Unit 1

Basics of JavaScript Programming

Marks: 12 (R-4, U-4, A-4)

Course Outcome: Create interactive web pages using program flow control structure.

Unit Outcome:

1) Create object to solve the given problem.

2) Develop javascript to implement the switch-case statement for the given problem.

3) Develop javascript to implement loop for solving the given iterative problem.

4) Display properties of the given object using getters and setters.

5) Develop program using basic features of javascript to solve the given problem.

Topics and Sub-topics:

1.1 Features of JavaScript

1.2 Object Name, Property, Method, Dot Syntax, Main Event

1.3 Values and Variables

1.4 Operators and Expressions

1.5 if statement, if…else. If…elseif, Nested if

1.6 switch… case statement

1.7 Loop statement

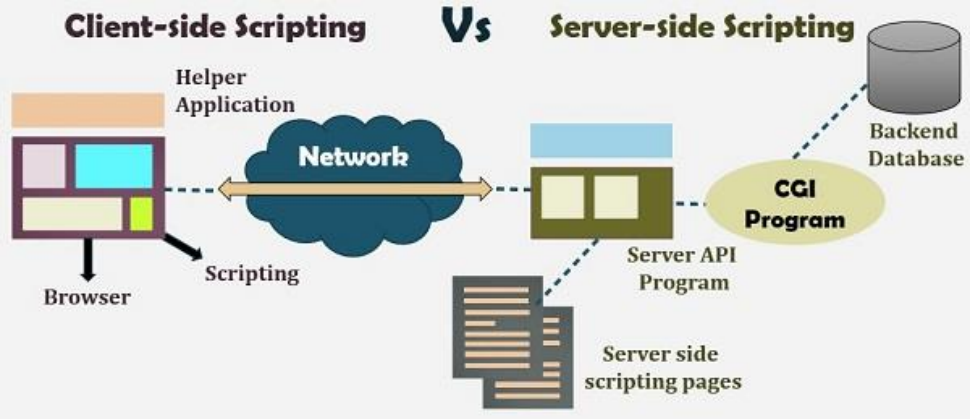
1.8 Querying and setting properties and Deleting properties,

Property Getters and Setters

Introduction:

The scripts can be written in two forms, at the server end (back end) or at the client end (server end). The main difference between server-side scripting and client-side scripting is that the server-side scripting involves server for its processing. On the other hand, client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.

A script is generally a series of program or instruction, which has to be executed on other program or application. As we know that the web works in a client-server environment. The client-side script executes the code to the client side which is visible to the users while a server-side script is executed in the server end which users cannot see.



Source: https://techdifferences.com/difference-between-server-side-scripting-and-client-side-scripting.html

Server-side scripting is a technique of programming for producing the code which can run software on the server side, in simple words any scripting or programming that can run on the web server is known as server-side scripting.

The operations like customization of a website, dynamic change in the website content, response generation to the user's queries, accessing the database, and so on are performed at the server end.

The server-side scripting constructs a communication link between a server and a client (user). Earlier the server-side scripting is implemented by the **CGI (Common Gateway Interface)** scripts. The CGI was devised to execute the scripts from programming languages such as C++ or Perl on the websites.

The server-side involves four parts: server, database, API's and back-end web software developed by the server-side scripting language. When a browser sends a request to the server for a webpage consisting of server-side scripting, the web server processes the script prior to serving the page to the browser. Here the processing of a script could include extracting information from a database, making simple calculations, or choosing the appropriate content that is to be displayed in the client end. The script is being processed and the output is sent to the browser. The web server abstracts the scripts from the end user until serving the content, which makes the data and source code more secure.

The Programming languages for server-side programming are:

1) PHP
2) C++
3) Java and JSP
4) Python
5) Ruby

Client-side scripting is performed to generate a code that can run on the client end (browser) without needing the server-side processing.

Basically, these types of scripts are placed inside an HTML document. The client-side scripting can be used to examine the user's form for the errors before submitting it and for changing the content according to the user input.

In Client-side scripting, the web requires three elements for its functioning which are, client, database and server.

The effective client-side scripting can significantly reduce the **server load**. It is designed to run as a scripting language utilizing a web browser as a host program. For example, when a user makes a request via browser for a webpage to the server, it just sent the HTML and CSS as plain text, and the browser interprets and renders the web content in the client end.

The Programming languages for client-side programming are:

1) Javascript
2) VBScript
3) HTML
4) CSS (Cascading Style Sheet)
5) AJAX

Comparison: Server-side vs. Client-side scripting

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, ASP.net, Ruby on Rails, Python, etc.	HTML, CSS, JavaScript, etc.
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure.

Why we use Javascript?

Using HTML, we can only design a web page but you cannot run any logic on web browser like addition of two numbers, check any condition, looping statements (for, while), decision making statement (if-else) at client side. All these are not possible using HTML So for perform all these tasks at client side you need to use JavaScript.

Where JavaScript is Used?

There are too many web applications running on the web that are using JavaScript technology like Gmail, Facebook, twitter, google map, YouTube etc.

Following are some uses of JavaScript:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time
- Validate user input in an HTML form before sending the data to a server.

- Build forms that respond to user input without accessing a server.
- Change the appearance of HTML documents and dynamically write HTML into separate Windows.
- Open and close new windows or frames.
- Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.
- Build small but complete client-side programs.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks etc.

1.1 Features of JavaScript:

JavaScript is a client-side technology, it is mainly used for gives client-side validation, but it has lot of features which are given below;

- JavaScript is an object-based scripting language.
- Giving the user more control over the browser.
- It Handling dates and time.
- It Detecting the user's browser and OS,
- It is light weighted.
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter-based scripting language.
- JavaScript is case sensitive.
- JavaScript is object-based language as it provides predefined objects.
- Every statement in JavaScript must be terminated with semicolon (;).
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.
- An important part of JavaScript is the ability to create new functions within scripts. Declare a function in JavaScript using **function** keyword.
- The concept of class and OOPs has been more refined. Also, in JavaScript, two important principles with OOP in JavaScript are Object Creation patterns

(**Encapsulation**) and Code Reuse patterns (**Inheritance**). Although JavaScript developers rarely use this feature but its there for everyone to explore.

- JavaScript is platform-independent or we can say it is portable; which simply means that you can simply write the script once and run it anywhere and anytime. In general, you can write your JavaScript applications and run them on any platform or any browser without affecting the output of the Script.

Advantages of JavaScript:

- **Speed:** Client-side JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server.
- **Simplicity:** JavaScript is relatively simple to learn and implement.
- **Popularity:** JavaScript is used everywhere on the web.
- **Interoperability:** JavaScript plays nicely with other languages and can be used in a huge variety of applications.
- **Server Load:** Being client-side reduces the demand on the website server.
- **Light-weight and interpreted:** JavaScript is a lightweight scripting language because it is made for data handling at the browser only. Since it is not a general-purpose language so it has a limited set of libraries. Also, as it is only meant for client-side execution and that too for web applications, hence the lightweight nature of JavaScript is a great feature. JavaScript is an interpreted language which means the script written inside JavaScript is processed line by line. These Scripts are interpreted by JavaScript interpreter which is a built-in component of the Web browser. But these days many JavaScript engines in browsers like the V8 engine in chrome uses just in time compilation for JavaScript code.
- Gives the ability to create rich interfaces.
- **Client-side Validations**: This is a feature which is available in JavaScript since forever and is still widely used because every website has a form in which users enter values, and to make sure that users enter the correct value, we must put proper validations in place, both on the client-side and on the server-side. JavaScript is used for implementing client-side validations.

Disadvantages of JavaScript:

- Security: As the code executes the user's computer, in some cases it can be exploited for malicious purpose.
- Javascript done not read and write the files.
- Javascript can not be used for networking applications.
- Javascript does not have multi-threading and multi-processing capabilities.
- Javascript does not support overloading and overriding.

1.2 **Object Name, Property, Method, Dot Syntax, Main Event:**

JavaScript is an Object based scripting language.

A JavaScript object is a collection of named values.

These named values are usually referred to as properties of the object.

A JavaScript objects are collection of properties and methods.

- ✓ A Methods is a function that is a member of an object.
- ✓ A Property is a value or set of values that is the member of an object.

In JavaScript, almost "everything" is an object.

- ✓ Booleans can be objects (if defined with the new keyword)
- ✓ Numbers can be objects (if defined with the new keyword)
- ✓ Strings can be objects (if defined with the new keyword)
- ✓ Dates are always objects.
- ✓ Maths are always objects
- ✓ Regular expressions are always objects.
- ✓ Arrays are always objects.
- ✓ Functions are always objects.
- ✓ Objects are always objects.

Object Name:

Each object is uniquely identified by a name or ID.

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

A. Define and create a single object, using an object literal.

- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of names: value pairs (like age:10) inside curly braces {}.
- The following example creates a new JavaScript object with 3 properties:

```
var person = {
  firstName: “Hhh",
  lastName: “Bbb",
  age: 10
};
```

In above example, person is an object and firstName , lastName and age are three properties.
“Hhh” , “Bbb” and 10 these are values associated with properties.

Code:

```
<html>
<body>
<script>
emp={id:"VP-179",name:"Aaa Bbb",salary:50000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

Output: **VP-179 Aaa Bbb 50000**

B. Define and create a single object, with the keyword “new” OR by creating instance of Object

new keyword is used to create object.
Syntax: var objectname=new Object();
Example:

```
var person = new Object();
person.firstName = “Hhh";
person.age = 10;
```

Code:

```
<html>
<body>
<script>
var emp=new Object();
```

```
emp.id="VP-179";
emp.name="Aaa ";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

Output: **VP-179 Aaa 50000**

C. Define an object constructor, and then create objects of the constructed type.

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.
Example:

```
function person(firstName, lastName, age)
{
this. firstName = firstName;
this. lastName = lastName;
this. age = age;
}
p=new person("aaa","vvv",10);
document.write(p.firstName+" "+p.lastName+" "+p.age);
```

Code:

```
<html>
 <body>
<script>
function emp(id,name,salary)
{
this.id=id;
this.name=name;
this.salary=salary;
```

```
}
e=new emp("VP-179","Aaa ",50000);
document.write(e.id+" "+e.name+" "+e.salary);
</script>
</body>
 </html>
```

Output:

```
VP-179 Aaa  50000
```

Types of Objects:

- **Native Objects/ Built-in Objects:** are those objects supplied by JavaScript. Examples of these are Math, Date, String, Number, Array, Image, etc.

1) **Math:**

Math Properties

Math	Description
SQRT2	Returns square root of 2.
PI	Returns Π value.
E	Returns Euler's Constant.
LN2	Returns natural logarithm of 2.
LN10	Returns natural logarithm of 10.
LOG2E	Returns base 2 logarithm of E.
LOG10E	Returns 10 logarithm of E.

Code:

```
<html>
<head>
 <title>JavaScript Math Object Properties</title>
 </head>
 <body>
<script type="text/javascript">
var value1 = Math.E;
document.write("E Value is :" + value1 + "<br>");
var value3 = Math.LN10;
```

```
document.write("LN10 Value is :" + value3 + "<br>");
var value4 = Math.PI;
 document.write("PI Value is :" + value4 + "<br>");
</script>
 </body>
</html>
```

Output:

```
E Value is :2.718281828459045
LN10 Value is :2.302585092994046
PI Value is :3.141592653589793
```

Math Methods

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

Code:

```
<html>
<body>
 <script type="text/javascript">
 var value = Math.abs(-20);
document.write("ABS Value : " + value +"<br>");
 var value = Math.tan(5);
 document.write("TAN Value : " + value +"<br>");
 </script>
</body>
</html>
```

Output:

```
ABS Value: 20
TAN Value : -3.380515006246586
```

2) Date

Date is a data type.

Date object manipulates date and time.

Date() constructor takes no arguments.

Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

Syntax:

var variable_name = new Date();

Example:

var current_date = new Date();

Date Methods:

Date Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.
getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.

setMilliseconds()	Sets the milliseconds.
setTime()	Sets the number of milliseconds since January 1, 1970 at 12:00 AM.
setMonth()	Sets the month.
toDateString()	Returns the date portion of the Date as a human-readable string.
toLocaleString()	Returns the Date object as a string.
toGMTString()	Returns the Date object as a string in GMT timezone.
valueOf()	Returns the primitive value of a Date object.

Code:

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

Output:

Date Methods
Locale String: 7/3/2020, 5:23:19 PM
Hours: 17
Day: 5
Month: 6
FullYear: 2020
Minutes: 23

In above code, getMonth() will returns 6 since months starts from 0 that is
0-> January , 1->February
2-> March , 3 ->April
And so on.

3) String

String objects are used to work with text.

It works with a series of characters.

Syntax:

var variable_name = new String(string);

Example:

var s = new String(string);

String Properties:

String properties	Description
length	It returns the length of the string.
constructor	It returns the reference to the String function

String Methods:

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

Code:

```
<html>
 <body>
<script type="text/javascript">
var str = "A JavaScript";
document.write("<b>Char At:</b> " + str.charAt(4)+"<br>");
document.write("<b>CharCode At:</b> " + str.charCodeAt(0)+"<br>");
document.write("<b>Index of:</b> " + str.indexOf("p")+"<br>");
document.write("<b>Lower Case:</b> " + str.toLowerCase()+"<br>");
document.write("<b>Upper Case:</b> " + str.toUpperCase()+"<br>");
 </script>
</body>  </html>
```

Output:

Char At: v
CharCode At: 65
Index of: 10
Lower Case: a javascript
Upper Case: A JAVASCRIPT

- **Host Objects:** are objects that are supplied to JavaScript by the browser environment. Examples of these are window, document, forms, etc.

Window:

The window object represents a window in browser.

An object of window is created automatically by the browser.

Window is the object of browser; it is not the object of javascript.

Window Methods:

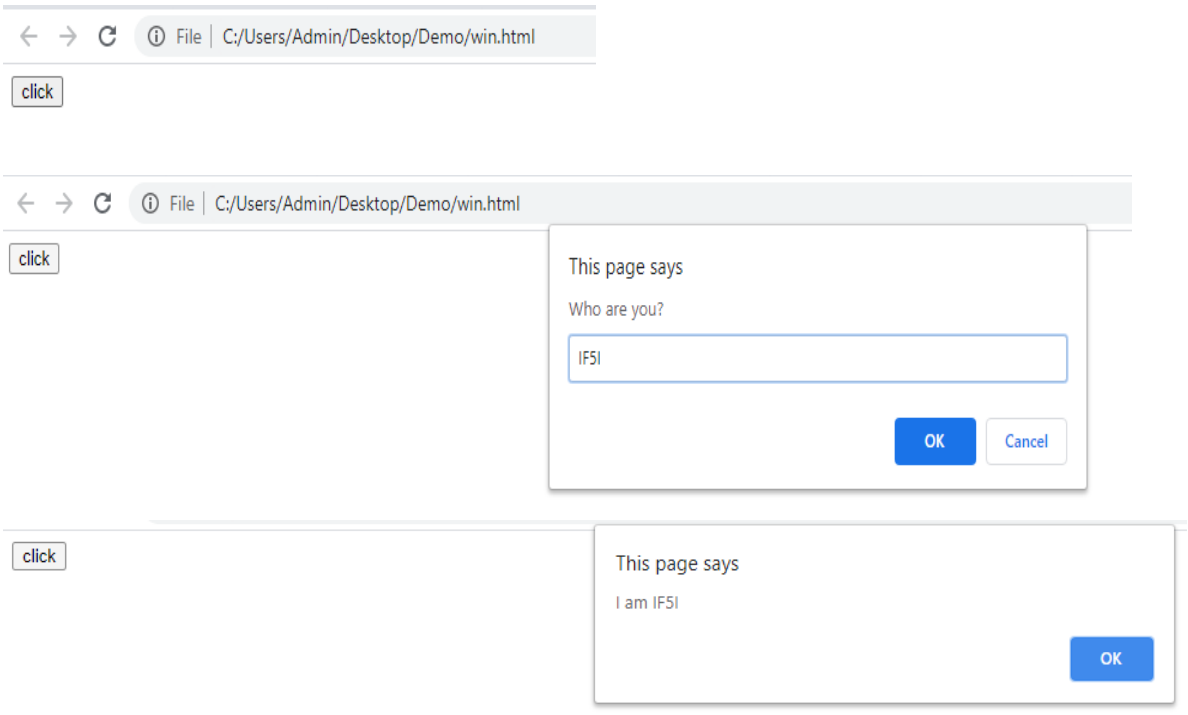
Window methods	Description
alert()	*displays the alert box containing message with ok button.*
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user along with with ok and cancel button.
open()	opens the new window.

close()	closes the current window.

Code:

```
<script type="text/javascript">
function msg()
{
var a= window.prompt("Who are you?");
window.alert("I am "+a);
}
</script>
<input type="button" value="click" onclick="msg()">
```

Output:



- **User-Defined Objects:** are those that are defined by you, the programmer.

Property:

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.

- Properties can usually be changed, added, and deleted, but some are read only.
- The syntax for accessing the property of an object is:

  objectName.property // person.age

  objectName["*property*"] // person["age"]

  objectName[*expression*] // x = "age"; person[x]

Dot Operator:

The properties and methods associated with any object can be accessed by using dot(.) Operator.

Example, emp.id or op.add();

Dot operator is used to how to interact with objects, methods, events and properties.

Dot operator is also used to add new property.

Example, emp.designation="Lecturer";

Code:

```
<html>
<body>
<script>
var person =
{
  firstname:"Hhh",
  age:10
};
person.std = "Fifth";    //adding new property as "std"
document.write(person.firstname+" "+"is in "+person.std+" standard"); //Accessing properties with dot
</script>
</body>  </html>
```

Output:

Hhh is in Fifth standard

Methods:

JavaScript methods are actions that can be performed on objects.

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is defined with the **function keyword**, followed by a **name**, followed by parentheses **().**
The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)
The code to be executed, by the function, is placed inside curly brackets: {}
Syntax:

```
function name(parameter1, parameter2, parameter3)
{
  // code to be executed
}
```

Code: simple method to define a function

```
<html>
<body>
<script>
function op_add(p1, p2)
{
  return p1 + p2;
}
document.write("Addition is="+op_add(4, 5));
</script>
</body>
</html>
```

Output:

Addition is=9

Code: define a function as a property

```
<script>
var person =
{
  firstname:"Hhh",
```

```
 lastname:"Bbb",
 Fullname:function()  // define a function as a property
 {
  return this.firstname+" "+this.lastname;
  }
};
document.write("Person Detail is="+person.Fullname());
</script>
```

Output:

```
Person Detail is=Hhh Bbb
```

Code: define a function as an expression

```
<script>
var x = function (a, b)    //  function as an expression
{
return a * b ;
}
document.write("function returns= " +x(4, 5));
</script>
```

Output:

```
function returns= 20
```

Main Event:

- An event is an action performed by user or web browser.
- In order to make a web pages more interactive, the script needs to be accessed the contents of the document and know when the user is interacting with it.
- Events may occur due to: 1) a document loading
  2) user clicking on mouse button
  3) browser screen changing size

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed

- An HTML button was clicked

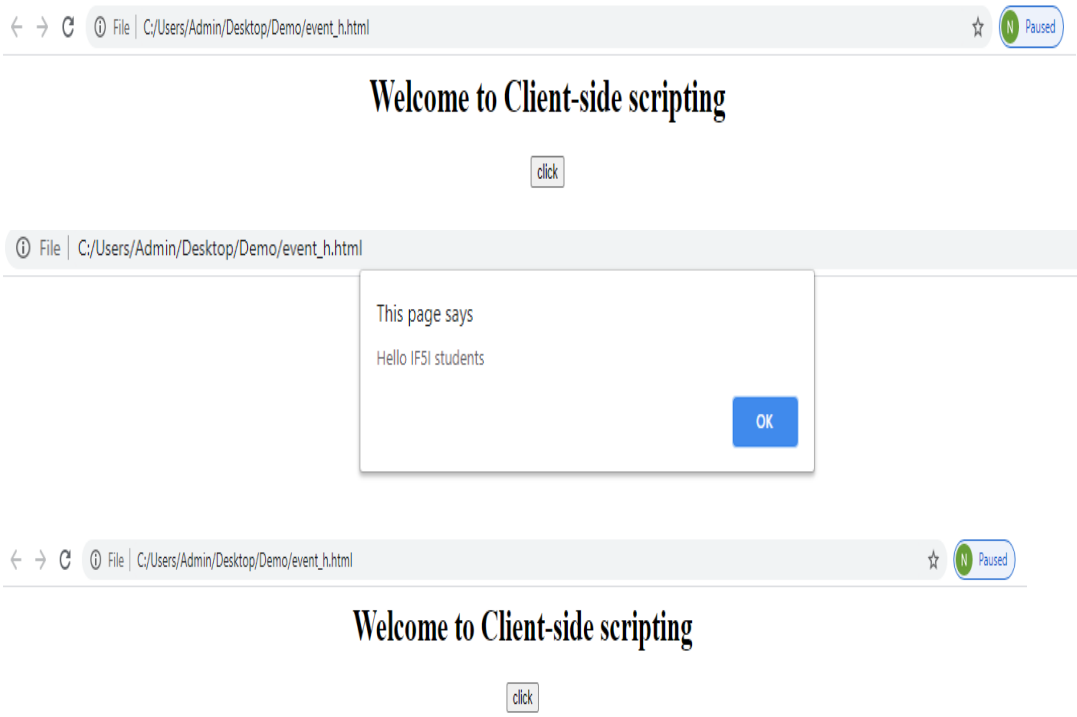
Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Code:

```
<html>
<head>
<script type="text/javascript">
function msg()
{
 alert("Hello IF5I students");
}
</script>
</head>
<body>
<center>
<p><h1>Welcome to Client-side scripting</h1></p>
<form>
<input type="button" value="click" onclick ="msg()"/>      // onclick event
</form>
</body>
</html>
```

Output:



DOM getElementById() Method

The getElementById() method returns the elements that has given ID which is passed to the function.

This function is widely used in web designing to change the value of any particular element or get a particular element.

Syntax: document. getElementById(element_id) ;

Parameter: This function accepts single parameter *element_id* which is used to hold the ID of element.

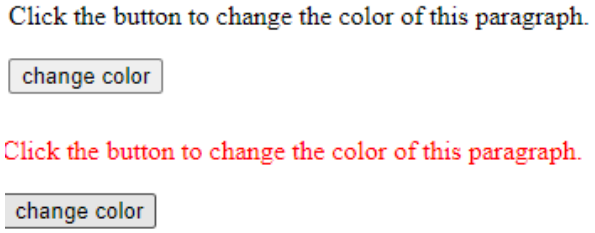
Return Value: It returns the object of given ID. If no element exists with given ID then it returns null.

Code:

```
<html>
<body>
<p id="demo">Click the button to change the color of this paragraph.</p>
<button onclick="myFunction()">change color</button>
<script>
function myFunction()
{
  var x = document.getElementById("demo");
```

```
 x.style.color = "red";
}
</script>
</body>
</html>
```

Output:



1.3 Values and Variables

- In JavaScript there are two types of scope:
  - Local scope
  - Global scope
- JavaScript has function scope: Each function creates a new scope.
- Scope determines the accessibility (visibility) of these variables.
- Variables defined inside a function are not accessible (visible) from outside the function.

There are some rules while declaring a JavaScript variable (also known as identifiers).

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.

We can say that variable is a container that can be used to store value and you need to declare a variable before using it.

In JavaScript, the var keyword is used to declare a variable. Also, starting from ES6 we can also declare variables using the let keyword.

JavaScript Syntax for Declaring a Variable

Following is the syntax for declaring a variable and assigning values to it.

```
var varible-name;
or
let varible-name;
```

We can also define a variable without using the semicolon too. Also, when we have to define multiple variables, we can do it like this:

var x,y,z;
or
var x,y,z

JavaScript Variable Example:

Now let's take a simple example where we will declare a variable and then assign it a value.

var emp_name;

var emp_name=”dhavan”;

JavaScript: Types of Variables

JavaScript supports two types of variables, they are:

- Local Variable
- Global Variable

You can use them according to the requirement in the application. Let's learn about both JavaScript Local variables and JavaScript Global variables with examples.

1. JavaScript Local Variable

JavaScript Local variable is a variable that is **declared inside a code block or a function body or inside a loop body** and it has scope within the code block or the function. In simple words, the scope of a local variable is between the opening and closing curly braces { }, when declared and defined inside a code block or a function body.

Starting from ES6 it is recommended to use the let keyword while declaring local variables.

A JavaScript local variable is declared inside block or function.

It is accessible within the function or block only.

For example:

```
<script>
function abc()
{
var x=10;     //x is a local variable
```

```
}
</script>
```

JavaScript Global Variable

JavaScript Global Variable is a variable that is **declared anywhere inside the script** and **has scope for the complete script execution**. Global variables are not declared inside any block or function but can be used in any function, or block of code.

Its recommended that we use the var keyword to declare the global variables, starting from ES6.

A JavaScript global variable is accessible from any function.

A variable i.e. declared outside the function or declared with window object is known as global variable.

Code:

```
<html>
<body>
<script>
var data=200;      //gloabal variable
function a()
{
document.write(data);
}
function b()
{
document.write(data);
}
a();      //calling JavaScript function
b();
</script>
</body>
</html>
```

Output:

200 200

Javascript let Keyword

In JavaScript, let is a keyword which is used to declare a local variable with block scope. Unlike var keyword which declares global scope variables, with **ECMAScript2016**(ES6) the let keyword was introduced to define local scope variables as defining all the variables in global scope leads to a lot of problems while writing large JavaScript applications.

It allows you to declare **limited scope variables** that cannot be accessible outside of the scope.

Let's take an example to understand the need of let keyword when we already had var keyword to create variables in JavaScript. Suppose you are writing a big JavaScript code which involves multiple loops, functions, and variables, as usual in all the loops you used the same variable name for the counter which is i, and you used the var keyword to define the variable i, now what will happen is, the variable i will carry on its changed value throughout the whole script as it is a global variable and if you forget to re-initialize it to zero anywhere, it will cause an error and your code will break. And you will have to put in extra efforts to look for the error.

Whereas, if you define the counter variable i using the let keyword, its scope will be limited to the loop only and when the first loop will end so will the counter variable. This way, using let keyword makes more sense because we use very few global variables and many local variables in general programming practice.

let does not create properties of the window object when declared globally.

The syntax for using the let keyword for defining a variable is the same as that of the var keyword.

let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN];

As shown in the syntax above, yes, we can use a single let keyword **to define multiple variables**, but this is not new, we can do so using the var keyword too.

Let's see a few examples to see how this let keyword is used and how it works.

Use let inside a code block:

JavaScript variable declared inside a **block** { } cannot be accessed from outside the block, if we have defined the variable using the let keyword. See the below example:

```
{
   let x = 2;
}
alert(x)  // not accessible
```

Output:

```
uncaught ReferenceError: x is not defined
```

In the above example, the variable is accessible only inside the block. See the below example, to see that:

```
{
    let x = 2;
    alert(x)  // accessible
}
```

Output:

```
2
```

Use let inside a Loop:

It is more suitable for a loop, where we declare local variables to be used as counters. So, the variable does not conflict with the code written outside of the loop. See the below example:

```
let i = 5;
for(let i = 0; i < 10; i++) {
      // code
}
alert(i);  // print 5
```

Output:

```
5
```

As you can see in the output it shows **5**, even though the loop incremented the value of i variable up to **10**, that is because of the scope of the local variable i in the for loop ending with the loop itself, hence it is not accessible outside the loop.

Use let inside a Function:

As we know, let keyword declares the local scope variable. So variable declared inside the function will retain within the function scope. If we will try accessing such variables from outside the function, we will get an error. See the below example:

```
function show()
{
```

```
    let amount = 2500;   // Function Scope
}
alert(amount) // not accessible
```

Output:

```
Uncaught ReferenceError: amount is not defined
```

JavaScript let vs var Keyword

The let and var, both keywords are used to declare variables, but the main difference is the scope of the declared variables.

A variable declared inside a block using var is accessible outside of the block as it has a global scope but a variable declared using the let keyword has a local scope. Let's see an example:

```
{
    let amount = 2500;   // block Scope
    var withdraw = 2000;    // global scope
}
document.write(withdraw)   // accessible
document.write(amount)   // not accessible
```

Output:

```
2000
Uncaught ReferenceError: amount is not defined
```

JavaScript const keyword is used to define constant values that cannot changed once a value is set. The value of a constant can't be changed through reassignment, and it can't be redeclared.

The scope of **const** is block-scoped it means it cannot be accessed from outside of block. In case of scope, it is much like variables defined using the **let** statement.

Constants can be either global or local to the block in which it is declared. Global constants do not become properties of the window object, unlike var variables.

JavaScript const **Keyword:**

Syntax

Below we have the syntax to define a constant value in JavaScript.

```
const name1 = value1 [, name2 = value2 [, ... [, nameN = valueN]]]
```

We don't have to use var or let keyword while using the const keyword. Also, we can **define a list or a simple value or a string etc as a constant**.

Lets understand, how to create constant in JavaScript program. See the below example:

```
{
        const Pi = 3.14;
        alert(Pi);

}
```

Output:

```
3.14
```

Let's try another example where we will try changing the value of the constant and see if we allowed to reassign value or not.

```
{
        const Pi = 3.14;
        alert(Pi);
        // Reassign value
        Pi = 3.143;
        alert(Pi);
}
```

Output:

```
3.14
Uncaught TypeError: Assignment to constant variable.
```

The scope of the variable defined using const keyword is same as that of a variable declared using the let keyword. So, **constant declared inside a block will not accessible outside of that block**. Let's take an example and see:

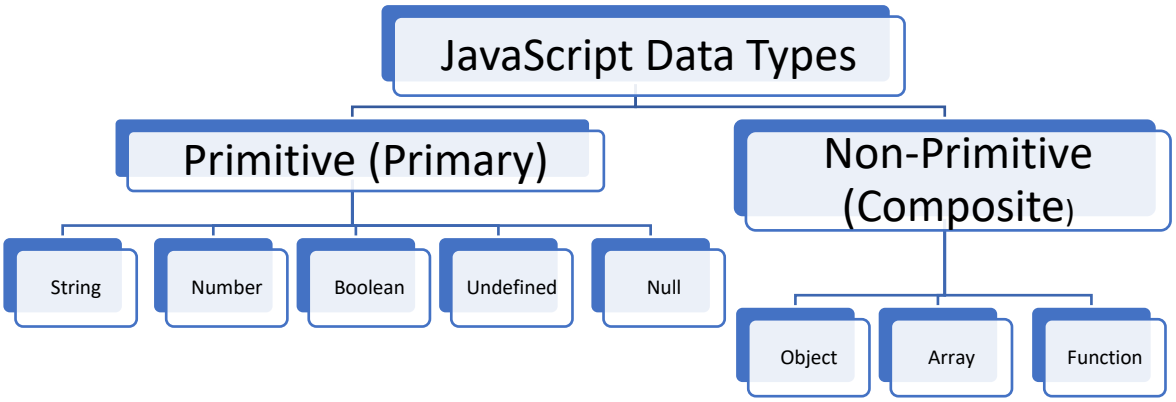
```
{
        const Pi = 3.14;
        alert(Pi);
}

// outside block

alert(Pi);  // error
```

Output:

```
3.14
Uncaught ReferenceError: Pi is not defined
```

Data Types

- JavaScript provides different data types to hold different types of values.
- There are two types of data types in JavaScript:
    - Primitive data type
    - Non-primitive (reference) data type/ Composit Data Types
- JavaScript is a dynamic type language; means you don't need to specify type of the variable.
- You need to use var here to specify the data type.
- It can hold any type of values such as numbers, strings etc.
- For example: var a=40;//holding number
    - var b="Info Technology"//holding string



Data Types: Primitive

Primitive data types can hold only one value at a time.

1) The String Data Type

The *string* data type is used to represent textual data (i.e. sequences of characters).

Strings are created using single or double quotes surrounding one or more characters, as shown below:

var a = 'Welcome'; // using single quotes

var b = "Welcome";// using double quotes

2) The Number Data Type

- ✓ The *number* data type is used to represent positive or negative numbers with or without decimal place.
- ✓ The Number data type also includes some special values which are: Infinity,-Infinity, NaN(Not a Number)
- ✓ Example,

var a = 25; // integer

```
var b = 80.5; // floating-point number
var c = 4.25e+6; // exponential notation, same as 4.25e6 or 4250000
var d = 4.25e-6; // exponential notation, same as 0.00000425
```

3) The Boolean Data Type

- ✓ The Boolean data type can hold only two values: True/False
- ✓ Example,

```
var a = 2, b = 5, c = 10;
alert(b > a) // Output: true
alert(b > c) // Output: false
```

4) The Undefined Data Type

- ✓ The undefined data type can only have one value-the special value "undefined".
- ✓ If a variable has been declared, but has not been assigned a value, has the value "undefined".
- ✓ Example,

```
var a;
var b = "Welcome";
alert(a) // Output: undefined
alert(b) // Output: Welcome
```

5) The Null Data Type

- ✓ A Null value means that there is no value.
- ✓ It is not equivalent to an empty string (" ") or zero or it is simply nothing.
- ✓ Example,

```
var a = null;
alert(a); // Output: null
var b = "Hello World!"
alert(b); // Output: Hello World!
b = null;
alert(b) // Output: null
```

Data Types: Non-primitive

1) The Object Data Type

- ✓ a complex data type that allows you to store collections of data.

- ✓ An object contains properties, defined as a key-value pair.
- ✓ A property key (name) is always a string, but the value can be any data type, like strings, numbers, Boolean, or complex data types like arrays, function and other objects.
- ✓ Example,
  var car =
  { model: "SUZUKI", color: "WHITE", model_year: 2019 }

2) The Array Data Type

- ✓ An array is a type of object used for storing multiple values in single variable.
- ✓ Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, Booleans, functions, objects, and even other arrays.
- ✓ The array index starts from 0, so that the first array element is arr [0].
- ✓ The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:
- ✓ var cities = ["London", "Paris", "New York"];
- ✓ alert(cities[2]); // Output: New York
- ✓ var a = ["London", 500, "aa56", 5.6];

3) The Function Data Type

- ✓ The function is callable object that executes a block of code.
- ✓ Since functions are objects, so it is possible to assign them to variables, as shown in the example below:
  var ab = function()
  {
  return "Welcome";
  }
  alert(typeof ab); //output: function
  alert(ab()); //output:Welcome

Code:

```
<html>
<body>
<h1>JavaScript Array</h1>
```

```
<script>
var stringArray = ["one", "two", "three"];
var mixedArray = [1, "two", "three", 4];
document.write(stringArray+"<br>");
document.write( mixedArray);
</script>
</body>
</html>
```

JavaScript Array
one,two,three
1,two,three,4

Output:

Values/Literals

They are **types** that can be assigned a single **literal** value such as the number 5.7, or a string of characters such as "hello".

Types of Literals:

- Array Literal
- Integer Literal
- Floating number Literal
- Boolean Literal (include True and False)
- Object Literal
- String Literal

Array Literal:

- an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [ ] ' .
- When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified.
- Creating an empty array :

  var tv = [ ];

  Creating an array with four elements.

  var tv = ["LG", "Samsung", "Sony", "Panasonic"]

✓ **Comma in array literals:**

- In the following example, the length of the array is four, and tv[0] and tv[2] are undefined.
- var tv = [, "Samsung", , "Panasonic"]
- This array has one empty element in the middle and two elements with values.

(tv[0] is “LG", tv[1] is set to undefined, and tv[2] is “Sony")

Var tv = [“LG", ,“Sony",]

Integer Literal:

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)

Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

Example: 123, -20, 12345

2. Hexadecimal (base 16)

Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f.

A leading 0x or 0X indicates the number is hexadecimal.

Example: 7b, -14, 3039

3. Octal (base 8)

Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Example: 0173, -24, 30071

Floating number Literal:

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Example of some floating numbers:

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4 x 10^3]
- 4E-3 [Equivalent to 4 x 10^{-3} => .004]

Object Literal:

An object literal is zero or more pairs of comma-separated list of property names and associated values, enclosed by a pair of curly braces.
In JavaScript an object literal is declared as follows:

1. An object literal without properties:

```
var userObject = { }
```

2. An object literal with a few properties :

```
var student = {
First-name : "Suresy",
Last-name : "Rayy",
Roll-No : 12
};
```

Syntax Rules

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.
- There will be no comma after the last property name/value pair.

String Literal:

- JavaScript has its own way to deal with string literals.
- A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings.
- The following are the examples of string literals:
  - string1 = "w3resource.com"
    string1 = 'w3resource.com'
  - string1 = "1000"
- In addition to ordinary characters, you can include special characters in strings, as shown in the following.
  - string1 = "First line. \n Second line."

Special characters in JavaScript:

character	Description
\'	Single quote
\"	Double quote
\\	Backslash

\b	Backspace
\f	Form Feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Comments in JavaScript:

The **JavaScript comments** are meaningful way to deliver message.
It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.
There are two types of comments in JavaScript.

1. Single-line Comment

It is represented by double forward slashes (//).
It can be used before and after the statement.
Example,

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

2. Multi-line Comment

It can be used to add single as well as multi line comments.
It is represented by forward slash with asterisk then asterisk with forward slash.
Example,

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

1.4 Operators and Expression

JavaScript operators are symbols that are used to perform operations on operands.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

1) Arithmetic Operators: used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus	20%10 = 0
++	Increment	var a=10; a++;
--	Decrement	var a=10; a--;

Code:

```
<html>
<body>
<script type = "text/javascript">
        var a = 33;
        var b = 10;
        var c = "Test";
        document.write("a + b = ");
        result = a + b;
        document.write(result+"<br>");
```

```
        document.write("a - b = ");
        result = a - b;
        document.write(result+"<br>");

        document.write("a / b = ");
        result = a / b;
        document.write(result+"<br>");

        document.write("a % b = ");
        result = a % b;
        document.write(result+"<br>");

        document.write("a + b + c = ");
        result = a + b + c;
        document.write(result+"<br>");

        a = ++a;
        document.write("++a = ");
        result = ++a;
        document.write(result+"<br>");

        b = --b;
        document.write("--b = ");
        result = --b;
        document.write(result+"<br>");
    </script>
</body>
</html>
```

Output:

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
```

2) Comparison (Relational) Operators: compares the two operands

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Code:

```
<html>
<body>
<script type = "text/javascript">
        var a = 10;
        var b = 20;

        document.write("(a == b) => ");
        result = (a == b);
        document.write(result+"<br>");

        document.write("(a < b) => ");
        result = (a < b);
        document.write(result+"<br>");

        document.write("(a > b) => ");
        result = (a > b);
        document.write(result+"<br>");
```

```
        document.write("(a != b) => ");
        result = (a != b);
        document.write(result+"<br>");

        document.write("(a >= b) => ");
        result = (a >= b);
        document.write(result+"<br>");

        document.write("(a <= b) => ");
        result = (a <= b);
        document.write(result+"<br>");
        </script>
</body>
</html>
```

Output:

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
```

3) **Bitwise Operator**: perform bitwise operations on operands

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10

<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

Code:

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

        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11

        document.write("(a & b) => ");
        result = (a & b);
        document.write(result+"<br>");

        document.write("(a | b) => ");
        result = (a | b);
        document.write(result+"<br>");

        document.write("(a ^ b) => ");
        result = (a ^ b);
        document.write(result+"<br>");

        document.write("(~b) => ");
        result = (~b);
        document.write(result+"<br>");

        document.write("(a << b) => ");
        result = (a << b);
        document.write(result+"<br>");
```

```
        document.write("(a >> b) => ");
        result = (a >> b);
        document.write(result+"<br>");
        </script>
</body>
</html>
```

Output:

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
```

4) **Logical Operator**:

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Code:

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

        var a = true;
        var b = false;

        document.write("(a && b) => ");
        result = (a && b);
```

```
            document.write(result+"<br>");

            document.write("(a || b) => ");
            result = (a || b);
            document.write(result+"<br>");

            document.write("!(a && b) => ");
            result = (!(a && b));
            document.write(result+"<br>");
          </script>
    </body>
    </html>
```

Output:

```
(a && b) => false
(a || b) => true
!(a && b) => true
```

```
<!DOCTYPE html>
<html>
<body>
        <h1>Demo: JavaScript Logical Operators</h1>
        <p id="p1"></p>
        <p id="p2"></p>
        <p id="p3"></p>
        <p id="p4"></p>
        <p id="p5"></p>
        <script>
                var a = 5, b = 10;
                document.getElementById("p1").innerHTML = (a != b) && (a < b);
                document.getElementById("p2").innerHTML = (a > b) || (a == b);
                document.getElementById("p3").innerHTML = (a < b) || (a == b);
                document.getElementById("p4").innerHTML = !(a < b);
                document.getElementById("p5").innerHTML = !(a > b);
   </script>
</body>
</html>
```

Demo: JavaScript Logical Operators

true

false

true

false

true

5) **Assignment Operator**:

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Code:

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

        var a = 33;
        var b = 10;

        document.write("Value of a => (a = b) => ");
        result = (a = b);
        document.write(result+"<br>");

        document.write("Value of a => (a += b) => ");
        result = (a += b);
        document.write(result+"<br>");
```

```
        document.write("Value of a => (a -= b) => ");
        result = (a -= b);
        document.write(result+"<br>");

        document.write("Value of a => (a *= b) => ");
        result = (a *= b);
        document.write(result+"<br>");

        document.write("Value of a => (a /= b) => ");
        result = (a /= b);
        document.write(result+"<br>");

        document.write("Value of a => (a %= b) => ");
        result = (a %= b);
        document.write(result+"<br>");
        </script>
</body>
</html>
```

Output:

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
```

6) **Special Operator**:

Operator	Description
(?:)	Conditional Operator/ternary returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement
delete	Delete Operator deletes a property from the object
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object
void	it discards the expression's return value

Code: *typeof* operator

```
<html>
<body>
 <script type = "text/javascript">
        var a = 10;
        var b = "Information";
        var c= function(x)
        {
        return x*x;
        }

        document.write(typeof a+"<br>");  // a=10  datatype is number
        document.write(typeof b+"<br>");  // b="Information" is a string
        document.write(typeof c+"<br>");   // c= function
        document.write(c(4));
```

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

Output:

```
number
string
function
16
```

Code: *Ternary (? :)* operator

```
    <script type = "text/javascript">
      var a = 10;
      var b = 20;
      r=(a>=b)? "a is large":"b is large";
      document.write(r);
    </script>
```

```
<script type = "text/javascript">
    var a = 10;
    var b = 20;
    if(a>=b)
    document.write("a is large");
    else
    document.write("b is large");
</script>
```

Output:

```
b is large
```

In above example, if else can be replaced with ternary operator.

JavaScript Operator Precedence and Associativity

Operator precedence determines the order in which operators are evaluated. Operators with higher precedence are evaluated first. For example, the expression `(3+4*5)`, returns **23**, because of multiplication operator(`*`) having **higher precedence** than addition(`+`). Thus `*` must be evaluated first.

Operator associativity determines the order in which operators of the same precedence are processed. For example, **assignment operators are right-associative**, so you can write `a=b=5`, and with this statement, `a` and `b` are assigned the value **5**.

The below table shows the precedence and associativity of operators. In this table, precedence is from bottom to top i.e items at the bottom having low precedence and precedence increases as we move to the top of the table.

Operator type	**Operator (Symbol)**	**Associativity**
member	. []	left-to-right
new	new	right-to-left
function call	()	left-to-right
increment	++	
decrement	--	
logical-not	!	right-to-left
bitwise not	~	right-to-left
unary +	+	right-to-left
unary negation	-	right-to-left
typeof	typeof	right-to-left
void	void	right-to-left
delete	delete	right-to-left
multiplication	*	left to right
division	/	left to right

Operator type	**Operator (Symbol)**	**Associativity**
modulus	%	left to right
addition	+	left to right
subtraction	-	left to right
bitwise-shift	<< >> >>>	left to right
relational	< <= > >=	left to right
in	in	left to right
instanceof	instanceof	left to right
equality	== != === !==	left to right
bitwise-and	&	left to right
bitwise-xor	^	left to right

Operator type	Operator (Symbol)	Associativity
bitwise-or	|	left to right
logical-and	&&	left to right
logical-or	||	left to right
conditional	?:	right to left
assignment	= += -= *= /= %= <<= >>= >>>= &= ^= |=	right to left
comma	,	left to right

Expression:

Any unit of code that can be evaluated to a value is an expression.

Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation.

Types of Expression:

1. **Primary Expression:**

   Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values.

```
'hello world'; // A string literal
23;            // A numeric literal
true;          // Boolean value true
sum;           // Value of variable sum
this;          // A keyword that evaluates to the current object.
```

2. **Object and Array Initializers**

   Object and array initializers are expressions whose value is a newly created object or array.
   Object initializer expressions uses curly brackets, and each subexpression is prefixed with a property name and a colon.
   Example, var emp={ name:"Aaa", branch:"IF"};

```
OR
var person={ };
person.name="Aaa";
person.branch="IF";
```

   An array initializer is a comma-separated list of expressions surrounded with a square bracket.
   Example, var tv=["LG", "Samsung"];

3. **Property Access Expressions**

   A property access expression evaluates to the value of an object property or an array element.
   JavaScript defines two syntaxes for property access:

```
expression.identifier;
expression[identifier];
```

   Example:

```
emp.firstName;
emp[lastName];
```

4. Function Definition Expression

A function expression is part of a variable assignment expression and may or may not contain a name.
Since this type of function appears after the assignment operator =, it is evaluated as an expression.
Function expressions are typically used to assign a function to a variable.
Function expressions are evaluated only when the interpreter reaches the line of code where function expressions are located.
Example:

```
var sq=function (x)
{ return x*x;
}
```

5. Invocation Expression

An *invocation expression* is JavaScript's syntax for calling (or executing) a function or method.

It starts with a function expression that identifies the function to be called.

The function expression is followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis.

When an invocation expression is evaluated, the function expression is evaluated first, and then the argument expressions are evaluated to produce a list of argument values.

Example,

```
f(0) // f is the function expression; 0 is the argument expression

Math.max(x,y,z)       // Math.max is the function; x, y and z are the arguments.
```

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

```
var obj = { add: function(a, b)
{ return a + b; }
};alert(obj.add(4,5));delete obj.add;alert(obj.add(9,5)); </script>
```

1.5 if statement, if...else. If...elseif, Nested if

Conditional statements are used to perform different actions based on different conditions. In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

1) if statement:

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

Syntax:

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

Example:

```
<html>
<body>
<script>
if (new Date().getHours() < 18)
{
  document.write("Good day!");
}
</script>
</body>
</html>
```

2) The else Statement

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

Syntax:

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

Example:

```
<html>
 <body>
<script>
if (new Date().getHours() < 18)
{
  document.write("Good day!");
}
else
{
document.write("Good Evening!");
}
</script>
</body>
</html>
```

3) The else if Statement

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

Syntax:

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

Example:

```
<html>
<body>
<script>
 var greeting;
 var time = new Date().getHours();
 if (time < 10)
 {
  greeting = "Good morning";
 }
 else if (time < 20)
 {
greeting = "Good day";
 }
 else
 {
  greeting = "Good evening";
 }
 document.write(greeting);
</script>
</body>
</html>
```

4) The switch case Statement

The switch statement is used to perform different actions based on different conditions. It is used to select one of many code blocks to be executed.

Syntax:

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

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example:

```
<html>
<body>
<script>
var day;
switch (new Date().getDay())
{
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
break;
case 3:
    day = "Wednesday";
```

```
    break;
 day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.write("Today is " + day);
</script>
</body>
</html>
```

default keyword:

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

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message.

Example:

```
switch (new Date().getDay())
{
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

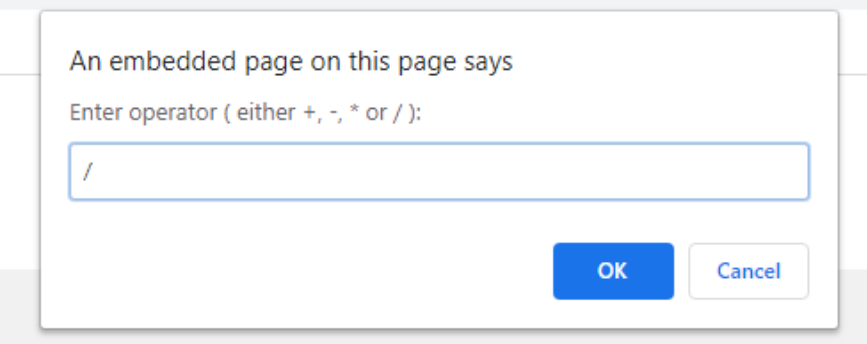
Example:

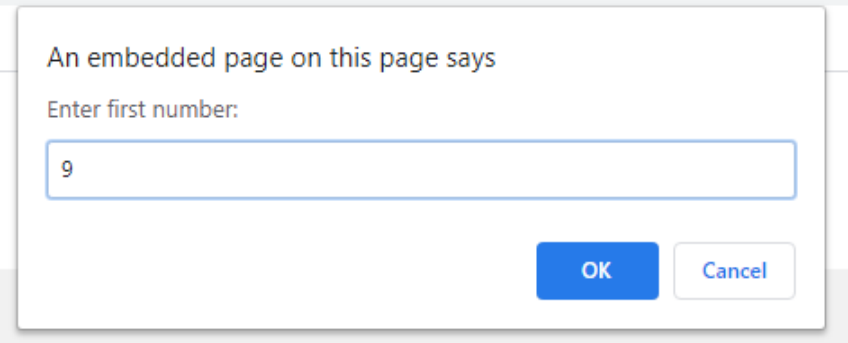
```
<html>
```

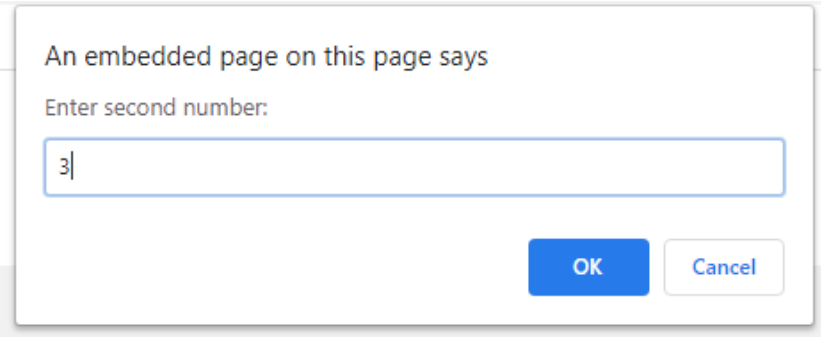
```
<body>
<script>
// program for a simple calculator
var result;

// take the operator input
var operator = prompt('Enter operator ( either +, -, * or / ): ');

// take the operand input
var number1 = parseFloat(prompt('Enter first number: '));
var number2 = parseFloat(prompt('Enter second number: '));

switch(operator)
{
   case '+':
        result = number1 + number2;
      document.write(`${number1} + ${number2} = ${result}`);
       break;

   case '-':
        result = number1 - number2;
      document.write(`${number1} - ${number2} = ${result}`);
       break;

   case '*':
        result = number1 * number2;
        document.write(`${number1} * ${number2} = ${result}`);
       break;

   case '/':
        result = number1 / number2;
        document.write(`${number1} / ${number2} = ${result}`);
       break;
```

```
        default:
            document.write('Invalid operator');
            break;
    }
    </script>
    </body>
    </html>
```

Output:







9 / 3 = 3

1.7 JavaScript Loop Statement

The JavaScript loops are used *to iterate the piece of code* using for, while, do while or for-in loops.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) for loop

- ✓ The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known.

Syntax:

```
for (initialization; condition; increment)
{
Code to be executed
}
```

Example:

```
<script>
for (i=0; i<=10; i=i+2)
{
document.write(i + "<br/>")
}
</script>
```

2) do while Loop

loop is a variant of the while loop.

This loop will execute the code block once.

before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

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

Example:

```
<script>
var i=21;
do{
document.write(i +"<br/>");
i++;
}while (i<=25);
</script>
```

3) while loop

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

Syntax:

```
while (condition)
{
Code to be executed
}
```

Example:

```
<script>
var i=11;
while (i<=20)
{
document.write(i + "<br/>");
```

```
i++;
}
</script>
```

4) for-in loop

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

The block of code inside the loop will be executed once for each property.

Syntax:

```
for (variable_name in object)
{
Code to be executed
}
```

Example:

```
<script type = "text/javaScript">
var lang = { first : "C", second : "Java",third : "Python", fourth : "PHP"};
        for (prog in lang)
   {
      document.write(lang[prog] + "<br >");
   }
</script>
```

Output:

```
C
Java
Python
PHP
```

Difference between While Loop and Do – While Loop

While Loop	**Do – While Loop**
In while loop, first it checks the condition and then executes the program.	In Do – While loop, first it executes the program and then checks the condition.
It is an entry – controlled loop.	It is an exit – controlled loop.
The condition will come before the body.	The condition will come after the body.
If the condition is false, then it terminates the loop.	It runs at least once, even though the conditional is false.
It is a counter-controlled loop.	It is a iterative control loop.

break statement

break statement breaks the loop and continues executing the code after the loop.

The break statement can also be used to jump out of a loop.

Example:

```
<script type = "text/javaScript">
var text = "";
var i;
for (i = 0; i < 10; i++)
{
  if (i === 4)
  { break;
  }
  text =text + "The number is " + i + "<br>";
}
document.write(text);
</script>
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
```

continue statement

Continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Example:

```
<script type = "text/javaScript">
var text = "";
var i;
for (i = 0; i < =6; i++)
{
  if (i === 4)
  {continue;
  }
  text =text + "The number is " + i + "<br>";
}
document.write(text);
</script>
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 5
The number is 6
```

1.8 Querying and Setting Properties

To obtain the value of a property, use . (dot) operator or square[ ] bracket.
The left-hand side should be an expression whose value is an object.
If using dot (.) operator, the right-hand must be a simple identifier that names the property.
If using square brackets, the value within the brackets must be an expression that evaluates to a string that contains the desired property name.

Example,

```
var name=author.lastname;    //get the "lastname " property of the book
 var title=book["main title"];     //get the "main title" property of the book
```

To create or set a property, use a dot or square brackets as you would to query the property, but put them on the left-hand side of an assignment expression:
Example,

```
book.price=250;    //create or set a property of price
book[“main title”]=“JavaScript”      //set the “main title” property
```

Deleting properties:

The delete operator deletes a property from an object.
The delete operator deletes both the value of the property and the property itself.
Syntax:

```
delete var_name.property;
```

Example, delete person.name; or
delete person[“name”];
Code:

```
<html>
<body>
<script>
var a={name:"Priti",age:35};
document.write(a.name+"   "+a.age+"<br>");
delete a.age;  //delete property
document.write(a.name+"   "+a.age);
</script>
</body>
</html>
```

Output:

```
Priti 35
Priti undefined
```

Property getter and setter

Also known as Javascript assessors.
Getters and setters allow you to control how important variables are accessed and updated in your code.
JavaScript can secure better data quality when using getters and setters.

Following example access fullName as a function: person.fullName().

```
<script>
// Create an object:
var person = {      firstName: "Chirag",
                    lastName : "Shetty",
                    fullName : function()
                     {
                     return this.firstName + " " + this.lastName;
                     }
                };
document.write(person.fullName());
</script>
```

Following example fullName as a property: person.fullName.

```
<script>
// Create an object:
var person = {   firstName: "Yash ",   lastName : "Desai",
                 get fullName()
                 {
                     return this.firstName + " " + this.lastName;
                 }
              };
// Display data from the object using a getter
document.write(person.fullName);
</script>
```

Code: **Getters and setters allow you to get and set properties via methods.**

```
<script>
var person = {
   firstName: 'Chirag',
   lastName: 'Shetty',
   get fullName()
   {
      return this.firstName + ' ' + this.lastName;
   },
   set fullName (name)
   {
      var words = name.split(' ');
      this.firstName = words[0];
      this.firstName = words[0].toUpperCase();
      this.lastName = words[1];
   }
}
document.write(person.fullName);     //Getters and setters allow you to get and set
properties via methods.
document.write("<br>"+"before using set fullname()"+"<br>");
person.fullName = 'Yash Desai';      //Set a property using set
document.writeln(person.firstName);     // Yash
document.write(person.lastName);       // Desai
</script>
```

Output:

Chirag Shetty
before using set fullname()
YASH Desai