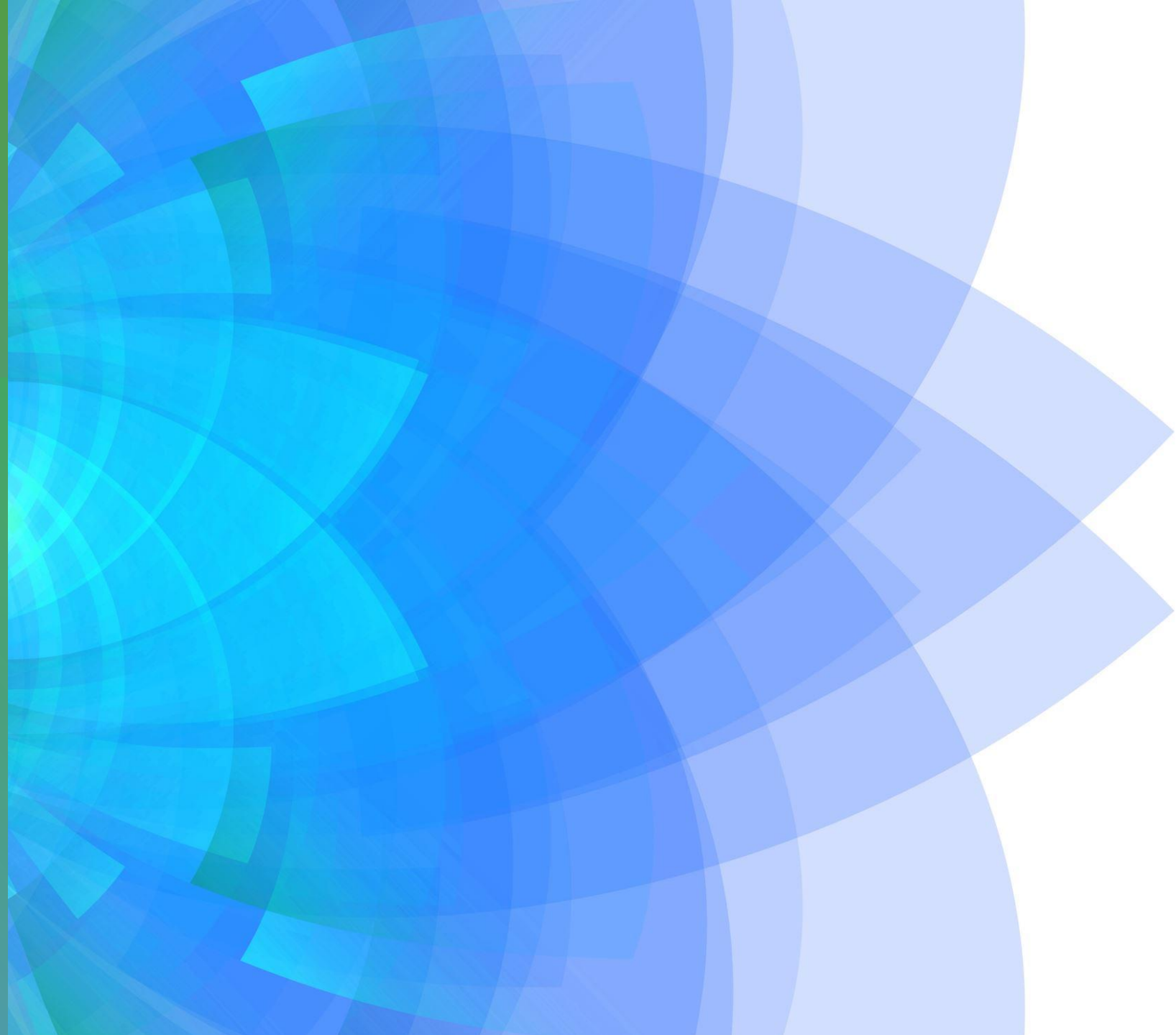
UNIT 3 (JAVASCRIPT)

WEB DESIGNING



What is JavaScript

- *JavaScript* was initially created to “make web pages alive”.
- The programs in this language are called *scripts*. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.
- JavaScript is *an object-based scripting language* which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language.
- The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- JavaScript is a very powerful **client-side scripting language**. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and mobile application development.

- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. JavaScript was developed by **Brendan Eich in 1995**, which appeared in Netscape, a popular browser of that time. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

- The language was initially called LiveScript and was later renamed JavaScript.

- Being a scripting language, **JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code**. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. The main advantage of JavaScript is that **all modern web browsers support** JavaScript. So, you do not have to worry about whether your site visitor uses Internet Explorer, Google Chrome, Firefox or any other browser. JavaScript will be supported. Also, JavaScript **runs on any operating system** including Windows, Linux or Mac.

When most people get interested in web development, they start with good old HTML and CSS. From there, they move on to JavaScript, which makes sense, because , these three elements together form the backbone of web development.
•HTML is the structure of your page like the headers, the body text, any images you want to include. It basically defines the contents of a web page.
•CSS controls how that page looks (it's what you'll use to customize fonts, background colors, etc.).
•JavaScript is the third element. Once you've created your structure (HTML) and your aesthetic vibe (CSS), JavaScript makes your site dynamic (automatically updateable).

What is JavaScript Used For?

•**Web Applications**: JavaScript is used for adding interactivity and automation to websites.
•**Mobile Applications**: JavaScript is also used for developing applications for phones and tablets. With frameworks like React Native, you can develop full-fledged mobile applications with all those fancy animations.
•**Web-based Games**: If you've ever played a game directly on the web browser, JavaScript was probably used to make that happen.
•**Back-end Web Development**: JavaScript has traditionally been used for developing the front-end parts of a web application. However, with the introduction of NodeJS, a prevalent back-end JavaScript framework, things have changed. And now, JavaScript is used for developing the back-end structure also.
•**Smartwatch Apps, art, presentations… etc.**

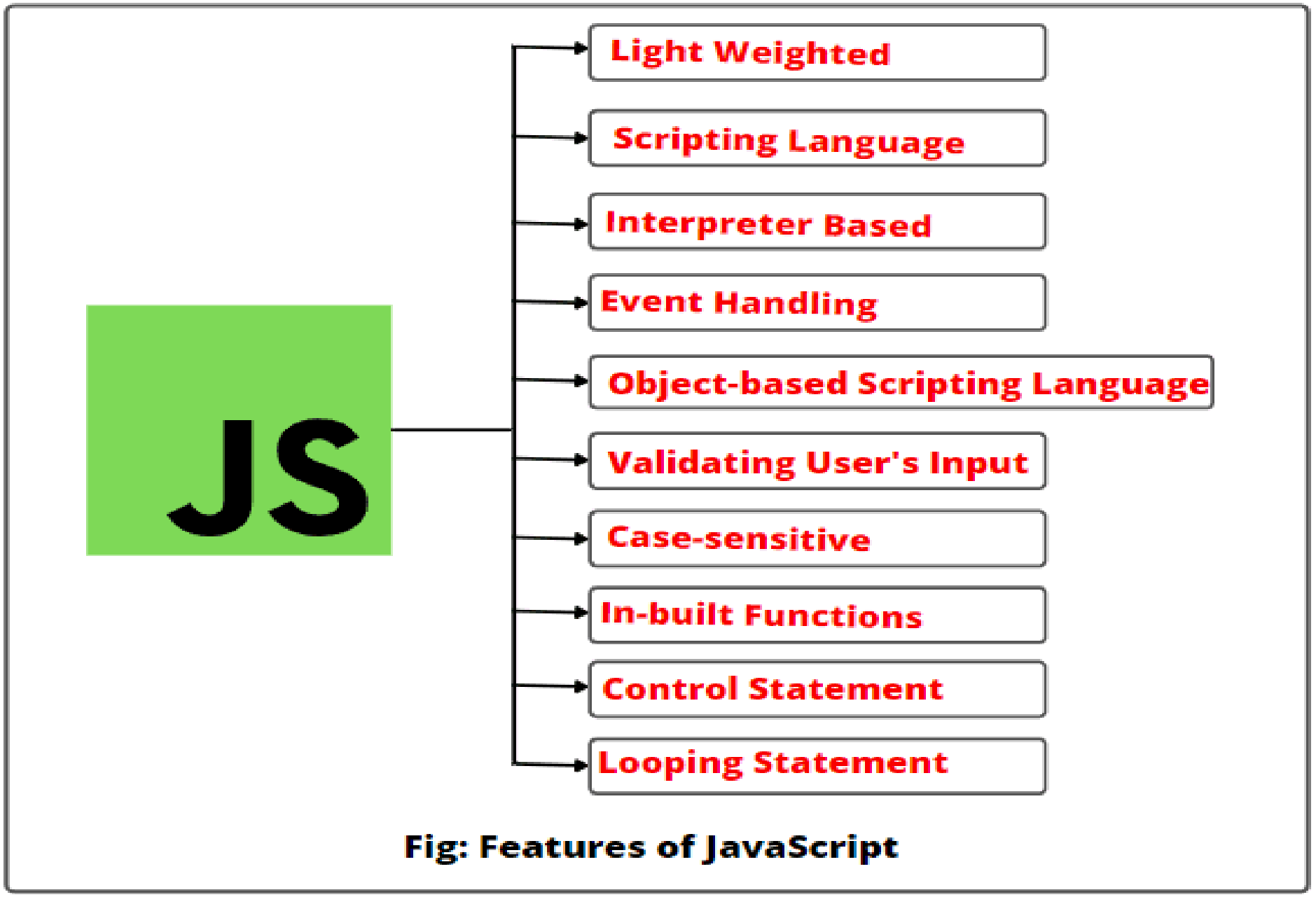


Fig: Features of JavaScript

JavaScript advantages

- **Fast speed:** JavaScript is executed on the client side that's why it is very fast.
- **Easy to learn:** JavaScript is easy to learn. Any one which have basic knowledge of programming can easily lean JavaScript.
- **Versatility:** It refers to lots of skills. It can be used in a wide range of applications.
- **Browser Compatible:** JavaScript supports all modern browsers. It can execute on any browser and produce same result.
- **Server Load:** JavaScript reduce the server load as it executes on the client side.
- **Rich interfaces:** JavaScript provides the drag and drop functionalities which can provides the rich look to the web pages.
- **Popularity:** JavaScript is a very popular web language because it is used every where on the web.
- **Regular Updates:** JavaScript updated annually by ECMA.
- **Interoperability** – Because JavaScript seamlessly integrates with other programming languages, many developers favour using it to create a variety of applications. Any webpage or the script of another programming language can contain it.

JavaScript disadvantages

- **Code Visibility:** JavaScript code is visible to every one and this is the biggest disadvantage of JavaScript.
- **Stop Render:** One error in JavaScript code can stop whole website to render.
- **No Multiple Inheritance:** JavaScript only support single inheritance.
- **Cannot Debug** – Although some HTML editors allow for debugging, they are not as effective as editors for C or C++. Additionally, the developer has a difficult time figuring out the issue because the browser doesn't display any errors.
- **Client-side Security** – The user can see the JavaScript code; it could be misused by others. These actions might involve using the source code anonymously. Additionally, it is very simple to insert code into the website that impair the security of data transmitted via the website.
- **Browser Support** – Depending on the browser, JavaScript is interpreted differently. Therefore, before publication, the code needs to run on various platforms. We also need to check the older browsers because some new functions are not supported by them.

A Simple JavaScript Program

You should place all your JavaScript code within **<script> tags** (<script> and </script>) if you are keeping your JavaScript code within the HTML document itself. This helps your browser distinguish your JavaScript code from the rest of the code.

Syntax:

```
<script>
  // JavaScript Code
</script>
```

Example:

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

WRITING JAVASCRIPT IN HTML

How to add JavaScript to html

JavaScript programs cannot get executed without the help of HTML or without integrated into HTML code.
Javascript is used in several ways in web pages such as generate warning messages, build image galleries, DOM manipulation, form validation, and more.
There are following three ways in which users can add JavaScript to HTML pages.

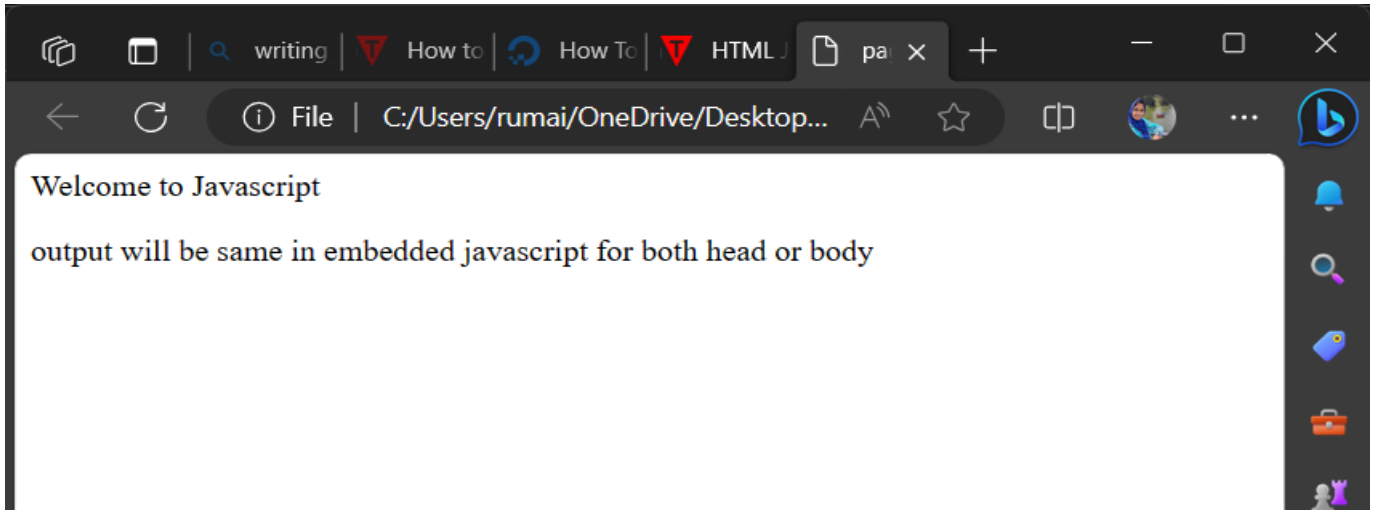
1.Embedding code
2.Inline code
3.External file

1. Embedding code:-

To add the JavaScript code into the HTML pages, we can use the <script>.....</script> tag of the HTML that wrap around JavaScript code inside the HTML program. Users can also define JavaScript code in the <body> tag (or we can say body section) or <head> tag because it completely depends on the structure of the web page that the users use.

```
<!DOCTYPE html >
<html>
<head>
<title> page title</title>
<script>
document.write("Welcome to Javascript");
</script>
</head>
<body>
<p>output will be same in embedded javascript for both head or body </p>
</body>
</html>
```

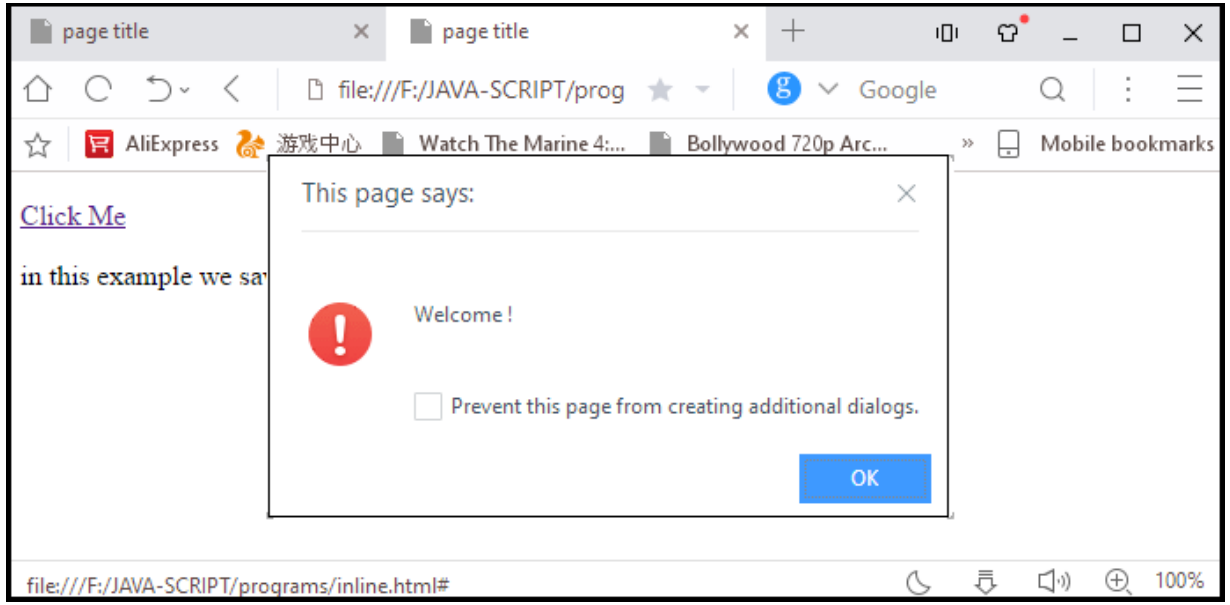
OR




```
<!DOCTYPE html >
<html>
<head>
<title> page title</title>
</head>
<body>
<script>
document.write("Welcome to Javascript");
</script>
<p>output will be same in embedded javascript for both head or body </p>
</body>
</html>
```

2.Inline code:-

Generally, this method is used when we have to call a function in the HTML event attributes. There are many cases (or events) in which we have to add JavaScript code directly eg., OnMover event,OnClick etc.
Let's see with the help of an example, how we can add JavaScript directly in the html without using the <script>.... </script> tag.




```
<!DOCTYPE html >
<html>
<head>
<title> page title</title>
</head>
<body>
<p>
<a href="#" onClick="alert('Welcome !');">Click Me</a>
</p>
<p> in this example we saw how to use inline JavaScript or directly in an HTML tag. </p>
</body>
</html>
```

3.External file:-

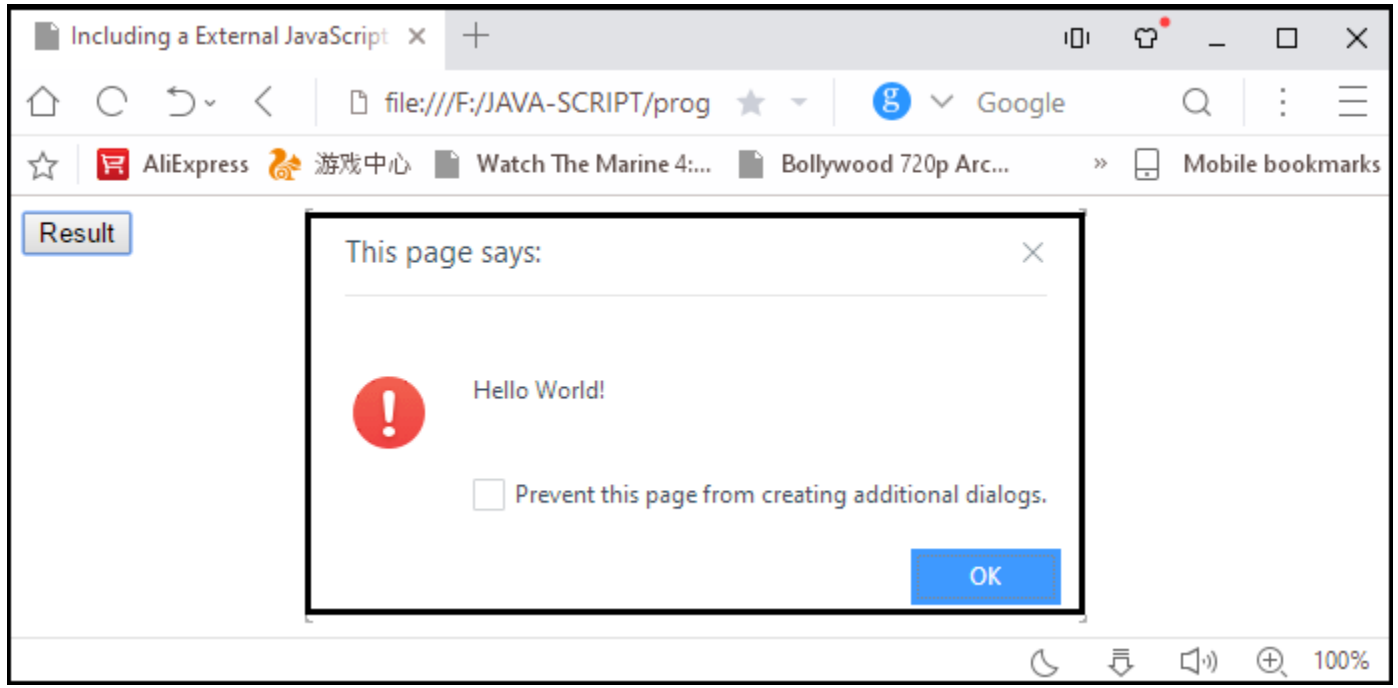
We can also create a separate file to hold the code of JavaScript with the (.js) extension and later incorporate/include it into our HTML document using the **src** attribute of the <script> tag. It becomes very helpful if we want to use the same code in multiple HTML documents. It also saves us from the task of writing the same code over and over again and makes it easier to maintain web pages.

```
<html>
<head>
<meta charset="utf-8">
<title>Including a External JavaScript File</title>
</head>
<body>
<form>
<input type="button" value="Result" onclick="display()"/>
</form>
<script src="hello.js">
</script>
</body>
</html>
```

Now let's create separate JavaScript file

Hello.js

```
function display()
{
alert("Hello World!");
}
```



The HTML noscript Element

The <noscript> element provides us an alternate way to create content for the users that either have browsers that don't support the JavaScript or have disabled JavaScript in the browser.

This element can contain any HTML element other than the <script> tag that can be included in the <HTML> element.

```
<!DOCTYPE html>
<html>
<body>
<h1>The noscript element</h1>
<p>If the user have a browser with JavaScript disabled will show the text inside the noscript element and "Hello World!" will not be displayed.</p>
<script>
document.write("Hello World!")
</script>
<noscript>Sorry, your browser may not support JavaScript! orJavaScript is disabled in your br owser </noscript>
</body>
</html>
```

Javascript Operators

JavaScript operators are symbols that are used to perform operations on operands.An operator performs some operation on single or multiple operands (data value) and produces a result. For example, in 1 + 2, the + sign is an operator and 1 is left side operand and 2 is right side operand. The + operator performs the addition of two numeric values and returns a result.

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**.

JavaScript includes following categories of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical (or relational) Operators
4. Assignment Operators
5. Conditional operators
6. Bitwise Operators
7. Typeof Operator

1.Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operands.

The following example demonstrates how arithmetic operators perform different tasks on operands.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Example: Arithmetic Operation

```
let x = 5, y = 10;
let z = x + y; //performs addition and returns 15
z = y - x; //performs subtraction and returns 5
z = x * y; //performs multiplication and returns 50
z = y / x; //performs division and returns 2
z = x % 2; //returns division remainder 1
x++; //post-increment, x will be 5 here and 6 in the next line
++x; //pre-increment, x will be 7 here
x--; //post-decrement, x will be 7 here and 6 in the next line
--x; //pre-decrement, x will be 5 here
```

The ++ and -- operators are unary operators. It works with either left or right operand only. When used with the left operand, e.g., x++, it will increase the value of x when the program control goes to the next statement. In the same way, when it is used with the right operand, e.g., ++x, it will increase the value of x there only. Therefore, x++ is called post-increment, and ++x is called pre-increment.

String Concatenation

The + operator performs concatenation operation when one of the operands is of string type. The following example demonstrates string concatenation even if one of the operands is a string.

```
let a = 5, b = "Hello ", c = "World!", d = 10;
a + b; //returns "5Hello "
b + c; //returns "Hello World!"
a + d; //returns 15
b + true; //returns "Hello true"
c - b; //returns NaN; - operator can only used with numbers
```

2.Comparison Operators

JavaScript provides comparison operators that compare two operands and return a boolean value `true` or `false`.

Operators	Description
==	Compares the equality of two operands without considering type.
===	Compares equality of two operands with type.
!=	Compares inequality of two operands.
>	Returns a boolean value true if the left-side value is greater than the right-side value; otherwise, returns false.
<	Returns a boolean value true if the left-side value is less than the right-side value; otherwise, returns false.
>=	Returns a boolean value true if the left-side value is greater than or equal to the right-side value; otherwise, returns false.
<=	Returns a boolean value true if the left-side value is less than or equal to the right-side value; otherwise, returns false.

Example: JavaScript Comparison Operators

```
let a = 5, b = 10, c = "5";
let x = a; a == c; // returns true
a === c; // returns false
a == x; // returns true
a != b; // returns true
a > b; // returns false
a < b; // returns true
a >= b; // returns false
a <= b; // returns true
```

3. Logical Operators

In JavaScript, the logical operators are used to combine two or more conditions. JavaScript provides the following logical operators.

Operator	Description
&&	&& is known as AND operator. It checks whether two operands are non-zero or not (0, false, undefined, null or "" are considered as zero). It returns 1 if they are non-zero; otherwise, returns 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero or not (0, false, undefined, null or "" is considered as zero). It returns 1 if any one of of them is non-zero; otherwise, returns 0.
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition). !false returns true, and !true returns false.

Example: Logical Operators

```
let a = 5, b = 10; (a != b) && (a < b); // returns true
(a > b) || (a == b); // returns false
(a < b) || (a == b); // returns true
!(a < b); // returns false
!(a > b); // returns true
```

4. Assignment Operators

JavaScript provides the assignment operators to assign values to variables with less key strokes.

Assignment operators	Description
=	Assigns right operand value to the left operand.
+=	Sums up left and right operand values and assigns the result to the left operand.
-=	Subtract right operand value from the left operand value and assigns the result to the left operand.
*=	Multiply left and right operand values and assigns the result to the left operand.
/=	Divide left operand value by right operand value and assign the result to the left operand.
%=	Get the modulus of left operand divide by right operand and assign resulted modulus to the left operand.

Example: Assignment operators

```
let x = 5, y = 10, z = 15;
x = y; //x would be 10
x += 1; //x would be 6
x -= 1; //x would be 4
x *= 5; //x would be 25
x /= 5; //x would be 1
x %= 2; //x would be 1
```

5. Conditional Operators

JavaScript **Conditional Operators** allow us to perform different types of actions according to different conditions. We make use of the 'if' statement.

```
if(expression){
        do this;
        }
```

The above argument named 'expression' is basically a condition that we pass into the 'if' and if it returns 'true' then the code block inside it will be executed otherwise not.

- ```
  // if example
  let age = 20;
  if(age == 20){
  console.log('Hola!'); // Hola! Will be the output
  }
  ```
- ```
  if(0){
      console.log('hey'); // Will not be printed
       }
  ```

- ```
  if(1){
   console.log('Yo')// Will print Yo
    }
  ```
  ```

- ```
  if(this is true){
          do this;
              }
  else{
          do this;
     }
  ```

- ```
  // else example
  let age = 21;
  if(age == 20){
      console.log('You are 20'); // Not executed
  }else{
      console.log('Adios!'); // Will be alerted
  }
  ```

- ```
  if(expression){
  do this;
  }
  else if(expression){
  do this;
  }
  else{
  do this;
  }
  ```

Eg:

```
// The else-if example
let age = 22;
if(age < 18 ){
    console.log('Too Young.');
}else if( age > 18 && age < 60) {
    console.log(‘hello'); // hello will be printed
}else {
    console.log(‘too old');
}
```
```

Ternary Operator: In Javascript we also have a ternary operator which is a very short way of performing an action on based of a condition.

```
let result = condition ? value1 : value2;
```

It works similarly to an if-else, where based on a condition we evaluate on the result. In the above code snippet if the 'condition' evolves to 'true' then 'value1' will be executed otherwise 'value2' will be executed.
Eg:

```
// Ternary Operator Example
let age = 20;
let result = age>18 ? 'Great' : 'Not so great';
console.log(result); // Great
```

Example: Ternary operator

```
let a = 10, b = 5;
let c = a > b? a : b; // value of c would be 10
let d = a > b? b : a; // value of d would be 5
```

6.Bitwise Operators

The bitwise operators perform bitwise operations on operands.
The bitwise operators are as follows:

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

7.typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1.Primitive data type
2.Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. 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:

```
1.var a=40;//holding number
2.var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called **Literals**.

Variable values are called **Variables**.

JavaScript Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals
2. **Strings** are text, written within double or single quotes

variables

One of the basic constructs of JavaScript is the variable. Variables are used to store and manipulate data. They are declared using the keyword “var,” “let,” or “const.” The difference between “var,” “let,” and “const” is the scope and the ability to reassign the value. “Var” is function scope (they are only accessible within the function in which they are declared), “let” is block scope (they are only accessible within the block of code in which they are declared), and “const” is block scope and cannot be reassigned (if you try to reassign a value to a variable declared with “const,” you will get an error).

```
// How to create variables:
var x;
let y;
// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

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

```
let x;
x = 6;
```

JavaScript Expressions

An **expression** is a block of code that evaluates to a value. A **statement** is any block of code that is performing some action.

JavaScript's expression is a valid set of literals, variables, operators, and expressions that evaluate a single value that is an expression. This single value can be a number, a string, or a logical value depending on the expression.

An *expression* is any valid unit of code that resolves to a value.

Conceptually speaking, there are **two kinds of expressions**: **those that perform some sort of assignment** and **those that evaluate to a value.**

For example,x=10 is an expression that performs an assignment. This expression itself evaluates to 1010. Such expressions make use of the *assignment operator*.

On the flip side, the expression ,10+5 simply evaluates to 1919. These expressions make use of simple *operators*.

Eg:

Result=x+y/z

Result=2+4/2

JavaScript has the following expression categories:

- **Arithmetic**: evaluates to a number, for example 3.14159.
- **String**: evaluates to a character string, for example, "Fred" or "234".
- **Logical**: evaluates to true or false.
- **Primary expressions**: Basic keywords and general expressions in JavaScript.
- **Left-hand-side expressions**: Left values are the destination of an assignment.

Primary expressions

Primary expressions consist of basic keywords in JavaScript.

this

this is used to refer to the current object; it usually refers to the method or object that calls it.

this is used either with the dot operator or the bracket operator.

this['element']

this.element

Grouping operator

The grouping operator () is used to determine the evaluation precedence of expressions. For example, the two expressions below evaluate to different results because the order of operations is different in both of them.

Eg: a * b - c

a * (b - c)

a * b - c applies the multiplication operator first and *then* evaluates the result of the multiplication with -c. While a * (b - c) evaluates the brackets first.

Left-hand side expressions

new

new creates an instance of the object specified by the user and has the following prototype.

var objectName = new objectType([param1, param2, ..., paramN]);

super

super calls on the current object's parent and is useful in classes to call the parent object's constructor.

super([arguments]); *// parent constructor*

super.method(args...) *// parent's method*

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.
There are 3 ways to construct array in JavaScript

1.By array literal
2.By creating instance of Array directly (using new keyword)
3.By using an Array constructor (using new keyword)

1) JavaScript array literal

The syntax of creating array using array literal is given below:

```
var arrayname=[value1,value2.....valueN];
```

As you can see, values are contained inside [ ] and separated by , (comma).

Eg:

```
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
```

The .length property returns the length of an array.

Output of the above example:

Sonoo
Vimal
Ratan

2) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
var arrayname=new Array();
```

Here, **new keyword** is used to create instance of array.

Eg:

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

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

Output of the above example:

Arun
Varun
John

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

Eg:

```
 <script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Output of the above example:

Jai
Vijay
Smith

JavaScript Array Methods

Let's see the list of JavaScript array methods with their description.

	Description
concat()	It returns a new array object that contains two or more merged arrays.
copywithin()	It copies the part of the given array with its own elements and returns the modified array.
entries()	It creates an iterator object and a loop that iterates over each key/value pair.
every()	It determines whether all the elements of an array are satisfying the provided function conditions.
fill()	It fills elements into an array with static values.
from()	It creates a new array carrying the exact copy of another array element.
find()	It returns the value of the first element in the given array that satisfies the specified condition.
findIndex()	It returns the index value of the first element in the given array that satisfies the specified condition.
forEach()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.
isArray()	It tests if the passed value ia an array.
join()	It joins the elements of an array as a string.

lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
map()	It calls the specified function for every array element and returns the new array
of()	It creates a new array from a variable number of arguments, holding any type of argument.
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
some()	It determines if any element of the array passes the test of the implemented function.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.
sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
toString()	It converts the elements of a specified array into string form, without affecting the original array.
unshift()	It adds one or more elements in the beginning of the given array.
values()	It creates a new iterator object carrying values for each index in the array.

JAVASCRIPT PROGRAMMING CONSTRUCTS

Most of the programming languages have a common set of programming constructs. JavaScript also provides a complete range of basic programming constructs.

JavaScript Conditional Statements

As in any other programming language conditional statements in JavaScript are used to perform different actions based on different conditions. While writing a code you may want to perform different actions for different decisions. For this JavaScript has following conditional statements:

1. **If statement** - use this statement if you want to execute some code only if a specified condition is true.

   Syntax

   ```
   if (condition)
   {
   /* code to be executed if condition is true*/
   }
   ```

Eg:

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```

OUTPUT

value of a is greater than 10

2. **if...else statement**:- This statement is used if you want to execute some code if the condition is true and any another code if the condition is false.

Syntax:

```
if (condition)
{
// code executed if condition is true
}
else
{
// code executed if condition is not true
}
```

Eg:

```
let a = [2,3,"hello"]
//OUTPUT prints false because 2 is not greater than 3
if (a[0] > a[1]) {
 console.log("true")
}
else
{
console.log("false")
}
```

3. **if...else if....else statement** - use this statement if you want to select one of many blocks of code to be executed.

Syntax:

```
if (condition1)
{
//code to be executed if condition1 is true
}
else if (condition2)
{
//code to be executed if condition2 is true
}
else
{
//code to be executed if condition 1 and 2 are not true
}
```

Eg:

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20"); //this will be the output
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```

5. **Switch statement** – You can use Switch statement to select one of many blocks of code that is to be executed. With the Switch statement you can select from a number of alternatives.

Syntax:

```
switch(value)
{
Case 1:
//This code executed if first alternative is chosed
break;
Case 2;
//This code executed if second alternative is chosed
break;
default://used to take a default action
}
```

Note: You need to use break to come out of the Switch statement once selected case code is executed

Eg:

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";  //this will be the output
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
```

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

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

1) JavaScript 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. The syntax of for loop is given below.

syntax

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

Eg:

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

OUTPUT

1
2
3
4
5

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

syntax

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

Eg:

```
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
i++;
}
</script>
```

Output

11
12
13
14
15

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

syntax

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

Eg:

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

OUTPUT

```
21
22
23
24
25
```

4) For-in loop

The syntax of the **for...in** loop is:

```
for (key in object) {
// body of for...in
}
```

In each iteration of the loop, a key is assigned to the key variable. The loop continues for all object properties.

Eg1:

```
const student = { name: 'Monica', class: 7, age: 12 }
 // using for...in
for ( let key in student ) {
// display the properties
console.log(`${key} => ${student[key]}`);
}
```

OUTPUT

```
name => Monica
class => 7
age => 12
```

In the above program, the for...in loop is used to iterate over the student object and print all its properties.

- The object key is assigned to the variable key.
- student[key] is used to access the value of key.

Eg2:

```
const string = 'code';
 // using for...in loop
 for (let i in string) {
console.log(string[i]);
 }
```

OUTPUT

```
 C
 o
 d
 e
```

Eg3:

```
// define array
const arr = [ 'hello', 1, 'JavaScript' ];
// using for...in loop
for (let x in arr) {
 console.log(arr[x]);
 }
```

OUTPUT

```
hello
1
JavaScript
```

JavaScript Dialogue Boxes

Dialogue boxes are a kind of popup notification, this kind of informative functionality is used to show success, failure, or any particular/important notification to the user.

JavaScript uses 3 kinds of dialog boxes:

- **Alert**
- **Prompt**
- **Confirm**

These dialog boxes can be of very much help in making our website look more attractive.

Alert Box: An alert box is used on the website to show a warning message to the user that they have entered the wrong value other than what is required to fill in that position. Nonetheless, an alert box can still be used for friendlier messages. The alert box gives only one button “OK” to select and proceed.

Eg:

```
<script type="text/javascript">
        function Warning() {
                alert ("Warning danger you have not filled everything");
                console.log ("Warning danger you have not filled everything");

        }
</script>
<p>Click the button to check the Alert Box functionality</p>
<form>
        <input type="button" value="Click Me" onclick="Warning();" />
</form>
```

Confirm box: A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either “OK” or “Cancel” to proceed. If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false and will show null.

Example:

```
<script type="text/javascript">
        function Confirmation() {
                var Val = confirm("Do you want to continue ?");
                if (Val == true) {
                        console.log(" CONTINUED!");
                        return true;
                } else {
                        console.log("NOT CONTINUED!");
                        return false;
                }
        }
</script>
<p>Click the button to check the Confirm Box functionality</p>
<form>
        <input type="button" value="Click Me" onclick="Confirmation();" />
</form>
```

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

Example:

```
<script type="text/javascript">
        function Value(){
        var Val = prompt("Enter your name : ", "Please enter your name");
        console.log("You entered : " + Val);
        }

</script>

<p>Click the button to check the Prompt Box functionality</p>
<form>
        <input type="button" value="Click Me" onclick="Value();" />
</form>
```

JavaScript Functions

JavaScript functions also called methods are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1.Code reusability: We can call a function several times so it save coding.

2.Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

There are two types of functions:

1. Built-in functions
2. User defined functions

Built in function

A ***built-in function in JavaScript*** is a function that's already available for use without needing any extra code. These functions are a part of JavaScript itself and are designed to do common tasks like working with text, doing math calculations, and handling lists of data.

String Functions:

- **charAt(index)**: Returns the character at the specified index in a string.
- **concat(str1, str2)**: Combines two or more strings and returns a new string.
- **toUpperCase()**: Converts a string to uppercase.
- **toLowerCase()**: Converts a string to lowercase.
- **split(separator)**: Splits a string into an array of substrings based on a specified separator.
- **indexOf(searchValue)**: Returns the index of the first occurrence of a substring.
- **replace(searchValue, replaceValue)**: Replaces a substring with another substring.
- **substring(start, end)**: Extracts a portion of a string.
- **trim()**: Removes whitespace from the beginning and end of a string.

Array Functions:

- **push(element)**: Adds an element to the end of an array.
- **pop()**: Removes the last element from an array.
- **shift()**: Removes the first element from an array.
- **unshift(element)**: Adds an element to the beginning of an array.
- **concat(arr1, arr2)**: Combines two or more arrays and returns a new array.
- **slice(start, end)**: Returns a shallow copy of a portion of an array.
- **splice(start, deleteCount, element1, element2, ...)**: Changes the contents of an array by removing, replacing, or adding elements.

Math Functions:

- **Math.random()**: Generates a random floating-point number between 0 and 1.
- **Math.floor(num)**: Rounds a number down to the nearest integer.
- **Math.ceil(num)**: Rounds a number up to the nearest integer.
- **Math.round(num)**: Rounds a number to the nearest integer.

Date Functions:

- **new Date()**: Creates a new Date object representing the current date and time.
- **getDate()**, **getMonth()**, **getYear()**, **getHours()**, **getMinutes()**, **getSeconds()**: Get various components of a date.
- **setDate()**, **setMonth()**, **setYear()**, **setHours()**, **setMinutes()**, **setSeconds()**: Set various components of a date.

Regular Expression Functions:

- **test(str)**: Tests if a regular expression pattern matches a string.
- **exec(str)**: Searches for a match in a string and returns the matched text.

Global Functions:

- **parseInt(str)**: Parses a string and returns an integer.
- **parseFloat(str)**: Parses a string and returns a floating-point number.
- **isNaN(value)**: Checks if a value is NaN (Not-a-Number).
- **typeof(value)**: Returns a string that represents the type of a value.

USER DEFINED FUNCTIONS

The syntax of declaring function is given below.

SYNTAX:

```
function functionName(Parameter1, Parameter2, ...)
   {
     // Function body
   }
```

To create a function in JavaScript, we have to first use the keyword *function*, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces {} is the body of the function.

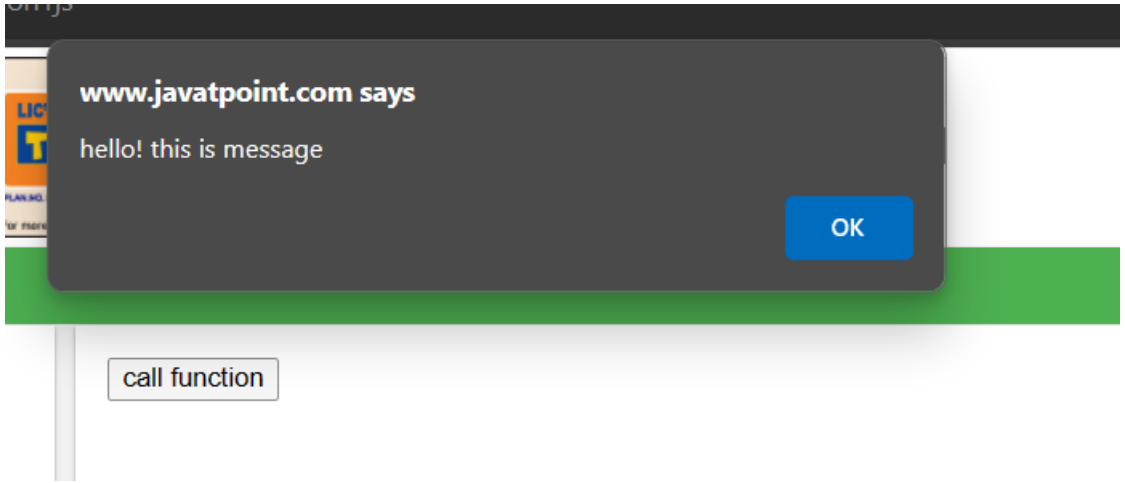
In javascript, functions can be used in the same way as variables for assignments, or calculations.

Function Definition: Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript. A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

•Every function should begin with the keyword *function* followed by,
•A user-defined function name that should be unique,
•A list of parameters enclosed within parentheses and separated by commas,
•A list of statements composing the body of the function enclosed within curly braces {}.

Eg1:

```
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
```



JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has argument.

Eg2:

```
function calcAddition(number1, number2) {
            return number1 + number2;
}
console.log(calcAddition(6,9)); //OUTPUT WILL BE 15
```

Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

Eg:

```
<script>
function getInfo(){
return "hello javatpoint! How r u?";
}
</script>
<script>
document.write(getInfo());
</script>
```

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax:

```
new Function ([arg1[, arg2[, ....argn]],] functionBody)
```

Eg:

```
<script>
var add=new Function("num1","num2","return num1+num2");
document.writeln(add(2,5));   //output will be 7
</script>
```

Scope of Variables in Javascript

Scope of variables refers to the accessibility of a particular variable within the program.
For example, assume you have two different functions. First, you declare a variable in function 1. Then, you move on to the following function, i.e., function 2. Is it possible if you try to access the variable made in function 1 from function 2? This refers to the Scope of a Variable in JavaScript.

```
1  function One(){
2  const message = "Welcome to Simplilearn"; //Defining a variable
3  }
4
5  function two(){
6      console.log(message);              // Accessing variable "message" from another function
7  }
8
9  two();                                 // Calling function two
10
11
```

```
Uncaught ReferenceError: message is not defined                    Article.js:7
    at two (Article.js:7)
    at Article.js:10
>
```