2nd Semester MCA02005 Internet and Web Programming

L-T-P 3-0-0 3 CREDITS

Module I (8 Periods)
Internet Architecture: Internet overview, evolution of internet. Internet components: Local Area Networks, Access Networks, Core Networks, Routers, Transmission infrastructure, ISPs. TCP/IP model, TCP/IP vs OSI model. HTML: HTML Overview, Structure of HTML Documents, Document Types, HTML Elements and attributes. Anchor Attributes, Image Tag
and its attributes, Image and Anchors, Table.

Module II (8 Periods)
Image Map: Attributes, Client Side Image Maps and Server Side Maps.
HTML Layout: Background, colors and text, Tables, Frames, Layers, Page content Division <Div>, <SPAN>. CSS: Style Sheet Basic, Properties, Positioning with Style Sheet.
Forms: <FORM> Elements, Form controls. Dynamic HTML.

Module III (8 Periods)
Java Script: Introduction, Client-Side JavaScript, Server-Side JavaScript, JavaScript Objects,
JavaScript Security. Operators: Assignment Operators, Comparison Operators, Arithmetic Operators, Increment, Decrement, Unary Negation, Logical Operators, String Operators, Special Operators, Conditional operator, Comma operator, delete, new, this, void.
Statements: Break, comment, continue, delete, do ... while, export, for, for...in, function, if...else, import, labelled, return, switch, var, while.

Module IV (8 Periods)
JavaScript (Properties and Methods of Each) :Array, Boolean, Date, Function, Math, Number, Object, String, regExp. Document and its associated objects, document, Link, Area,
Anchor, Image, Applet, Layer.
Events and Event Handlers: General Information about Events, Defining Event Handlers, event.

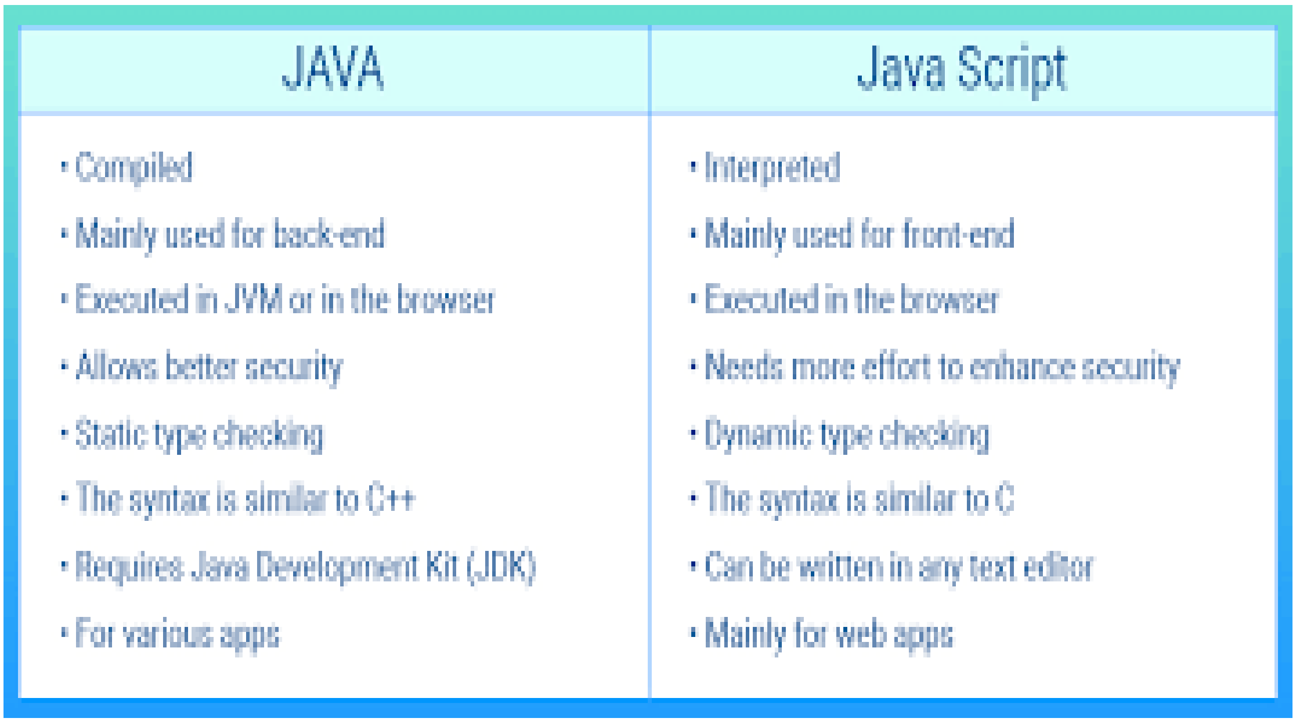
Module V (8 Periods)
Server Side Programming: Common Gateway Interface (CGI), Active Server Pages.
Internet applications: FTP, Telnet, Email, Chat. World Wide Web: HTTP protocol. Search Engines. E-commerce and security issues including symmetric and asymmetric key, encryption and digital signature, and authentication. Emerging trends, Internet telephony, and
virtual reality over the web, etc. Intranet and extranet, firewall.

Books:
1. Computer Networking: A Top-Down Approach Featuring the Internet by Kurose and Ross, Pearson.
2. Web Design the Complete Reference by Thomas Powell, Tata McGrawHill.
3. HTML The Complete Reference by Thomas Powell, Tata McGrawHill.

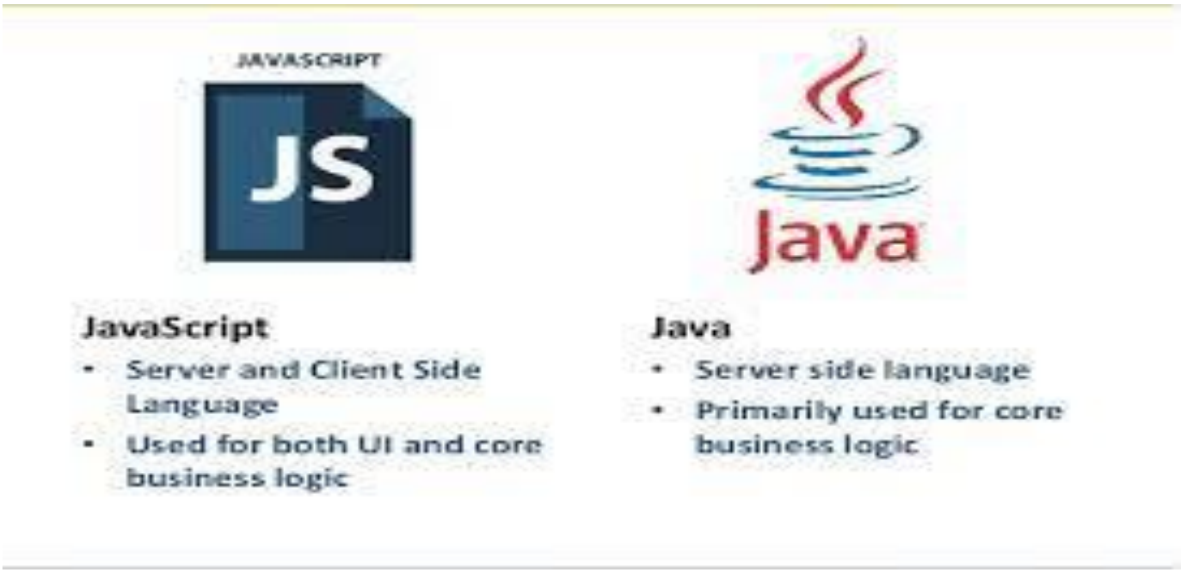
Difference between Scripting and Programming Languages

Parameters	**Scripting Language**	**Programming Language**
Language Type	The scripting languages are interpreter-based languages.	The programming languages are compiler-based languages.
Use	The scripting languages help in combining the existing components of an application.	The programming languages help in developing anything from scratch.
Running of Language	A user needs to run scripting languages inside an existing program. Thus, it’s program-dependent.	Programming languages are program-independent.
Conversion	Scripting languages convert high-level instructions into machine language.	Programming languages help in converting the full program into the machine language (at once).
Compilation	You don’t need to compile these languages.	These languages first need a compilation.
Design	These make the coding process simple and fast.	These provide full usage of the languages.
File Type	Scripting languages don’t create any file types.	Programming languages create .exe files.
Complexity	These are very easy to use and easy to write.	These are pretty complex in terms of writing and usage.
Type of Coding	Scripting languages help write a small piece of an entire code.	Programming languages help write the full code concerning a program.
Developing Time	These take less time because they involve lesser code.	These take more time because a programmer must write the entire code.
Interpretation	We usually interpret a scripting language in another program.	The compile results of a programming language are stand-alone. No other program needs to interpret it.
Requirement of Host	Scripting languages require hosts for execution.	Programming languages are self-executable. They don’t require any host.
Length of Codes	These involve very few and short coding lines.	These require numerous lines of coding for a single function.
Support	These provide limited support to data types, user interface design, and graphic design.	These provide rich support for graphic design, data types, and user interface design.
Maintenance	These involve very low maintenance.	These involve high maintenance.

Cost	It is easier and cheaper to maintain a scripting language.	Maintaining a programming language is comparatively more expensive.
Example	VB Script, Perl, Ruby, PHP, JavaScript, etc.	C, C++, COBOL, Basic, VB, C#, Pascal, Java, etc.

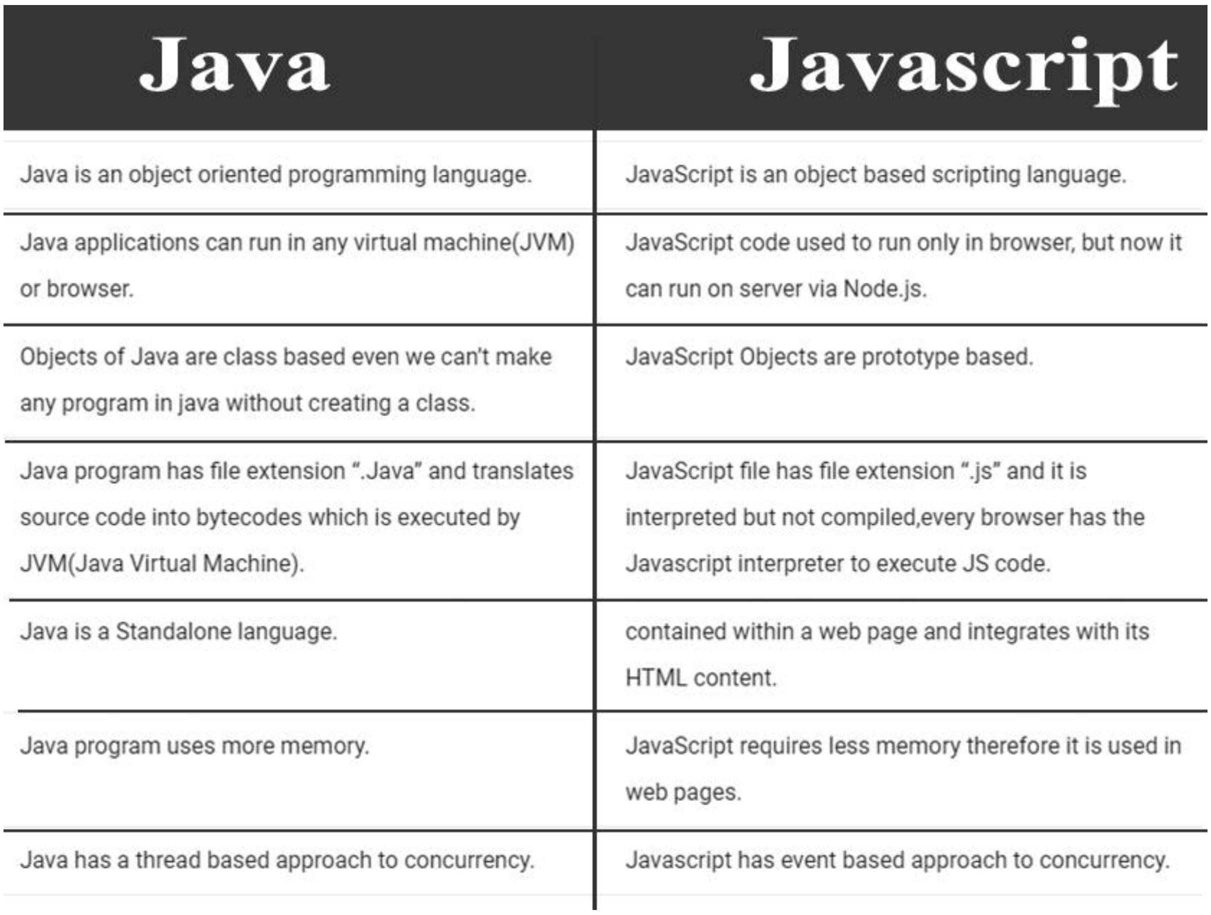


JAVA	Java Script
• Compiled	• Interpreted
• Mainly used for back-end	• Mainly used for front-end
• Executed in JVM or in the browser	• Executed in the browser
• Allows better security	• Needs more effort to enhance security
• Static type checking	• Dynamic type checking
• The syntax is similar to C++	• The syntax is similar to C
• Requires Java Development Kit (JDK)	• Can be written in any text editor
• For various apps	• Mainly for web apps

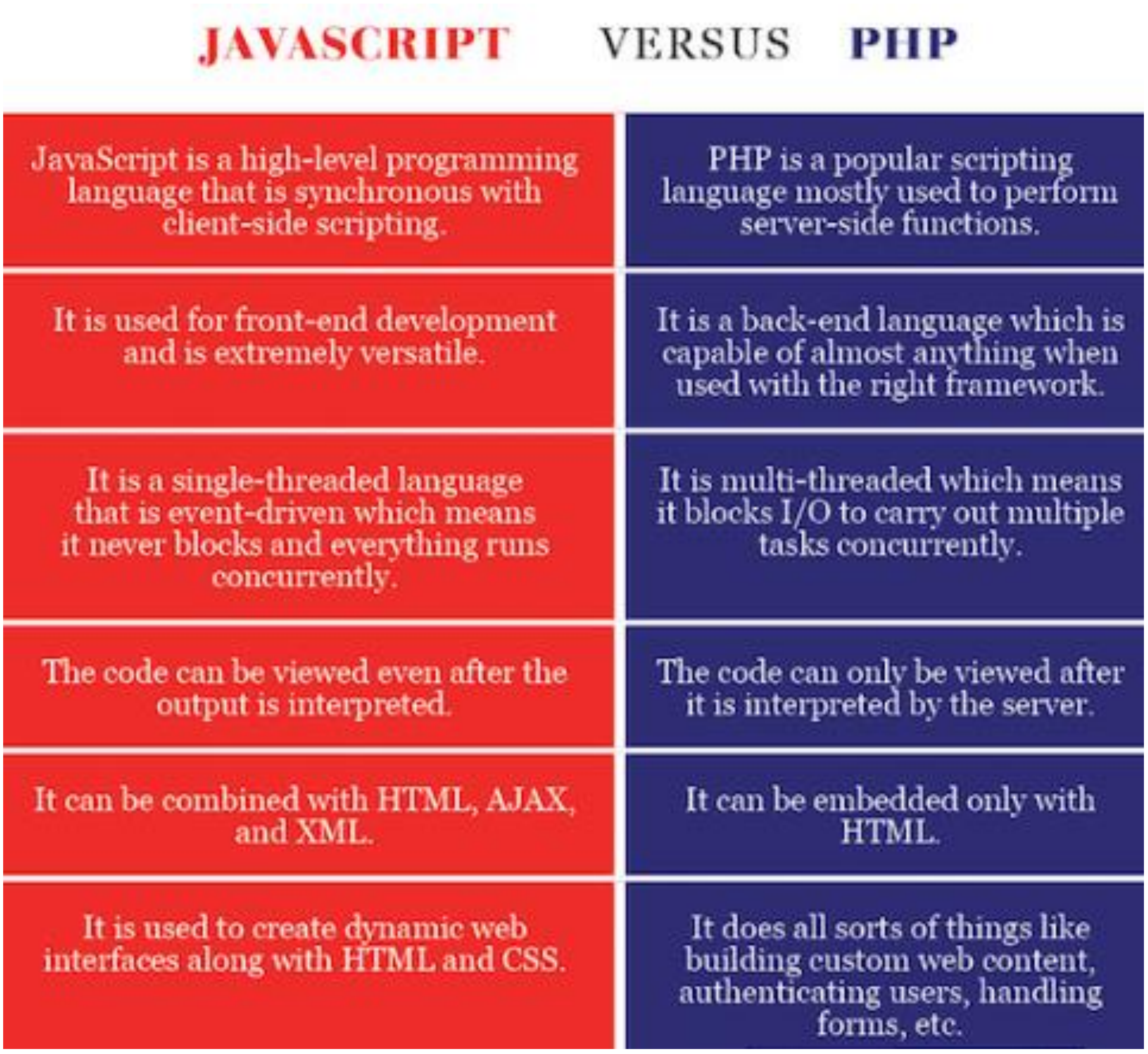

JAVASCRIPT
JS
Java

JavaScript
- Server and Client Side Language
- Used for both UI and core business logic

Java
- Server side language
- Primarily used for core business logic



Java	Javascript
Java is an object oriented programming language.	JavaScript is an object based scripting language.
Java applications can run in any virtual machine(JVM) or browser.	JavaScript code used to run only in browser, but now it can run on server via Node.js.
Objects of Java are class based even we can't make any program in java without creating a class.	JavaScript Objects are prototype based.
Java program has file extension ".Java" and translates source code into bytecodes which is executed by JVM(Java Virtual Machine).	JavaScript file has file extension ".js" and it is interpreted but not compiled,every browser has the Javascript interpreter to execute JS code.
Java is a Standalone language.	contained within a web page and integrates with its HTML content.
Java program uses more memory.	JavaScript requires less memory therefore it is used in web pages.
Java has a thread based approach to concurrency.	Javascript has event based approach to concurrency.



JAVASCRIPT	PHP
JavaScript is a high-level programming language that is synchronous with client-side scripting.	PHP is a popular scripting language mostly used to perform server-side functions.
It is used for front-end development and is extremely versatile.	It is a back-end language which is capable of almost anything when used with the right framework.
It is a single-threaded language that is event-driven which means it never blocks and everything runs concurrently.	It is multi-threaded which means it blocks I/O to carry out multiple tasks concurrently.
The code can be viewed even after the output is interpreted.	The code can only be viewed after it is interpreted by the server.
It can be combined with HTML, AJAX, and XML.	It can be embedded only with HTML.
It is used to create dynamic web interfaces along with HTML and CSS.	It does all sorts of things like building custom web content, authenticating users, handling forms, etc.

What is JavaScript ?
JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here −

- **Microsoft FrontPage** − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX** − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5** − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today ?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

What is client-side JavaScript and server side JavaScript?

These two terms are used in the context of web. **Client-side means that the JavaScript code is run on the client machine, which is the browser. Server-side JavaScript means that the code is run on the server which is serving web pages**.

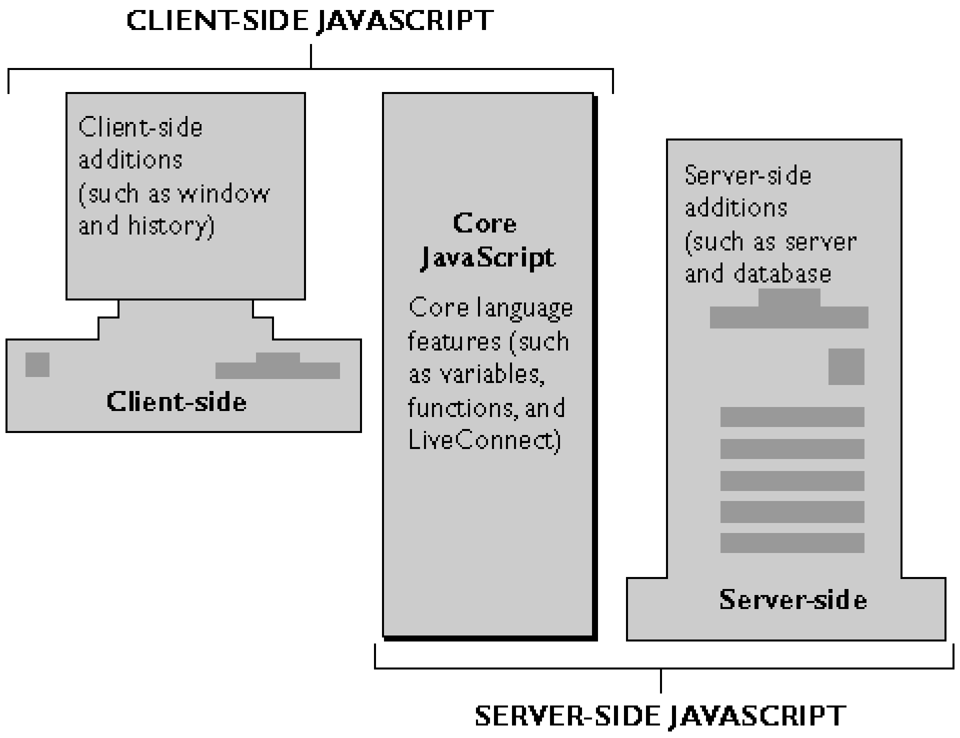
- ***Client-side JavaScript*** extends the core language by supplying objects to control a browser (Navigator or another web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
- ***Server-side JavaScript*** extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

JavaScript lets you create applications that run over the Internet. Client applications run in a browser, such as Netscape Navigator, and server applications run on a server, such as Netscape Enterprise Server. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases.

Through JavaScript's LiveConnect functionality, you can let Java and JavaScript code communicate with each other. From JavaScript, you can instantiate Java objects and access their public methods and fields. From Java, you can access JavaScript objects, properties, and methods.
Netscape invented JavaScript, and JavaScript was first used in Netscape browsers.

Core, Client-Side, and Server-Side JavaScript

The components of JavaScript are illustrated in the following figure.


CLIENT-SIDE JAVASCRIPT
Client-side additions (such as window and history)
Client-side
Core JavaScript
Core language features (such as variables, functions, and LiveConnect)
Server-side additions (such as server and database
Server-side
SERVER-SIDE JAVASCRIPT

The following sections introduce the workings of JavaScript on the client and on the server.

Core JavaScript

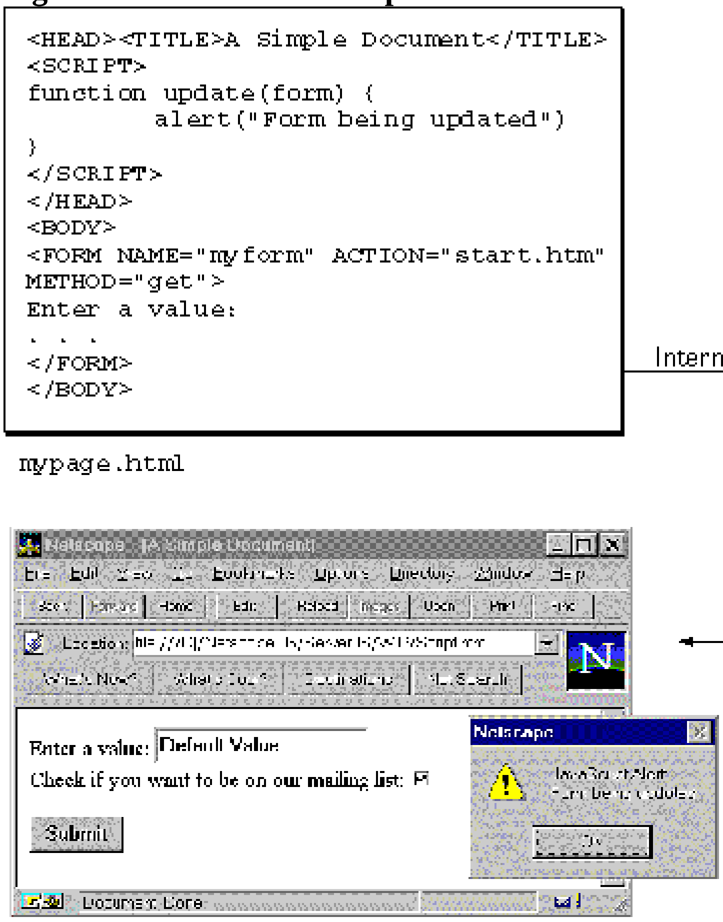
Client-side and server-side JavaScript have the following elements in common:

- Keywords
- Statement syntax and grammar
- Rules for expressions, variables, and literals
- Underlying object model (although client-side and server-side JavaScript have different sets of predefined objects)
- Predefined objects and functions, such as such as Array, Date, and Math

Client-Side JavaScript

Web browsers such as Navigator (2.0 and later versions) can interpret client-side JavaScript statements embedded in an HTML page. When the browser (or *client*) requests such a page, the server sends the full content of the document, including HTML and JavaScript statements, over the network to the client. The browser reads the page from top to bottom, displaying the results of the HTML and executing JavaScript statements as they are encountered. This process, illustrated in the following figure, produces the results that the user sees.

Figure. Client-side JavaScript



```
<HEAD><TITLE>A Simple Document</TITLE>
<SCRIPT>
function update(form) {
        alert("Form being updated")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform" ACTION="start.htm"
METHOD="get">
Enter a value:
. . .
</FORM>
</BODY>
```

Enter a value: Default Value

Check if you want to be on our mailing list:

Submit

JavaScript Alert:
Form being updated

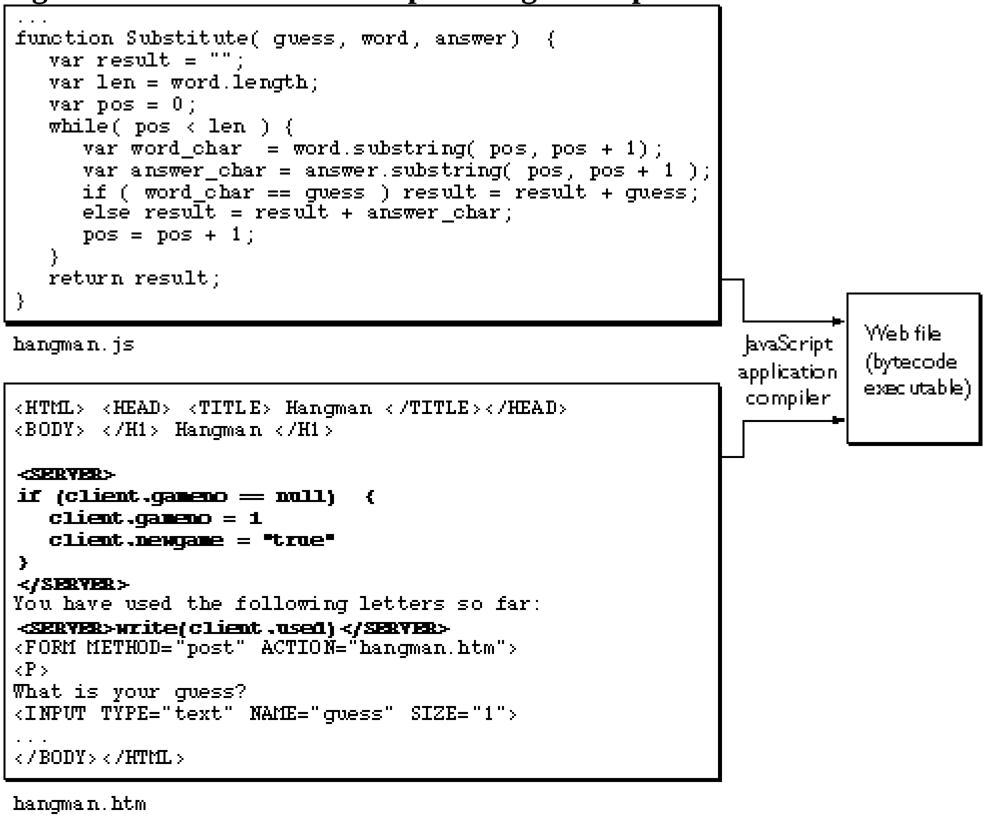
OK

Client-side JavaScript statements embedded in an HTML page can respond to user events such as mouse clicks, form input, and page navigation. For example, you can write a JavaScript function to verify that users enter valid information into a form requesting a telephone number or zip code. Without any network transmission, the embedded JavaScript on the HTML page can check the entered data and display a dialog box if the user enters invalid data.

Different versions of JavaScript work with specific versions of Navigator. For example, JavaScript 1.2 is for Navigator 4.0. Some features available in JavaScript 1.2 are not available in JavaScript 1.1 and hence are not available in Navigator 3.0.

Server-Side JavaScript
On the server, you also embed JavaScript in HTML pages. The server-side statements can connect to relational databases from different vendors, share information across users of an application, access the file system on the server, or communicate with other applications through LiveConnect and Java. HTML pages with server-side JavaScript can also include client-side JavaScript.
In contrast to pure client-side JavaScript pages, HTML pages that use server-side JavaScript are compiled into bytecode executable files. These application executables are run by a web server that contains the JavaScript runtime engine. For this reason, creating JavaScript applications is a two-stage process.

Figure Server-side JavaScript during development



```
...
function Substitute( guess, word, answer)  {
   var result = "";
   var len = word.length;
   var pos = 0;
   while( pos < len ) {
      var word_char  = word.substring( pos, pos + 1);
      var answer_char = answer.substring( pos, pos + 1 );
      if ( word_char == guess ) result = result + guess;
      else result = result + answer_char;
      pos = pos + 1;
   }
   return result;
}
```

hangman.js

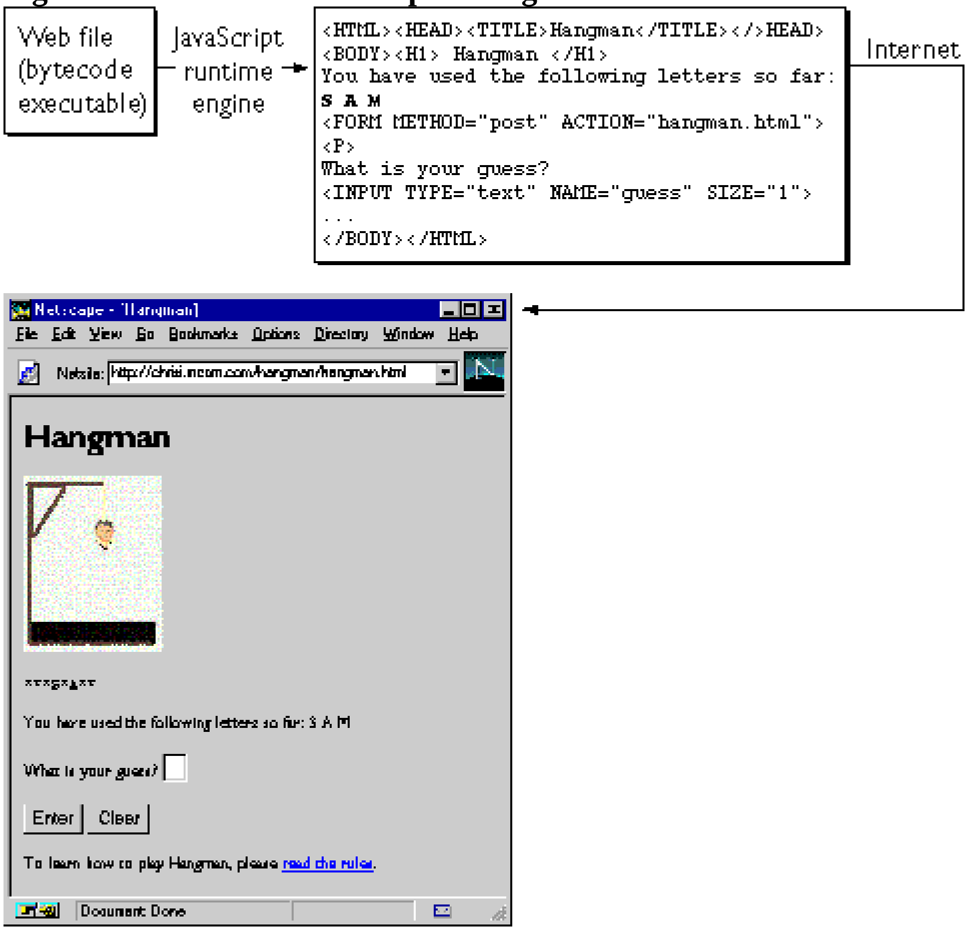
```
<HTML> <HEAD> <TITLE> Hangman </TITLE></HEAD>
<BODY> </H1> Hangman </H1>

<SERVER>
if (client.gameno == null)  {
   client.gameno = 1
   client.newgame = "true"
}
</SERVER>
You have used the following letters so far:
<SERVER>write(client.used)</SERVER>
<FORM METHOD="post" ACTION="hangman.htm">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

hangman.htm

JavaScript application compiler
Web file (bytecode executable)

Figure Server-side JavaScript during runtime


Web file (bytecode executable)
JavaScript runtime engine
Internet
Netscape - [Hangman]
File Edit View Go Bookmarks Options Directory Window Help
Netsite: http://chris.mcom.com/hangman/hangman.html
Enter
Clear
Document: Done

```
<HTML><HEAD><TITLE>Hangman</TITLE></HEAD>
<BODY><H1> Hangman </H1>
You have used the following letters so far:
S A M
<FORM METHOD="post" ACTION="hangman.html">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

Hangman

****g**a***

You have used the following letters so far: S A M

What is your guess?

To learn how to play Hangman, please read the rules.

In contrast to standard Common Gateway Interface (CGI) programs, all JavaScript source is integrated directly into HTML pages, facilitating rapid development and easy maintenance. Server-side JavaScript's Session Management Service contains objects you can use to maintain data that persists across client requests, multiple clients, and multiple applications. Server-side JavaScript's LiveWire Database Service provides objects for database access that serve as an interface to Structured Query Language (SQL) database servers.

JavaScript security

What Is JavaScript Security?

JavaScript security is related to investigating, preventing, protecting, and resolving security issues in applications where JavaScript is used.

JavaScript itself is a fundamental technology for building web applications and is also very popular for building server-side, desktop, and even mobile applications. It's widespread popularity, however, also makes it a prime target for hackers, looking to target it through various attack vectors. Because JavaScript is used mostly in the front-end, it makes sense to focus first on JavaScript security issues in browsers.

Software vendors have also recognized these JavaScript security issues, reacting with JavaScript security scanning software and a variety of JavaScript security testing tools that make applications more secure and greatly reduce

What Are the Common JavaScript Vulnerabilities?

Most common JavaScript attacks vectors include: executing malicious script, stealing a user's established session data or data from the browser's localStorage, tricking users into performing unintended actions, exploiting vulnerabilities in the source code of web applications.

Of course, this list is by no means exhaustive; rather, it is more focused on the front-end aspect of web applications.

JavaScript security Vulnerabilities

i.Source code vulnerabilities

Frequently, source code vulnerabilities may be combined with other—even a number of—JavaScript security holes. Unfortunately in such cases, using a single JavaScript obfuscation cannot prevent or hide these types of vulnerabilities. Because JavaScript is an interpreted, not a compiled, language, it would be virtually impossible to protect application code from being examined by potential hackers with this method. Nonetheless, obfuscation is still a good practice, as it slows down the hackers in their reverse-engineering attempts.

ii.Unintended script execution

The majority of unintended script execution attacks involve **cross-site scripting (XSS)**.

Posting such a script would make every end user a victim unintentionally facilitating the attack by simply running the application, with the malicious code appearing to be part of the web page. While the above code is harmless, a real-life hacker could of course post far more dangerous code.

To prevent XSS attacks, developers should apply sanitization—a combination of escaping, filtering, and validating string data—when handling user input and output from the server.

iii.Escaping/Encoding user input

XSS attacks rely on supplying data that contains certain special characters that are used in the underlying HTML, JavaScript, or CSS of a web page. When the browser is rendering the web page and encounters these characters, it sees them as part of the code of the web page rather than a value to be displayed. This is what allows the attacker to break out of a text field and supply additional browser-side code that gets executed.

iv.Filtering Input

In some cases, it might be preferable to simply remove dangerous characters from the data received as input. This can provide some degree of protection but should not be relied on alone for protection from data manipulation. There are various techniques attackers can use to evade such filters.

v.Input Validation

Whenever possible, browser-supplied input should be validated to ensure it only contains expected characters. For instance, phone number fields should only be allowed to contain numbers and perhaps a dash or parentheses characters. Input containing characters outside the expected set should be immediately rejected. These filters should be set up to look for acceptable characters and reject everything else.
vi.Reliance on client-side validation alone

While all of the methods discussed above are good and work well in browsers, hackers may use special tools to send data directly to the server, thus avoiding client-side validations. This would allow entry of potentially malicious or unverified data to the server. Without additional server-side validation, stored data could be corrupted or replaced with erroneous data.

vii.Stealing session data

Client-side browser script can be very powerful in that it has access to all the content returned by a web application to the browser. This includes cookies that could potentially contain sensitive data, including user session IDs. In fact, a common exploit of XSS attacks is to send the user's session ID tokens to the attacker so they can hijack the session.
viii.Inducing users to perform unintended actions

Cross-site request forgery (CSRF) attacks attempt to trick a browser into executing malicious requests on the websites the user is already logged in to, even if the site is not actually opened at that time. If sessions on the target site are cookie-based, requests to that site can be automatically enriched with authorization cookies.
How to Deal with JavaScript Security Issues?
Protecting applications and servers from JavaScript vulnerabilities can be managed through the adoption of JavaScript security best practices and the use of sophisticated **scanning tools**.
In the world of web development, software engineers must constantly keep on top of new JavaScript security risks that arise. Not only is it important to conduct functionality tests on applications; using JavaScript security testing tools on a regular basis is also key for preventing vulnerabilities. Last, following some simple and common best practices will definitely increase the durability of your applications.
The following JavaScript security best practices can reduce this risk.

- **Avoid eval():**
  Don't utilize this command in code, since it simply executes passed argument if it is a JavaScript expression. This means if the hacker succeeds in manipulating input value, he or she will be able to run any script she wants. Instead, opt for alternative options that are more secure.
- **Encrypt:**
  Use HTTPS/SSL to encrypt data exchanged between the client and the server.
- **Set secure cookies:**
  To ensure SSL/HTTPS is in use, set your cookies as "secure," which limits the use of your application's cookies to only secure web pages.
- **Set API access keys:**
  Assign individual tokens for each end user. If these tokens don't match up, access can denied or revoked.
- **Use safe methods of DOM manipulation:**
  Methods such as innerHTML are powerful and potentially dangerous, as they don't limit or escape/encode the values that are passed to them. Using a method like

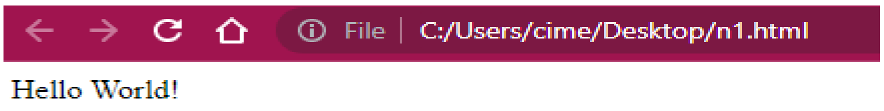
innerText instead provides inherent escaping of potentially hazardous content. This is particularly useful in preventing DOM-based XSS attacks.

Embedded Java Script

n1.html

```
<html>
   <body>
      <script>
       document.write("Hello World!")
       </script>
   </body>
</html>
```

O/P

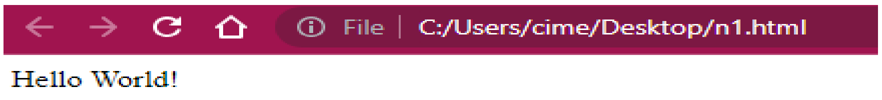


```
C:/Users/cime/Desktop/n1.html
Hello World!
```

Embedded Java Script

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

O/P



```
C:/Users/cime/Desktop/n1.html
Hello World!
```

External Javascript

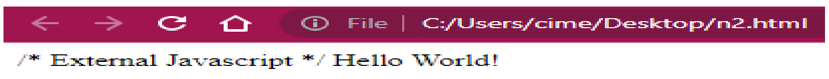
n2.js

```
document.write ("Hello World!")
```

n2.html

```
/* External Javascript */
<html>
<body>
<script src="n2.js">
</script>
</body>
</html>
```

O/P



```
C:/Users/cime/Desktop/n2.html
/* External Javascript */ Hello World!
```

JavaScript - HTML DOM Methods
The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

HTML DOM methods are actions you can perform (on HTML Elements).
HTML DOM properties are values (of HTML Elements) that you can set or change.

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

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

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

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

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

- The open() method opens a document for writing.
- The write() method writes directly to an open (HTML) document stream.
- The writeln() method is identical to the write() method, with the addition of writing a newline character (U+000A) after each statement.

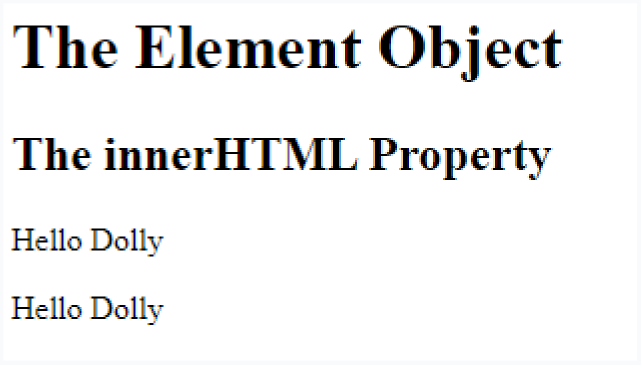
What is the use of document getElementById () innerHTML?

When you use innerHTML , you can change the page's content without refreshing the page. This can make your website feel quicker and more responsive to user input. The innerHTML property can be used along with getElementById() within your JavaScript code to refer to an HTML element and change its contents.
The innerHTML property sets or returns the HTML content (inner HTML) of an element.

```
<!DOCTYPE html>
<html>
<body>
<h1>The Element Object</h1>
<h2>The innerHTML Property</h2>
<p id="myP">This is a p element.</p>
<div id="myDIV">This is a div element.</div>
<script>
text = "Hello Dolly";
document.getElementById("myP").innerHTML = text;
document.getElementById("myDIV").innerHTML = text;
</script>
</body>
```

```
</html>
```

O/P



The Element Object

The innerHTML Property

Hello Dolly

Hello Dolly

JavaScript Variables
Variables are containers for storing data (storing data values).

There are four ways to declare a javascript variable:

- Using var
- Using let
- Using const
- Using nothing

Using var

```
<!DOCTYPE html>
<html>
<body>
<script>
var  x = 10;
{
var x = 80;
}
document.write(x);
</script>
</body>
</html>
```

Output
80

Using let

```
<!DOCTYPE html>
<html>
<body>
<script>
let  x = 50;
 {
  let x = 20;
  }
  document.write(x);
</script>
</body>
</html>
```

Output

```
50
```

Using nothing

```
<!DOCTYPE html>
<html>
<body>
<script>
x = 50;
y = 60;
z = x + y;
document.write(z);
</script>
</body>
</html>
```

Output

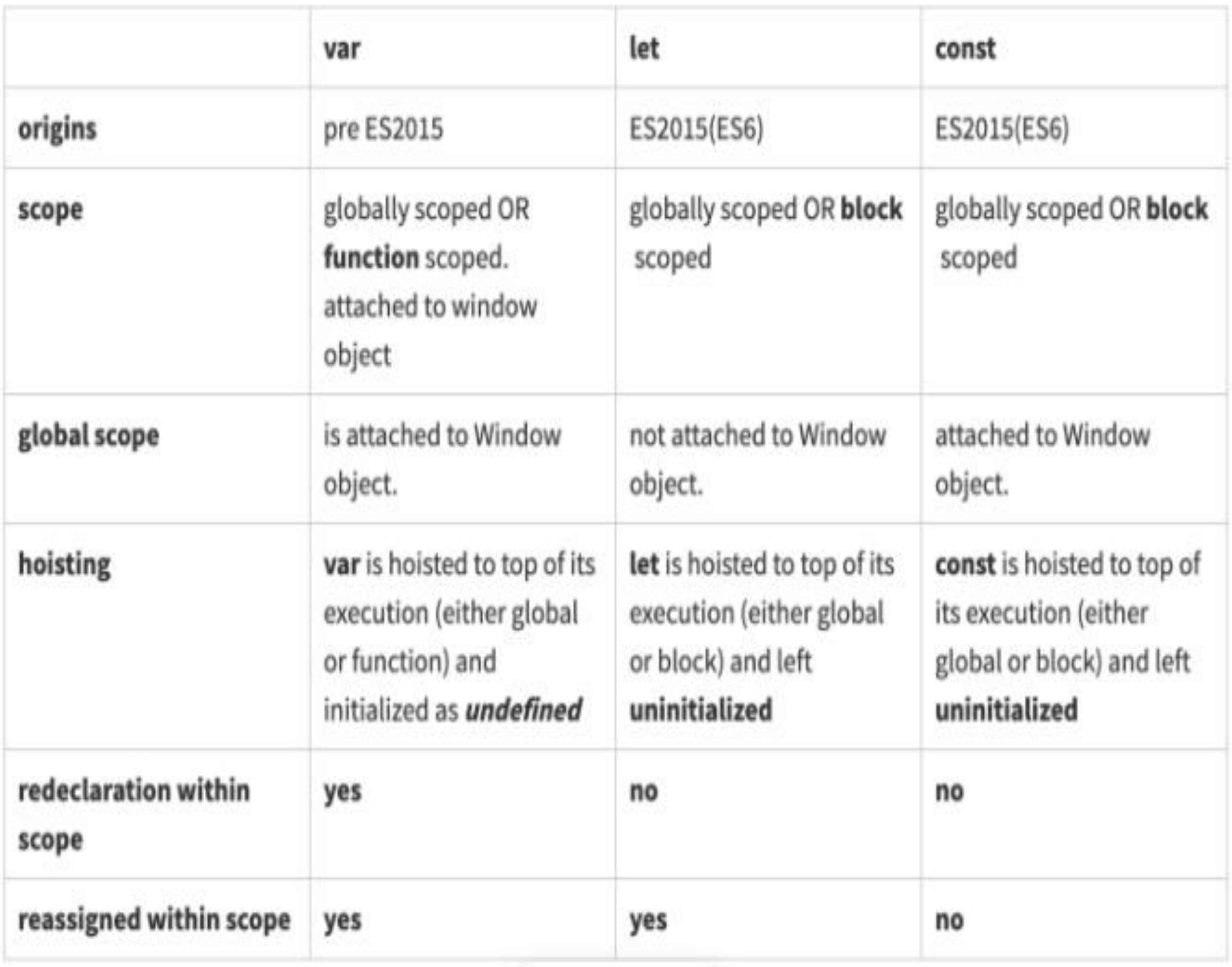
```
110
```

Using const

```
<!DOCTYPE html>
<html>
<body>
<script>
const  x = 110;
{
const x = 2;
}
document.write(x);
</script>
</body>
</html>
```

Output

```
110
```



	var	let	const
origins	pre ES2015	ES2015(ES6)	ES2015(ES6)
scope	globally scoped OR **function** scoped. attached to window object	globally scoped OR **block** scoped	globally scoped OR **block** scoped
global scope	is attached to Window object.	not attached to Window object.	attached to Window object.
hoisting	**var** is hoisted to top of its execution (either global or function) and initialized as ***undefined***	**let** is hoisted to top of its execution (either global or block) and left **uninitialized**	**const** is hoisted to top of its execution (either global or block) and left **uninitialized**
redeclaration within scope	**yes**	**no**	**no**
reassigned within scope	**yes**	**yes**	**no**

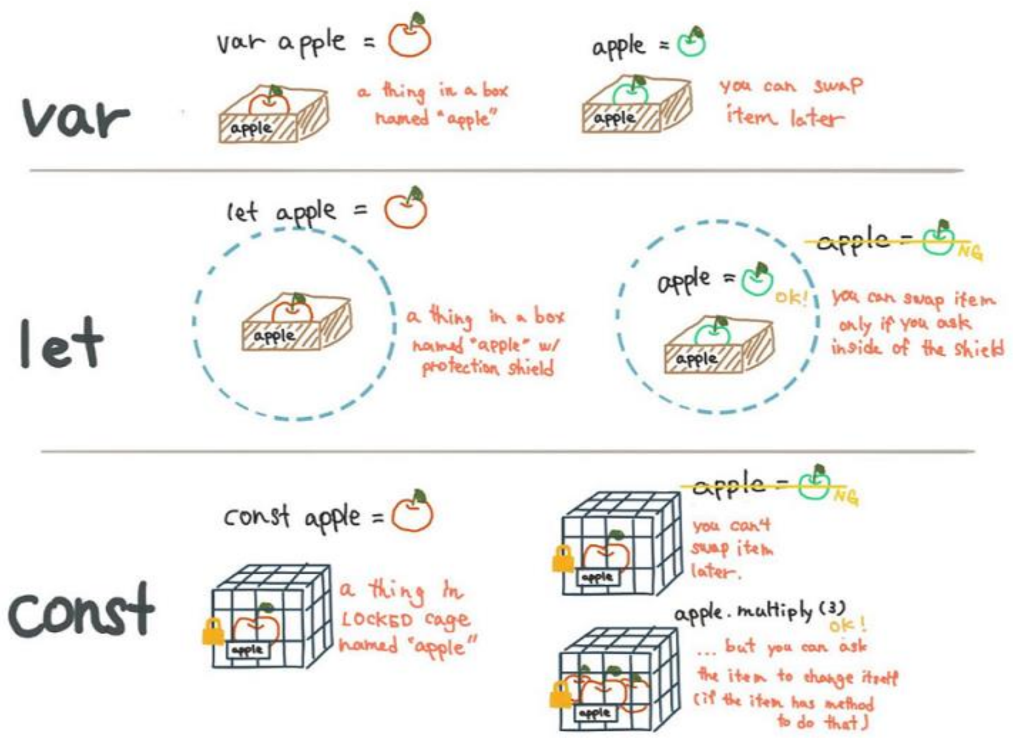
```
<!DOCTYPE html>
<html>
<body>
<script>
let x = 20;   // Allowed
{
  let x = 3;   // Allowed
}

{

  let x = 4;   // Allowed
}
document.write(x);
</script>
</body>
</html>
```

O/P

```
20
```


var
var apple =
apple
a thing in a box named "apple"
apple =
you can swap item later
let
let apple =
a thing in a box named "apple" w/ protection shield
apple =
ok!
~~apple =~~ NG
you can swap item only if you ask inside of the shield
const
const apple =
a thing in LOCKED cage named "apple"
~~apple =~~ NG
you can't swap item later.
apple.multiply(3)
ok!
... but you can ask the item to change itself (if the item has method to do that)

JavaScript Data Types

There are eight basic data types in JavaScript. They are:

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

Object	key-value pairs of collection of data	let student = { };

Here, all data types except Object are primitive data types, whereas Object is non-primitive.
Note: The Object data type (non-primitive type) can store collections of data, whereas primitive data type can only store a single data.

JavaScript Operators

Example

Assign values to variables and add them together:

```
 x = 5;         // assign the value 5 to x
 y = 2;         // assign the value 2 to y
 z = x + y;     // assign the value 7 to z (5 + 2)
```

The assignment operator (=) assigns a value to a variable.

Assignment

```
 x = 10;
```

The addition operator (+) adds numbers:

Adding

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

The multiplication operator (*) multiplies numbers.

Multiplying

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

Types of JavaScript Operators

There are different types of JavaScript operators:

- Aritmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

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

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

JavaScript Assignment Operators
Assignment operators assign values to JavaScript variables.

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

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

Assignment
```
 x = 10;
x += 5;
```

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

Example
```
 text1 = "John";
 text2 = "Doe";
 text3 = text1 + " " + text2;
```
The result of text3 will be:

John Doe
The += assignment operator can also be used to add (concatenate) strings:

Example
```
text1 = "What a very ";
text1 += "nice day";
```
The result of text1 will be:

What a very nice day
When used on strings, the + operator is called the concatenation operator.

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

Example
```
 x = 5 + 5;
 y = "5" + 5;
 z = "Hello" + 5;
```
The result of x, y, and z will be:

```
10
55
Hello5
```
If you add a number and a string, the result will be a string!

JavaScript Comparison Operators

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

Comparison operators are fully described in the JS Comparisons chapter.

JavaScript Logical Operators

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

Logical operators are fully described in the JS Comparisons chapter.

JavaScript Type Operators

Operator	**Description**
typeof	Returns the type of a variable
instanceof	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	
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

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

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

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

Arithmetic Operations
A typical arithmetic operation operates on two numbers.
The two numbers can be literals:

Example
```
 x = 100 + 50;
```

or variables:

Example
```
 x = a + b;
```

or expressions:

Example
```
 x = (100 + 50) * a;
```

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

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

Adding

The **addition** operator (+) adds numbers:

Example

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

Subtracting

The **subtraction** operator (-) subtracts numbers.

Example

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

Multiplying

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

Example

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

Dividing

The **division** operator (/) divides numbers.

Example

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

Remainder

The **modulus** operator (%) returns the division remainder.

Example

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

In arithmetic, the division of two integers produces a **quotient** and a **remainder**.
In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.

Incrementing

The **increment** operator (++) increments numbers.

Example

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

Decrementing

The **decrement** operator (--) decrements numbers.

Example

```
 x = 5;
x--;
 z = x;
```

Exponentiation

The **exponentiation** operator (**) raises the first operand to the power of the second operand.

Example

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

x ** y produces the same result as Math.pow(x,y):

Example

```
 x = 5;
 z = Math.pow(x,2);
```

Operator Precedence

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

Example

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

Is the result of example above the same as 150 * 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

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

Example

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

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

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

Example

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

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	**Example**	**Same As**
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y

/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

The **= operator is a part
It does not work in Internet Explorer 11 or earlier.

Shift Assignment Operators

Operator	**Example**	**Same As**
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Logical Assignment Operators

Operator	**Example**	**Same As**
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

The = Operator
The = assignment operator assigns a value to a variable.

Simple Assignment

```
 x = 10;
```

The += Operator
The += assignment operator adds a value to a variable.

Addition Assignment

```
 x = 10;
x += 5;
```

The -= Operator
The -= assignment operator subtracts a value from a variable.

Subtraction Assignment

```
 x = 10;
x -= 5;
```

The *= Operator
The *= assignment operator multiplies a variable.

Multiplication Assignment

```
 x = 10;
x *= 5;
```

The /= Operator
The /= assignment divides a variable.
Division Assignment
```
 x = 10;
x /= 5;
```

The %= Operator
The %= assignment operator assigns a remainder to a variable.
Remainder Assignment
```
 x = 10;
x %= 5;
```

The <<= Operator
The <<= assignment operator left shifts a variable.
Left Shift Assignment
```
 x = -100;
x <<= 5;
```

The >>= Operator
The >>= assignment operator right shifts a variable (signed).
Right Shift Assignment
```
 x = -100;
x >>= 5;
```

The >>>= Operator
The >>>= assignment operator right shifts a variable (unsigned).
Unsigned Right Shift Assignment
```
 x = -100;
x >>>= 5;
```

The &= Operator
The &= assignment operator ANDs a variable.
Bitwise AND Assignment
```
 x = 10;
x &= 5;
```

The != Operator
The != assignment operator ORs a variable.
Bitwise OR Assignment
```
 x = 10;
x != 5;
```

Test JavaScript Operator Precedence
Operator precedence describes the order in which operations are performed in an arithmetic expression.

Multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

Example

As in traditional mathematics, multiplication is done first:

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

Example

As in traditional mathematics, the precedence can be changed by parentheses:

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

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

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

Example

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

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, the table below explains the comparison operators:

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

>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Logical Operators

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

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

The **console.log()** is a function in JavaScript which is used to print any kind of variables defined before in it or to just print any message that needs to be displayed to the user.

Syntax:

console.log(A);
Parameters: It accepts a parameter which can be an array, an object or any message.
Return value: It returns the value of the parameter given.

Syntax of Increment & Decrement Operator
Increment Operator ++x or x++. This is equal to x=x+1

Decrement Operator --x or x--. This is equal to x=x-1

Example

```
let x=10
++x;    //increment the value by one  x=x+1
console.log(x);   //11

let x=10
x--;    //decrement the value by one  x=x-1
console.log(x);   //9
```

Prefix & Postfix
There are two ways you can use the operator. One is before the operand, which is known as Prefix. The other method is to use it after the operand, which is known as Postfix.

Prefix Example

```
let a=10;
a++;
console.log(a)   // 11
```

Postfix Example

```
let a=10;
++a;
console.log(a)   // 11
```

Difference Between Prefix & Postfix
When we use the ++ operator as a prefix as in ++a

The value of the variable a is incremented by 1
Then it returns the value.

```
a=10;
b=++a;      //a is incremented, a is then assigned to b
console.log(b);  //11
console.log(a);  //11
```

When we use the ++ operator as a Postfix as in a++,

The value of the variable is returned.
Then the variable a is incremented by 1

```
a=10;
b=a++;  //a is assigned to b, then a is incremented
console.log(b);  //10
console.log(a);  //11
```

Unary negation(-) Operator

Unary negation(-) operation is a single operand operator (which means it worked with only a single operand preceding or succeeding to it), which is used to convert its operand to a negative number.

Syntax:
-Operand

Example 1:
```
<script>
  const a = 20;
  const b = -a;
  console.log(b);
  console.log(typeof b);
  const x = '20';
  const y = -x;
  console.log(y);
  console.log(typeof y);
</script>
Output:
-20
number
-20
number
```

JavaScript Comma Operator

JavaScript uses a comma (,) to represent the comma operator. A comma operator takes two expressions, evaluates them from left to right, and returns the value of the right expression.

Here's the syntax of the comma operator:

```
leftExpression, rightExpression
```

For example:

```
let result = (10, 10 + 20);
console.log(result);
```

Output:

```
30
```

In this example, the 10, 10+20 returns the value of the right expression, which is 10+20. Therefore, the result value is 30.

```
let x = 10;
let y = (x++, x + 1);
console.log(x, y);
Output:
11 12
```

In this example, we increase the value of x by one (x++), add one to x (x+1) and assign x to y. Therefore, x is 11, and y is 12 after the statement.

However, to make the code more explicit, you can use two statements rather than one statement with a comma operator like this:

```
let x = 10;
x++;
let y = x + 1;
console.log(x, y);
```

String Operators
String Concatenation
When working with JavaScript strings sometimes you need to join two or more strings together into a single string. Joining multiple strings together is known as concatenation.

The concatenation operator

The concatenation operator (+) concatenates two or more string values together and return another string which is the union of the two operand strings.

The shorthand assignment operator += can also be used to concatenate strings.

Example:

The following web document demonstrates the use of concatenation operator (+) and shorthand assignment operator +=.

'google' + '.' + ' com=google. com

Special Operators in JavaScript

Conditional operator

The *conditional operator* (also known as the *ternary operator*) uses three operands. It evaluates a logical expression and then returns a value based on whether that expression is true or false. The conditional operator is the only operator that requires three operands. For example:

```
var isItBiggerThanTen = (value > 10) ? "greater than 10" : "not greater than 10";
```

Comma operator

The *comma operator* evaluates two operands and returns the value of the second one. It's most often used to perform multiple assignments or other operations within loops. It can also serve as a shorthand for initializing variables. For example:

```
var a = 10 , b = 0;
```

Because the comma has the lowest precedence of the operators, its operands are always evaluated separately.

delete operator

The delete operator removes a property from an object or an element from an array.
When you use the delete operator to remove an element from an array, the length of the array stays the same. The removed element will have a value of undefined.

```
var animals = ["dog","cat","bird","octopus"];
console.log (animals[3]); // returns "octopus"
delete animals[3];
console.log (animals[3]); // returns "undefined"
```

in operator
The in operator returns true if the specified value exists in an array or object.

```
var animals = ["dog","cat","bird","octopus"];
if (3 in animals) {
 console.log ("it’s in there");
}
```

In this example, if the animals array has an element with the index of 3, the string “it’s in there” will print out to the JavaScript console.

instanceof operator
The instanceof operator returns true if the object you specify is the type of object that has been specified.

```
var myString = new String();
if (myString instanceof String) {
 console.log("yup, it’s a string!");
}
```

new operator
The new operator creates an instance of an object. JavaScript has several built-in object types, and you can also define your own. In the following example, Date() is a built-in JavaScript object, while Pet() and Flower() are examples of objects that a programmer could create to serve custom purposes within a program.

```
var today = new Date();
var bird = new Pet();
var daisy = new Flower();
```

this operator
The this operator refers to the current object. It’s frequently used for retrieving properties within an object.

typeof operator
The typeof operator returns a string containing the type of the operand:

```
var businessName = "Harry’s Watch Repair";
console.log typeof businessName; // returns "string"
```

void operator

The void operator causes an expression in the operand to be evaluated without returning a value. The place where you most often see void used is in HTML documents when a link is needed, but the creator of the link wants to override the default behavior of the link using JavaScript:

```
<a href="javascript:void(0);">This is a link, but it won't do anything</a>
```

JavaScript Operator Precedence Values

	Operator	**Description**	**Example**
18	( )	Expression grouping	(3 + 4)
17	.	Member	person.name
17	[]	Member	person["name"]
17	()	Function call	myFunction()
17	new	(with arguments)	new Person("John", "Doe")
16	new	(without arguments)	new Date()
15	++	Postfix Increment	i++
15	--	Postfix Decrement	i--
14	++	Prefix Increment	++i
14	--	Prefix Decrement	--i
14	!	Logical NOT	!(x==y)
14	~	Bitwise NOT	~x
14	typeof	Data Type	typeof x
13	**	Exponentiation ECMAScript 2016	10 ** 2

12	*	Multiplication	10 * 5
12	/	Division	10 / 5
12	%	Division Remainder	10 % 5
11	+	Addition	10 + 5
11	-	Subtraction	10 - 5
10	<<	Shift Left	x << 2
10	>>	Shift Right (signed)	x >> 2
10	>>>	Shift Right (unsigned)	x >>> 2
9	<	Less than	x < y
9	<=	Less than or equal	x <= y
9	>	Greater than	x > y
9	>=	Greater than or equal	x >= y
9	in	Property in Object	"PI" in Math
9	instanceof	Instance of Object	instanceof Array
8	==	Equal	x == y
8	===	Strict equal	x === y
8	!=	Unequal	x != y
8	!==	Strict unequal	x !== y
7	&	Bitwise AND	x & y
6	^	Bitwise XOR	x ^ y
5	|	Bitwise OR	x | y
4	&&	Logical AND	x && y

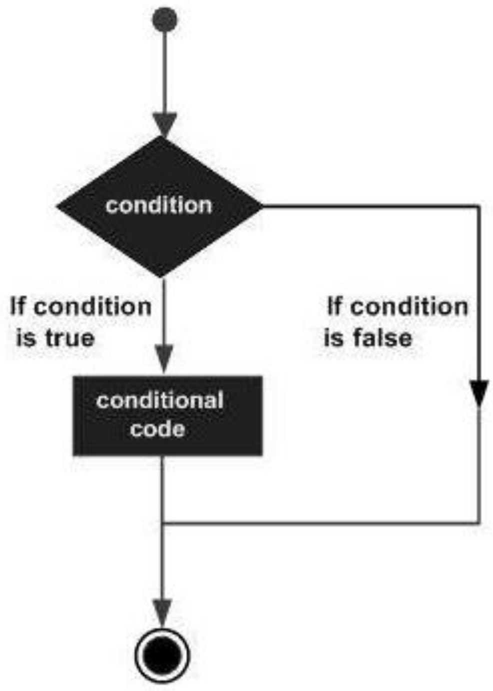
3	||	Logical OR	x || y
3	??	Null Coalescing	x ?? y
2	? :	Condition	? "yes" : "no"
2	=	Simple Assignment	x += y
2	+=	Addition Assignment	x += y
2	-=	Subtraction Assignment	x -= y
2	*=	Multiplication Assignment	x *= y
2	/=	Division Assignment	x /= y
2	%=	Remainder Assignment	x %= y
2	<<=	Left Shift Assignment	x <<= y
2	>>=	Right Shift Assignment	x >>= y
2	>>>=	Unsigned Right Shift	x >>>= y
2	&=	Bitwise AND Assignment	x &= y
2	|=	Bitwise OR Assignment	x |= y
2	^=	Bitwise XOR Assignment	x ^= y
2	&&=	Logical AND Assignment	x &= y
2	||=	Logical OR Assignment	x ||= y
2	=>	Arrow	x => y
2	yield	Pause / Resume	yield x
2	yield*	Delegate	yield* x
2	...	Spread	... x
1	,	Comma	x , y

JavaScript - if...else Statement

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.

Flow Chart of if-else
The following flow chart shows how the if-else statement works.



Decision Making
JavaScript supports the following forms of if..else statement −

- if statement
- if...else statement
- if...else if... statement.

if statement
The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax
The syntax for a basic if statement is as follows −

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

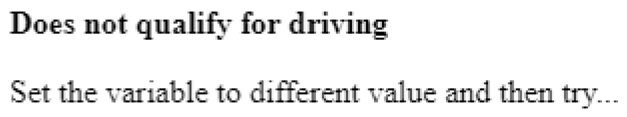
Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

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

        var age = 20;
         if( age > 18 )
        {
          document.write("<b>Qualifies for driving</b>");
        }
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output



```
Does not qualify for driving

Set the variable to different value and then try...
```

if...else statement

The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

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

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
  <body>
    <script type = "text/javascript">
          var age = 15;
          if( age > 18 )
                {
                        document.write("<b>Qualifies for driving</b>");
        }
            else
                {
                        document.write("<b>Does not qualify for driving</b>");
            }
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Does not qualify for driving
Set the variable to different value and then try...
```

if...else if... statement

The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
if (expression 1)
{
  Statement(s) to be executed if expression 1 is true
} else if (expression 2)
{
  Statement(s) to be executed if expression 2 is true
} else if (expression 3)
{
  Statement(s) to be executed if expression 3 is true
} else
{
  Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

```
<html>
  <body>
    <script type = "text/javascript">
        var book = "maths";
        if( book == "history" )
        {
          document.write("<b>History Book</b>");
        } else if( book == "maths" )
        {
          document.write("<b>Maths Book</b>");
        } else if( book == "economics" )
        {
          document.write("<b>Economics Book</b>");
        }
        else
        {
            document.write("<b>Unknown Book</b>");
         }
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
<html>
```

Output

```
Maths Book
Set the variable to different value and then try...
```

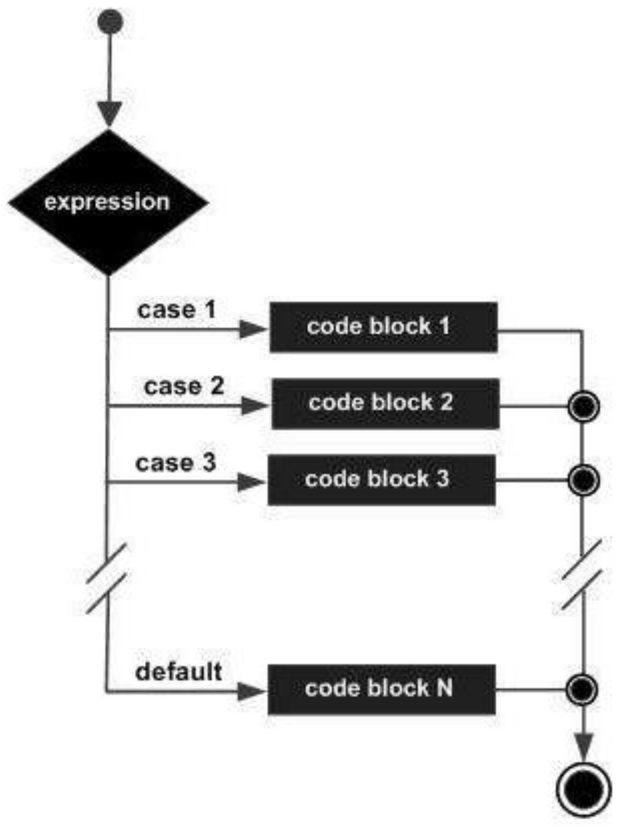
JavaScript - Switch Case

You can use multiple if...else…if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

Flow Chart

The following flow chart explains a switch-case statement works.



Switch case

Syntax

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

```
switch (expression) {
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain break statement in Loop Control chapter.

Example

```
<html>
   <body>
      <script type = "text/javascript">
            var grade = 'A';
            document.write("Entering switch block<br />");
            switch (grade) {
               case 'A': document.write("Good job<br />");
               break;

               case 'B': document.write("Pretty good<br />");
               break;

               case 'C': document.write("Passed<br />");
               break;

               case 'D': document.write("Not so good<br />");
               break;

               case 'F': document.write("Failed<br />");
               break;

               default:  document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering switch block
```

```
Good job
Exiting switch block
Set the variable to different value and then try...
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>
  <body>
    <script type = "text/javascript">
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          case 'B': document.write("Pretty good<br />");
          case 'C': document.write("Passed<br />");
          case 'D': document.write("Not so good<br />");
          case 'F': document.write("Failed<br />");
          default: document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...
```

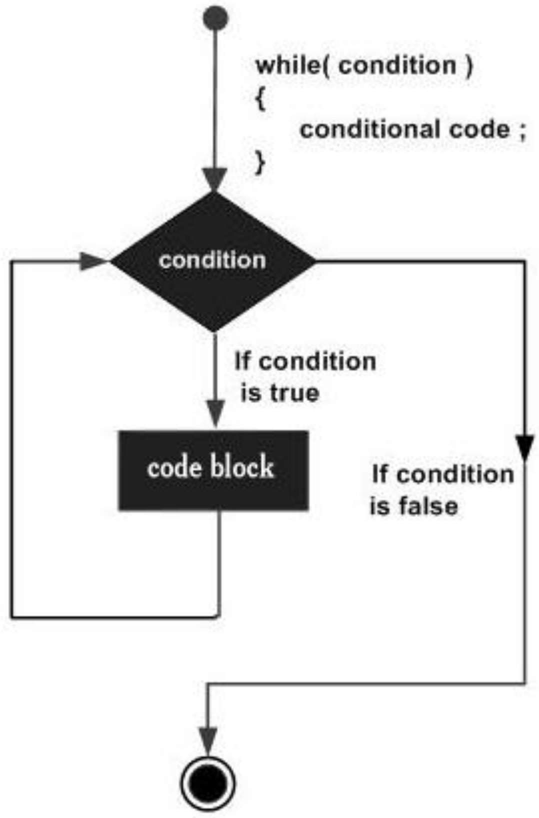
JavaScript - While Loops

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop
The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Flow Chart
The flow chart of while loop looks as follows −



While loop

Syntax
The syntax of while loop in JavaScript is as follows −

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

Example

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

```
    <script type = "text/javascript">
        var count = 0;
        document.write("Starting Loop ");
        while (count < 10) {
           document.write("Current Count : " + count + "<br />");
           count++;
        }
        document.write("Loop stopped!");
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

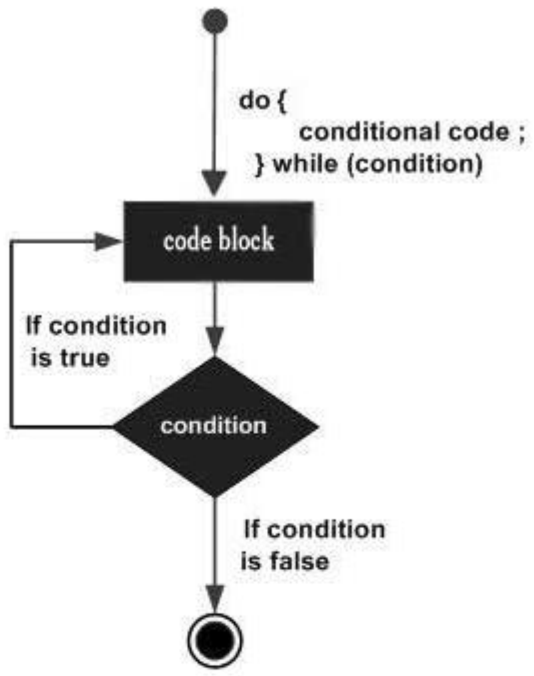
Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

The do...while Loop

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

Flow Chart
The flow chart of a do-while loop would be as follows −



Do While Loop

Syntax

The syntax for do-while loop in JavaScript is as follows −

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

Note − Don’t miss the semicolon used at the end of the do...while loop.

Example

```
<html>
  <body>
    <script type = "text/javascript">
     var count = 0;
        document.write("Starting Loop" + "<br />");
        do {
          document.write("Current Count : " + count + "<br />");
          count++;
        }
        while (count < 5);
        document.write ("Loop stopped!");
        </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

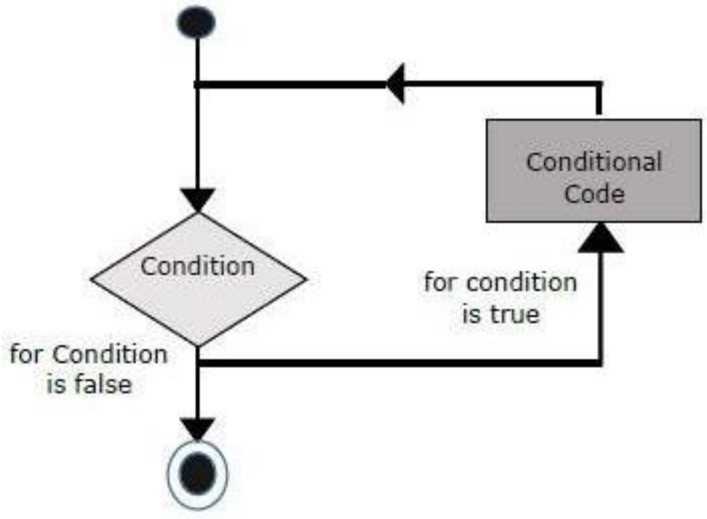
JavaScript - For Loop
The 'for' loop is the most compact form of looping. It includes the following three important parts −
The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart
The flow chart of a for loop in JavaScript would be as follows −



For Loop
Syntax
The syntax of for loop is JavaScript is as follows −

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

Example

```
<html>
  <body>
    <script type = "text/javascript">
      var count;
        document.write("Starting Loop" + "<br />");
```

```
             for(count = 0; count < 10; count++) {
               document.write("Current Count : " + count );
               document.write("<br />");
            }
            document.write("Loop stopped!");
         </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

JavaScript for...in loop

The for...in loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax
The syntax of 'for..in' loop is −

```
for (variablename in object) {
   statement or block to execute
}
```

In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement 'for-in' loop. It prints the web browser's Navigator object.

```
<html>
   <body>
         <script type = "text/javascript">
         var aProperty;
            document.write("Navigator Object Properties<br /> ");
            for (aProperty in navigator) {
               document.write(aProperty);
               document.write("<br />");
            }
            document.write ("Exiting from the loop!");
         </script>
      <p>Set the variable to different object and then try...</p>
   </body>
</html>
```

Output

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
Exiting from the loop!
Set the variable to different object and then try...
```

JavaScript for...of loop
The For Of Loop
The JavaScript for of statement loops through the values of an iterable object.
It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax
```
for (variable of iterable)
        {
          // code block to be executed
        }
```
variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

Browser Support
- For Of Loop was added to JavaScript in 2015 (ES6)
- Browser supporting For Of Loop are
    - ✓ Safari 7
    - ✓ Chrome 38
    - ✓ Edge 12
    - ✓ Firefox 51
    - ✓ Safari 7
    - ✓ Opera 25

- For/of is not supported in Internet Explorer.

The for...of statement executes a loop that operates on a sequence of values sourced from an iterable object. Iterable objects include instances of built-ins such as Array, String, TypedArray, Map, Set, NodeList (and other DOM collections), as well as the arguments object, generators produced by generator functions, and user-defined iterables.

Example
```
const array1 = [10,20,30];
for (x of array1)
{
  console.log(x);
}
```

Output
```
10
20
30
```

JavaScript - Loop Control
JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

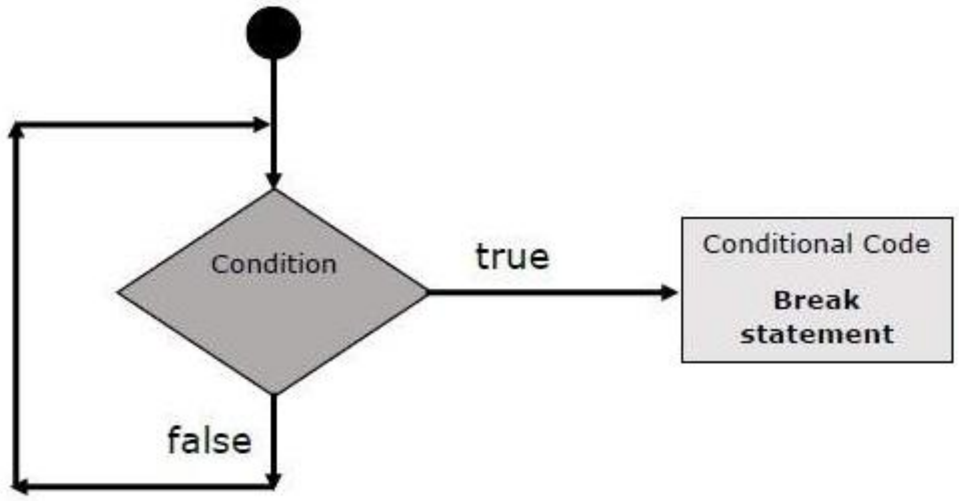
To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows −



Break Statement

Example

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace −

```
<html>
  <body>
    <script type = "text/javascript">
      var x = 1;
       document.write("Entering the loop<br /> ");

       while (x < 20) {
         if (x == 5) {
           break;   // breaks out of loop completely
         }
         x = x + 1;
         document.write( x + "<br />");
       }
       document.write("Exiting the loop!<br /> ");
       </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

The continue Statement

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example
This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 −

```
<html>
   <body>
      <script type = "text/javascript">
       var x = 1;
           document.write("Entering the loop<br /> ");
              while (x < 10) {
              x = x + 1;
              if (x == 5) {
                 continue;   // skip rest of the loop body
              }
              document.write( x + "<br />");
           }
           document.write("Exiting the loop!<br /> ");
         </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering the loop
2
3
4
6
7
8
```

```
9
10
Exiting the loop!
Set the variable to different value and then try...
```

Labels
Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with break and continue to control the flow more precisely. A label is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with break and continue.

Note − Line breaks are not allowed between the ‘continue’ or ‘break’ statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example 1

The following example shows how to implement Label with a break statement.

```
<html>
   <body>
      <script type = "text/javascript">
       document.write("Entering the loop!<br /> ");
            outerloop:         // This is the label name
            for (var i = 0; i < 5; i++) {
               document.write("Outerloop: " + i + "<br />");
               innerloop:      // This is the label name
               for (var j = 0; j < 5; j++) {
                  if (j > 3 ) break ;          // Quit the innermost loop
                  if (i == 2) break innerloop;  // Do the same thing
                  if (i == 4) break outerloop;  // Quit the outer loop
                  document.write("Innerloop: " + j + " <br />");
               }
            }
            document.write("Exiting the loop!<br /> ");
         </script>
   </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
```

```
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!
```

Example 2

```
<html>
   <body>
            <script type = "text/javascript">
         document.write("Entering the loop!<br /> ");
         outerloop:     // This is the label name
         for (var i = 0; i < 3; i++) {
            document.write("Outerloop: " + i + "<br />");
            for (var j = 0; j < 5; j++) {
               if (j == 3) {
                  continue outerloop;
               }
               document.write("Innerloop: " + j + "<br />");
            }
         }
            document.write("Exiting the loop!<br /> ");
        </script>
    </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
```

Exiting the loop!

JavaScript Comments
JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments
Single line comments start with //.

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

This example uses a single-line comment before each code line:

Example
```
let x = 5;      // Declare x, give it the value of 5
let y = x + 2;  // Declare y, give it the value of x + 2
```

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

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

This example uses a multi-line comment (a comment block) to explain the code:

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

JavaScript - Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

Function Definition

Before we use a function, we need to define it. 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.

Syntax

The basic syntax is shown here.

```
<script type = "text/javascript">
    function functionname(parameter-list) {
      statements
    }
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<!DOCTYPE html>
<html>
<body>
<script>
                        abc();

                        function abc()
                        {
                        x=2;
                        y=6;
                        document.write(x**y);
                        }
</script>
</body>
</html>
```

Output

```
64
```

```
<!DOCTYPE html>
<html>
<body>
<script>
var x = mul(40, 30);
document.write(x);
function mul(a, b) {
  return a * b;
}
</script>
</body>
</html>
```

O/P

```
1200
```

Function Parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

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.

JavaScript - Dialog Boxes / Popup Boxes

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

Alert Dialog Box

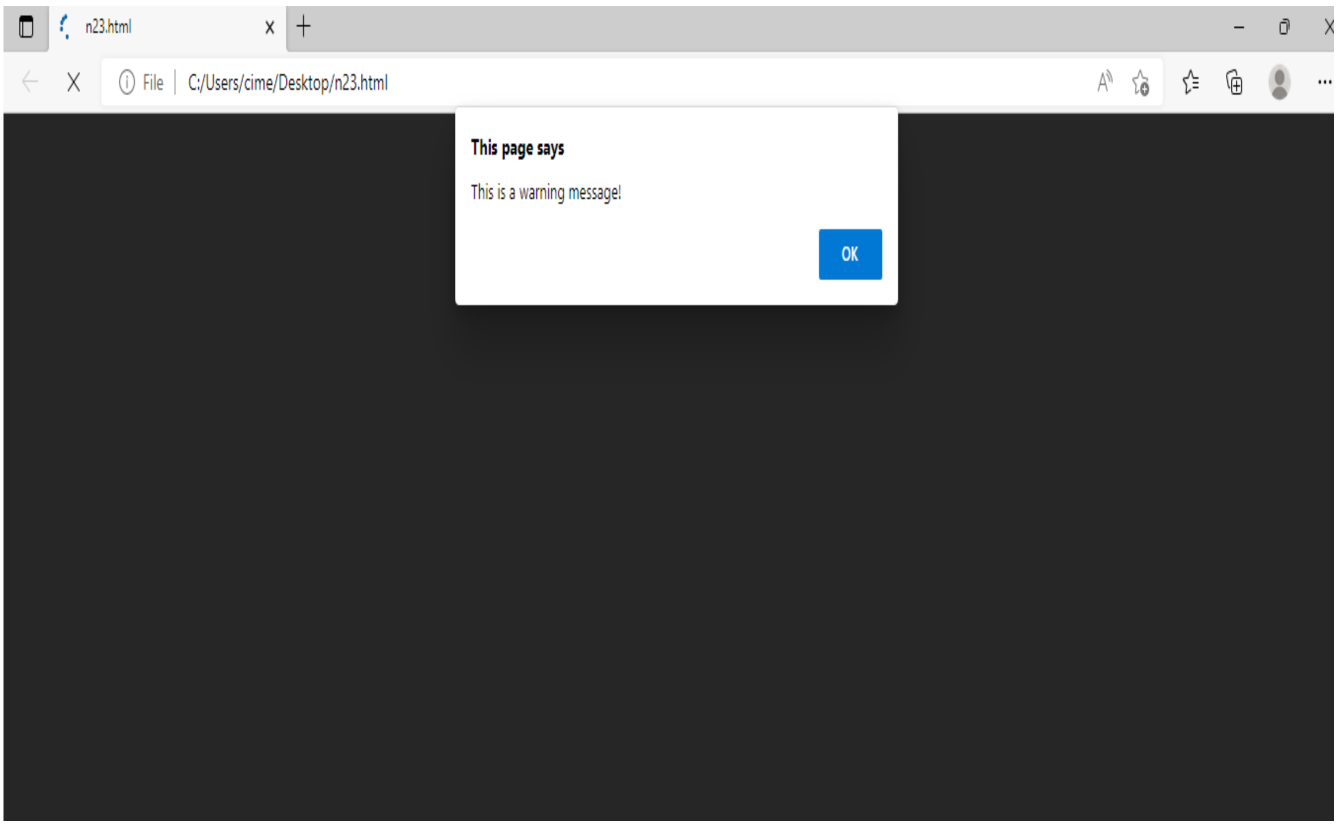
An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example

```
<html>
   <body>
    <script type = "text/javascript">
                    alert ("This is a warning message!");
      </script>
   </body>
</html>
```

O/P



This page says

This is a warning message!

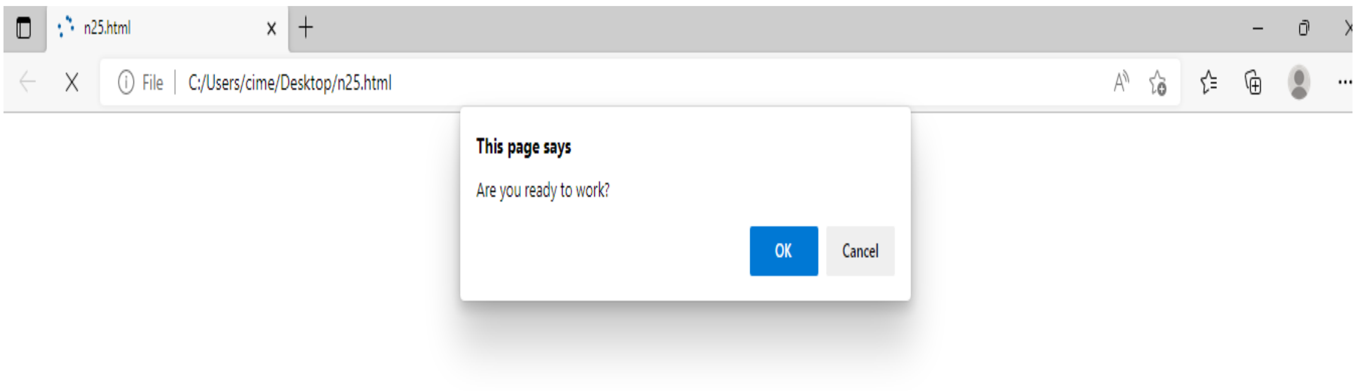
Confirmation Dialog Box
A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false.
Ex

```
<html>
  <body>
   <script type = "text/javascript">
var c=confirm(“Are you ready to work”);

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

O/P


This page says

Are you ready to work?

Prompt Dialog Box
The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

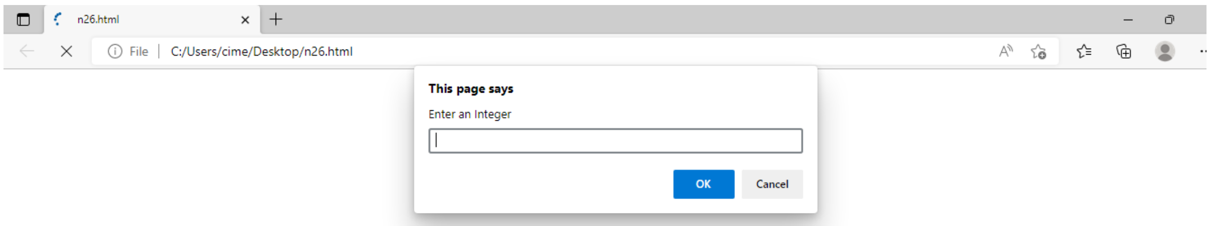
This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.

Ex

```
<html>
  <body>
   <script type = "text/javascript">
var c=prompt("Enter an Integer");

     </script>
  </body>k
</html>
```
O/P



n26.html

C:/Users/cime/Desktop/n26.html

This page says

Enter an Integer

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

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

Why Use Arrays?
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

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

Syntax:

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

It is a common practice to declare arrays with the const keyword.

Ex

```
<!DOCTYPE html>
<html>
<body><script>
const cars = ["Saab", "Volvo", "BMW"];
for(i in cars)
{
document.write(cars[i]+ "<br />");
}
</script>
</body>
</html>
```

O/P

```
Saab
Volvo
BMW
```

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

```
<!DOCTYPE html>
```

```
<html>
<body>
<script>
const cars = new Array("Saab", "Volvo", "BMW");
for(i in cars)
{
document.write(cars[i]+ "<br />");
}
</script>

</body>
</html>
```

O/P

```
Saab
Volvo
BMW
```

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Access the Full Array

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

```
<!DOCTYPE html>
<html>
<body>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.write(cars);
</script>
</body>
</html>
```

O/P

```
Saab,Volvo,BMW
```

Arrays are Objects

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

```
<!DOCTYPE html>
<html>
<body>
<script>
const person = ["John", "Doe",'g',46,6.7,true];
for(i in person)
{
document.write(person[i]+ "<br />");
}
</script>
```

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

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.length);
</script>
</body>
</html>
```

O/P

```
4
```