterators use the next() method to access the next element in the collection.*
- *However, in order to use an Iterator the value or data structure should be iterable.*
- *Arrays, Strings, Maps, Sets are some of the iterables in JavaScript.*
- *A normal object is not iterable.*

- *Why is Iterator better than a normal for loop?*
- *Iterators are better in some cases.*
- *For example, in ordered collections such as Arrays, with no random access, Iterators perform better as they can retrieve elements directly based on the current position.*

- *Iterators improve efficiency by letting you consume a list of items, one at a time, similar to a stream of data.*
- *Generators are a special kind of function capable of pausing the execution.*
- *Invoking a generator allows producing data in chunks (one at a time) without storing it in a list first.*
- *Then, why do we need Iterators?*
- *Remember, using a normal looping algorithm, such as for loop or while loop , you can only loop through collections that allow iterations.*

```
const favouriteMovies = [
  'Harry Potter',
  'Lord of the Rings',
  'Rush Hour',
  'Interstellar',
  'Evolution',
];

// For loop
for (let i=0; i < favouriteMovies.length; i++) {
  console.log(favouriteMovies[i]);
}

// While loop
let i = 0;
while (i < favouriteMovies.length) {
  console.log(favouriteMovies[i]);
  i++;
}
```

- *Since an Array is iterable, using the for loop to traverse through the list possible.*
- *We can implement an iterator for the above as well, which will allow better access to elements based on the current position, without loading the entire collection.*
- *Implementing an iterator to the above would be as follows.*

```
const iterator = favouriteMovies[Symbol.iterator]();

iterator.next(); // { value: 'Harry Potter', done: false }
iterator.next(); // { value: 'Lord of the Rings', done: false }
iterator.next(); // { value: 'Rush Hour', done: false }
iterator.next(); // { value: 'Interstellar', done: false }
iterator.next(); // { value: 'Evolution', done: false }
iterator.next(); // { value: undefined, done: true }
```

- *The next() method will return the Iterator result.*
- *This consists of two values; the element in the collection and the done status.*
- *As you can see, when the traversing is complete, even if we access an element out of the bounds of the array, it won't throw an error.*
- *It would simply return an object with a undefined value and a done status as true .*
- *Iterators improve efficiency by letting you consume a list of items, one at a time, similar to a stream of data.*
- *Generators are a special kind of function capable of pausing the execution.*
- *Invoking a generator allows producing data in chunks (one at a time) without storing it in a list first.*

- ***Introduction to Symbol***
- *ES6 introduces a new primitive type called Symbol.*
- *They are helpful to implement metaprogramming in JavaScript programs.*
- ***Syntax***
- *const mySymbol = Symbol()*
- *const mySymbol = Symbol(stringDescription)*
- *A symbol is just a piece of memory in which you can store some data.*
- *Each symbol will point to a different memory location.*
- *Values returned by a Symbol() constructor are unique and immutable.*

- Let us understand this through an example.
- Initially, we created two symbols without description followed by symbols with same description.
- In both the cases the equality operator will return false when the symbols are compared.

```
<script>
```

```
const s1 = Symbol();
const s2 = Symbol();
console.log(typeof(s1))
console.log(s1 === s2)
const s3 = Symbol("hello");//description
const s4 = Symbol("hello");
console.log(s3)
console.log(s4)
console.log(s3 === s4)
```

```
</script>
```

- *The output of the above code will be as mentioned below –*

*symbol*

*false*

*Symbol(hello)*

*Symbol(hello)*

*false*

- **Symbol.for(key)**
- *searches for existing symbols in a symbol registry with the given key and returns it, if found.*
- *Otherwise, a new symbol gets created in the global symbol registry with this key.*
  
- **Symbol.keyFor(sym)**
- *Retrieves a shared symbol key from the global symbol registry for the given symbol.*

- *This function creates a symbol and adds to registry.*
- *If the symbol is already present in the registry it will return the same; else a new symbol is created in the global symbol registry.*
- *Syntax*
- *Symbol.for(key)*
- *where, key is the identifier of the symbol*

- *The following example shows the difference between `Symbol()` and `Symbol.for()`*

```
<script>
```

```
const userId = Symbol.for('userId') // creates a new Symbol in registry
```

```
const user_Id = Symbol.for('userId') // reuses already created Symbol
```

```
console.log(userId == user_Id)
```

```
const studentId = Symbol("studentID") // creates symbol but not in registry
```

```
const student_Id = Symbol.for("studentID")// creates a new Symbol in registry
```

```
console.log(studentId == student_Id)
```

```
</script>
```

- *The output of the above code will be as shown below –*
- *true*
- *false*

- **Generator-Function :**
- *A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return.*
- *The yield statement suspends function's execution and sends a value back to caller, but retains enough state to enable function to resume where it is left off.*
- *When resumed, the function continues execution immediately after the last yield run.*

- **A Generator is**
- *A function that generates a series of values instead of a single value and can pause and resume when required during the execution.*
- *function\* is the syntax used to write generator functions.*
- *It includes an operator called yield which allows pausing the generator function itself until the next value is requested.*
- *example of a generator would be as follows.*

- ***How does a Generator function differ from a normal function?***
- *As we all know, a normal function cannot pause until its execution is completed.*
- *The only method to break out of a normal function would be to use the return keyword.*
- *The following diagram gives a high-level picture of the difference between a normal function and a generator function.*

- *Syntax :*

```
// An example of generator function
function* gen(){
    yield 1;
    yield 2;
    ...
    ...
}
```

- ***Generator-Object :***
- *Generator functions return a generator object.*
- *Generator objects are used either by calling the next method on the generator object or using the generator object in a “for of” loop (as shown in the above program)*
- *The Generator object is returned by a generating function and it conforms to both the iterable protocol and the iterator protocol.*

```
<script>  
// Generate Function generates three  
// different numbers in three calls  
function *fun()  
{  
    yield 10;  
    yield 20;  
    yield 30;  
}  
  
// Calling the Generate Function  
var gen = fun();  
document.write(gen.next().value);  
document.write("<br>");  
document.write(gen.next().value);  
document.write("<br>");  
document.write(gen.next().value);  
</script>
```

```
<script>// Generate Function generates an
// infinite series of Natural Numbers
function * nextNatural()
{
    var naturalNumber = 1;
    // Infinite Generation
    while (true) {
        yield naturalNumber++;
    }
}
// Calling the Generate Function
var gen = nextNatural();
// Loop to print the first
// 10 Generated number
for (var i = 0; i < 10; i++) {
    // Generating Next Number
    document.write(gen.next().value);
    // New Line
    document.write("<br>");
}
</script>
```

## *Output*

1

2

3

4

5

6

7

8

9

10

```
<script>
```

```
var array = ['a', 'b', 'c'];
function* generator(arr) {
  let i = 0;
  while (i < arr.length) {
    yield arr[i++]
  }
}
const it = generator(array);
// we can do it.return() to finish the generator
</script>
```

- *Encountering yield and yield\* syntax*
- *yield : pauses the generator execution and it returns the value of the expression which is being written after the yield keyword.*
- *yield\* : it iterates over the operand and returns each value until done is true.*

```
<script>  
const arr = ['a', 'b', 'c'];  
function* generator() {  
    yield 1;  
    yield* arr;  
    yield 2;  
}  
for (let value of generator()) {  
    document.write(value);  
    document.write("<br>");  
}  
</script>
```

- *Output*

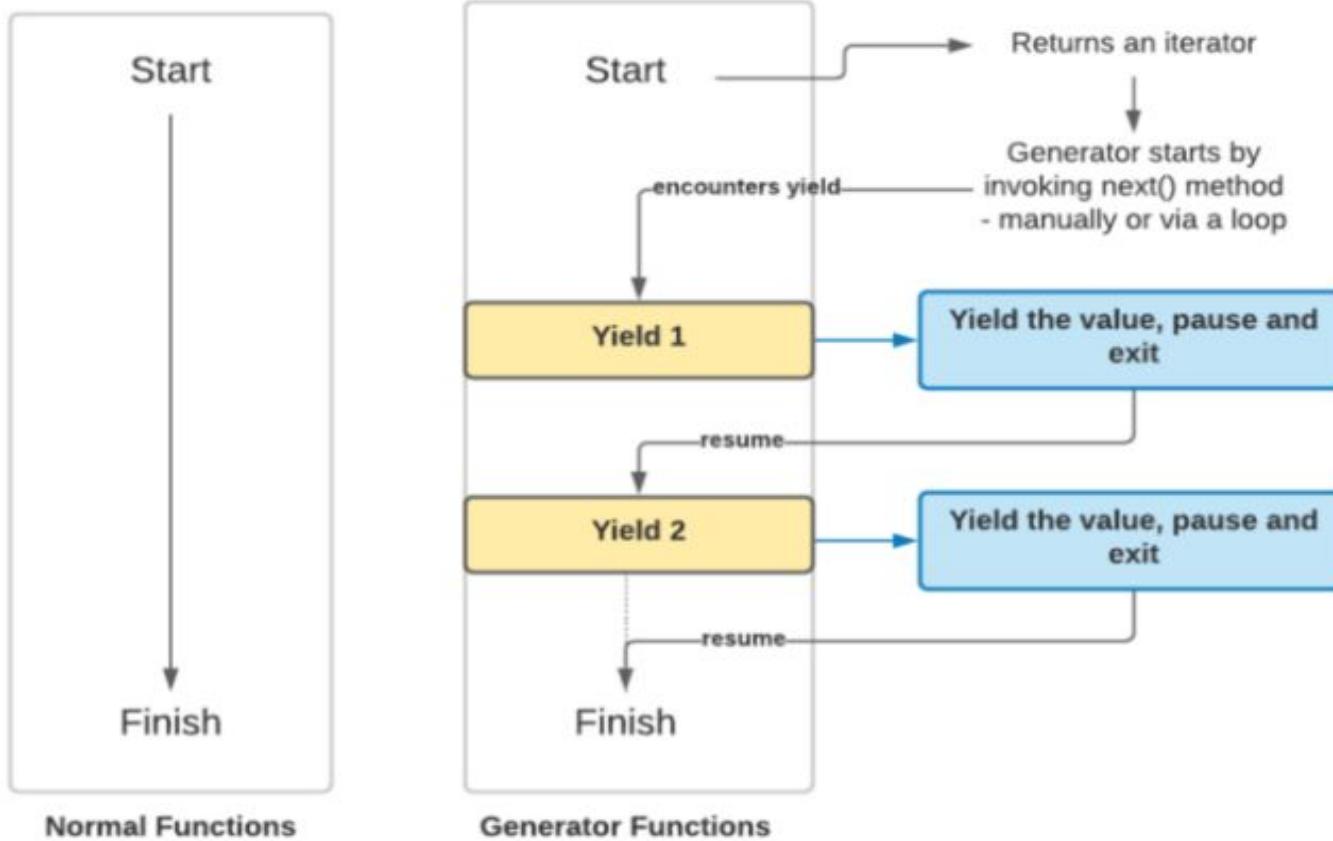
*l*

*a*

*b*

*c*

*2*



# Difference between Iterator and generator

Iterator uses `iter()` and `next()` functions

Generator uses `yield` keyword

Every iterator is not a generator

Every generator is an iterator

- **Generators**
- *Generator functions once called, returns the Generator object, which holds the entire Generator iterable and can be iterated using next() method.*
- *Every next() call on the generator executes every line of code until it encounters the next yield and suspends its execution temporarily.*
- *Generators are a special type of function in JavaScript that can pause and resume state.*
- *A Generator function returns an iterator, which can be used to stop the function in the middle, do something, and then resume it whenever.*

- *This generator object needs to be assigned to a variable to keep track of the subsequent next() methods called on itself.*
- *If the generator is not assigned to a variable then it will always yield only till the first yield expression on every next().*
- *A generator function is a function marked with the \* and has at least one yield-statement in it.*
- *Syntactically they are identified with a \*, either function\* X or function \*X, — both mean the same thing*

- *Generator functions are written using the function\* syntax –*
- *Iterators and generators in JavaScript*
- *'function\*' is a new 'keyword' for generator functions*
- *yield is an operator with which a generator can pause itself*
- *Additionally, generators can also receive input and send output via yield. In short, a generator appears to be a function but it behaves like an iterator.*

- *A generator is a function that produces a sequence of results instead of a single value, i.e you generate a series of values*
- *The value property will contain the value.*
- *The done property is either true or false.*
- *When the done becomes true, the generator stops and won't generate any more values.*
- *The yield is a magical keyword that can do more things other than simply return a value and next() can do more things aside from retrieving the value.*
- *A passing argument to next() - The argument passed to next() will be received by yield –*
- *Iterators and generators in JavaScript*

```
function *gen(x){  
    yield x  
    return  
}  
const generator = gen(2)  
const genVal1 = generator.next()  
console.log("genVal1",genVal1);  
const genVal2 = generator.next()  
console.log("genVal2",genVal2);
```

## Output

```
genVal1 { value: 2, done: false }  
genVal2 { value: undefined, done: true }
```

- *Passing a function to yield*
- *Apart from returning values, the yield can also call a function –*

```
function *gen(){
    yield func()
    return
}

function func(){
    return "Function call"
}

const generator = gen();
const generatorVal1 = generator.next();
console.log("generatorVal1",generatorVal1);
const generatorVal2 = generator.next();
console.log("generatorVal2",generatorVal2);
```

## Output

```
generatorVal1 { value: 'Function call', done: false }
generatorVal2 { value: undefined, done: true }
```

- *Delegating to another generator or iterable using yield\* expression*
- *Iterators and generators in JavaScript*

```
function *gen(){
    yield 3
}
function* func(){
    yield* gen()
}
const iterator1 = func();
const value = iterator1.next().value;
console.log("value",value);
```

- *Output*
- *3*

```
const Iterable = {  
  [Symbol.iterator]0 {  
    let step = 0;  
    const iterator = {  
      next() {  
        step++;  
        if (step === 1)  
          return { value: 'Example', done: 'false' }  
        else if (step === 2)  
          return { value: 'for', done: 'false' }  
        else if (step === 3)  
          return { value: 'Iterator', done: 'false' }  
        return { value: undefined, done: 'true' }  
      }  
    };  
    return iterator;  
  }  
}
```

```
var iterator = Iterable[Symbol.iterator]();
iterator.next() // {value: 'Example', done: 'false'}
iterator.next() // {value: 'for', done: 'false'}
iterator.next() // {value: 'iterator', done: 'false'}
iterator.next() // {value: undefined, done: 'false'}
```

- **Making objects iterable**
- *So as we learnt in the previous section, we need to implement a method called `Symbol.iterator`.*
- *We will use computed property syntax to set this key.*

```
const array = ['a', 'b', 'c', 'd', 'e'];
```

```
const iterator = array[Symbol.iterator]();
```

```
const first = iterator.next().value
```

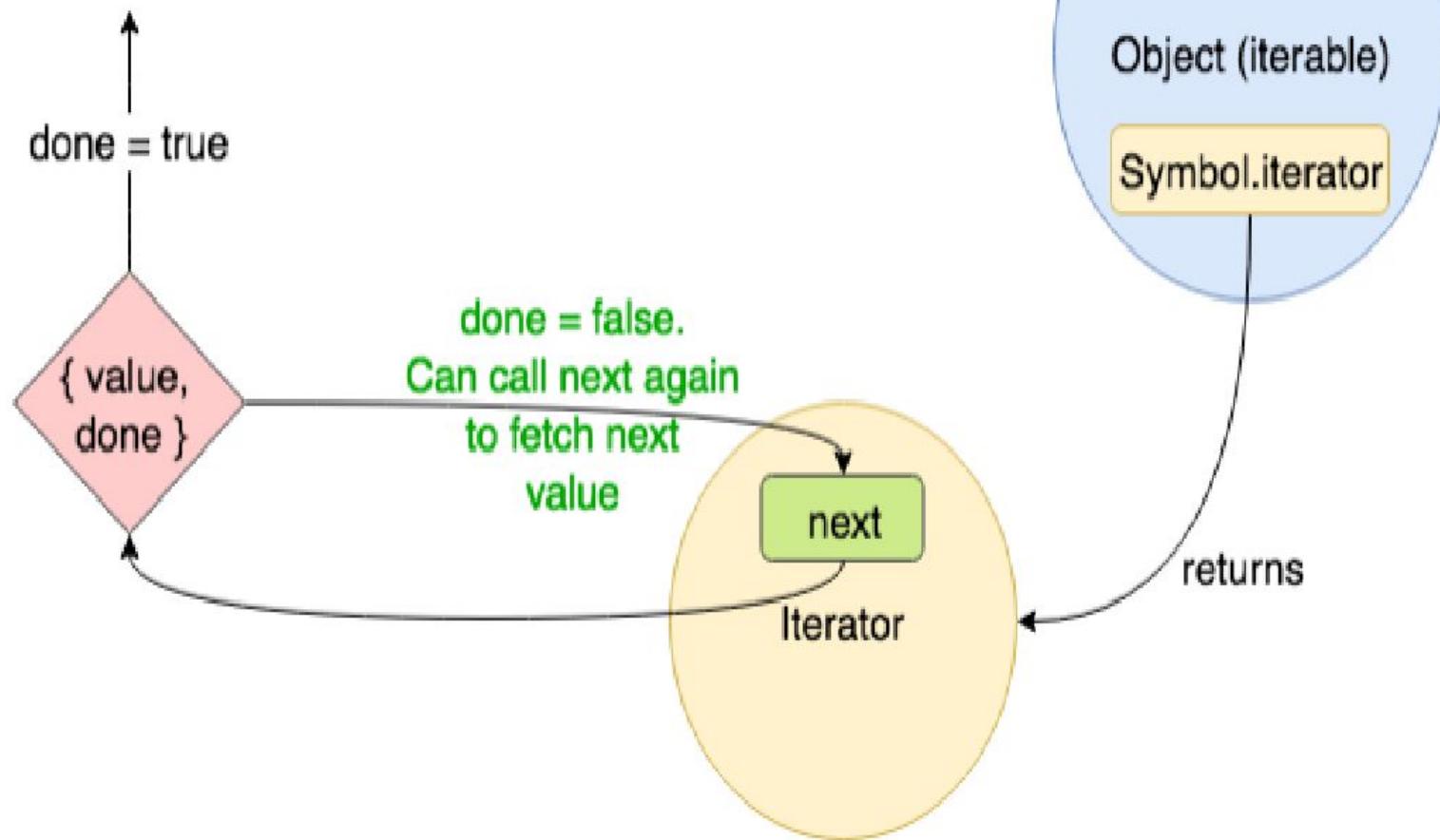
```
iterator.next().value // Since it was skipped, so it's not assigned
```

```
const third = iterator.next().value
```

```
iterator.next().value // Since it was skipped, so it's not assigned
```

```
const last = iterator.next().value
```

The iterator can not give  
more values



- *Iterators improve efficiency by letting you consume a list of items, one at a time, similar to a stream of data.*
- *Generators are a special kind of function capable of pausing the execution. Invoking a generator allows producing data in chunks (one at a time) without storing it in a list first.*

- *What is an Iterator and how does it work?*
- *There are many ways to loop through a data structure in JavaScript.*
- *For example, using a for loop or using a while loop. An Iterator has similar functionality but with a significant difference.*
- *An iterator only needs to know the current position in the collection as opposed to other loops where they require to load the entire collection upfront in order to loop through it.*
- *Iterators use the next() method to access the next element in the collection. However, in order to use an Iterator the value or data structure should be iterable.*
- *Arrays, Strings, Maps, Sets are some of the iterables in JavaScript.*
- *A normal object is not iterable.*

- *Why is Iterator better than a normal for loop?*
- *Iterators are better in some cases.*
- *For example, in ordered collections such as Arrays, with no random access, Iterators perform better as they can retrieve elements directly based on the current position.*
- *However, for unordered collections, since there is no sequence, you cannot experience a major difference in performance.*

- *Then, why do we need Iterators?*
- *Remember, using a normal looping algorithm, such as for loop or while loop , you can only loop through collections that allow iterations. Let's look at an example.*

```
const favouriteMovies = [
  'Harry Potter',
  'Lord of the Rings',
  'Rush Hour',
  'Interstellar',
  'Evolution',
];

// For loop
for (let i=0; i < favouriteMovies.length; i++) {
  console.log(favouriteMovies[i]);
}

// While loop
let i = 0;
while (i < favouriteMovies.length) {
  console.log(favouriteMovies[i]);
  i++;
}
```

- *Since an Array is iterable, using the for loop to traverse through the list possible.*
- *We can implement an iterator for the above as well, which will allow better access to elements based on the current position, without loading the entire collection.*
- *Implementing an iterator to the above would be as follows.*

```
const iterator = favouriteMovies[Symbol.iterator]();

iterator.next(); // { value: 'Harry Potter', done: false }
iterator.next(); // { value: 'Lord of the Rings', done: false }
iterator.next(); // { value: 'Rush Hour', done: false }
iterator.next(); // { value: 'Interstellar', done: false }
iterator.next(); // { value: 'Evolution', done: false }
iterator.next(); // { value: undefined, done: true }
```

- *The next() method will return the Iterator result.*
- *This consists of two values; the element in the collection and the done status.*
- *As you can see, when the traversing is complete, even if we access an element out of the bounds of the array, it won't throw an error.*
- *It would simply return an object with a undefined value and a done status as true .*

- *A Generator is*
- *A function that generates a series of values instead of a single value and can pause and resume when required during the execution.*
- *function\* is the syntax used to write generator functions.*
- *It includes an operator called yield which allows pausing the generator function itself until the next value is requested.*
- *example of a generator would be as follows.*

- *How does a Generator function differ from a normal function?*
- *As we all know, a normal function cannot pause until its execution is completed. The only method to break out of a normal function would be to use the return keyword.*
- *The following diagram gives a high-level picture of the difference between a normal function and a generator function.*