Introduction JavaScript

Script means small piece of code. Scripting languages are two kinds one is client-side other one is servers-side scripting. In general client-side scripting is used for verifying simple validation at client side; server-side scripting is used for database verifications. VBScript, java script and J script are examples for client-side scripting and ASP, JSP, Servlets, PHP etc. are examples of server-side scripting.

JavaScript is a scripting language that runs inside the browser to manipulate and enhance the contents of Web pages. JavaScript is an object-based scripting language which is lightweight and cross-platform. JavaScript was developed by **Brendan Eich** in 1995

The language was initially called LiveScript and was later renamed JavaScript. There are many programmers who think that JavaScript and Java are the same. In fact, JavaScript and Java are very much unrelated. Java is a very complex programming language whereas JavaScript is only a scripting language.

Java Script is designed to add interactivity to HTML pages. Web pages are two types

1. Static web page
2. Dynamic webpage

- Static web page where there is no specific interaction with the client

  Example: HTML

- Dynamic web page which is having interactions with client and as well as validations can be added.

  Example: to add script to HTML page it is called dynamic page. Netscape navigator developed java script. Microsoft's version of JavaScript is Jscript.

It has no relation to Java at all.

- Java script code as written between <script> -------- </script>tags
- All java script statements end with a semicolon
- Java script ignores white space
- Java script is case sensitive language
- Script program can save as either. Js or. html

Benefits of JavaScript

- It is widely supported by web browsers;
- It gives easy access to the document objects and can manipulate most of them.
- Java Script gives interesting animations with long download times associated with many multimedia data types;
- Web surfers don't need a special plug-in to use your scripts
- Java Script relatively secure

Problems with JavaScript

- Most scripts relay upon manipulating the elements of DOM;
- Your script does not work then your page is useless
- Because of the problems of broken scripts many web surfers disable java script support in their browsers
- Script can run slowly and complex scripts can take long time to start up

Disadvantages of JavaScript:

- Client-Side Security. Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable JavaScript.
- Browser Support. JavaScript is sometimes interpreted differently by different browsers. Whereas server-side scripts will always produce the same output, client-side scripts can be a little unpredictable. Don't be overly concerned by this though - as long as you test your script in all the major browsers you should be safe. Also, there are services out there that will allow you to test your code automatically on check in of an update to make sure all browsers support your code.

The syntax of the script tag isas follows:

```
<script language="Javascript" type="text/javascript">
        Java Script Code
</script>
```

The language attribute specifies what scripting language you are using. Typically, its value will be javascript.

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

JavaScript is placed in three places in HTML Code

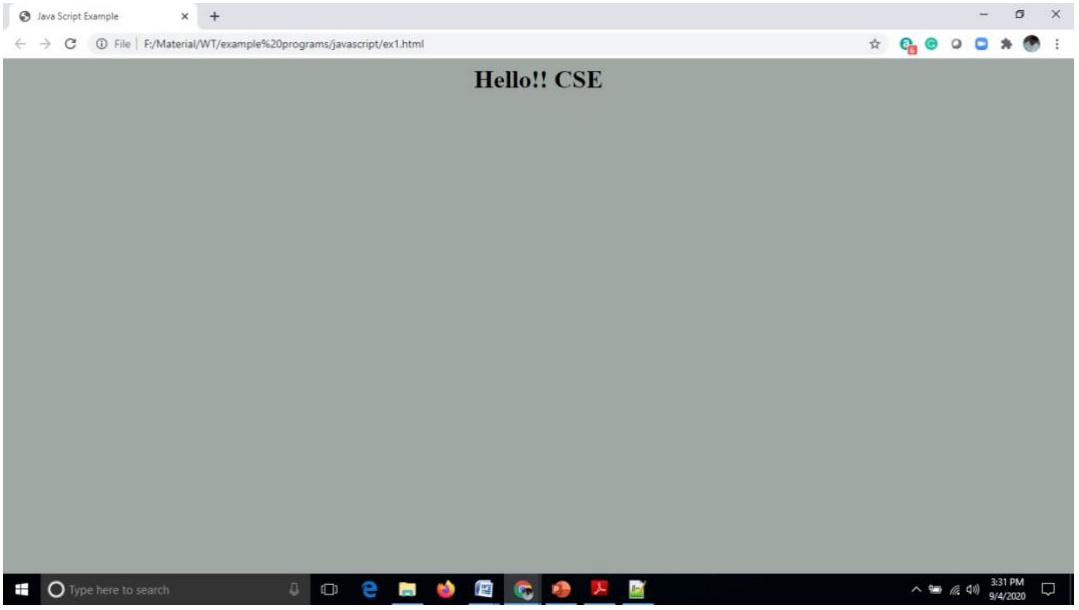
1. within body tag,
2. within head tag and
3. external JavaScript file.

Ex: write a Simple Javascript Program **(With in Head Tag)**

```
<html>
  <head>
      <title> Java Script Example</title>
      <center>
          <script type="text/javascript">
              document.write("<h1>"+"Hello!! CSE"+"</h1>");
          </script>
      </center>
```

```
</head>
<body bgcolor="#9fa8a3">
</body>
</html>
```

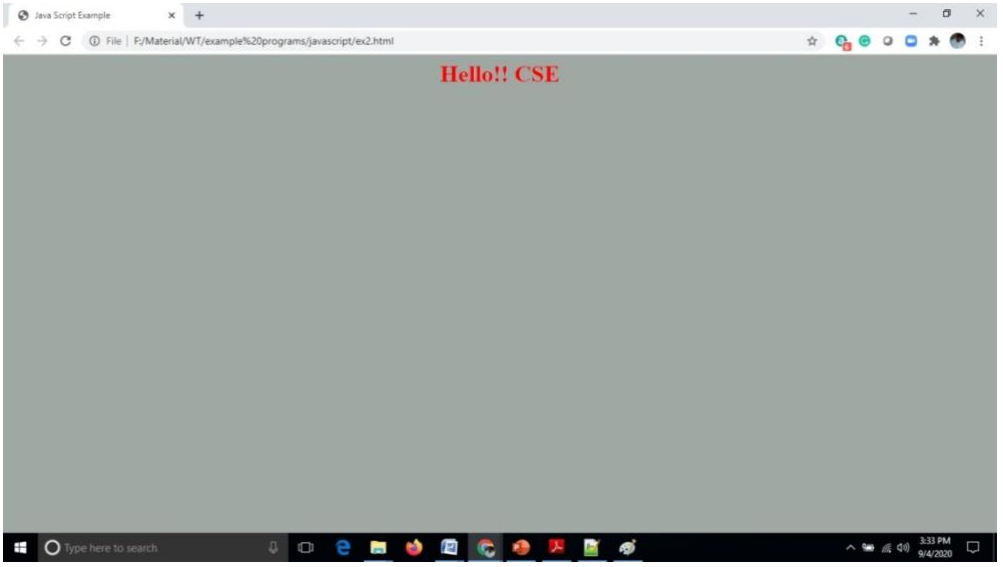
Output:



Example-2: Ex: write a Simple Javascript Program **(With in Body Tag)**

```
<html>
    <head>
        <title> Java Script Example</title>
    </head>
    <body bgcolor="#9fa8a3">
    <center>
        <script type="text/javascript">
            document.write("<h1><font color='red'>"+"Hello!!
        CSE"+"</font></h1>");
        </script>
    </center>
    </body>
</html>
```

Output:



In above example we write use <script> tag for defining JavaScript code and value of type attribute is **text/javascript, it** is the content type that provides information to the browser about the data. The **document.write()** function is used to display dynamic content through JavaScript.

External JavaScript file

We can create external JavaScript file and embed it in many html page. JavaScript code write in any text editor and it is saved with ".js" extension and it is embedded in HTML code with the help of 'src' attribute of <script> tag which is placed in header part

Example:

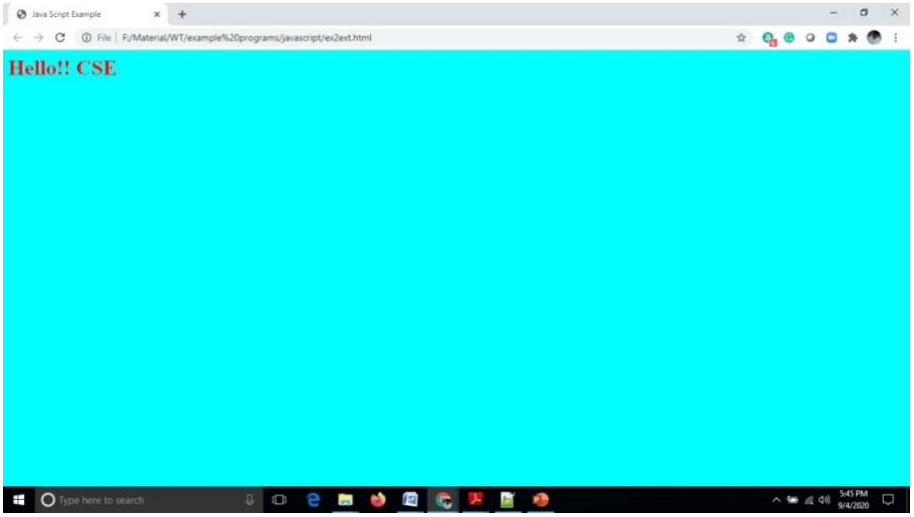
ext.js

```
document.write("<h1><font color='red'>"+"Hello!! CSE"+"</font></h1>");
```

Example.html

```
<html>
    <head>
        <title> Java Script Example</title>
        <script type="text/javascript" src="ext.js">
        </script>
    </head>
    <body bgcolor="#00FFFF">
    </body>
</html>
```

Output:



In above example we define the some javascript code as to display **Hello!! CSE** with the help external javascript "ext.js" then it it is embedded in the HTML page "example.html"

Advantage:

It allows easy code readability.

Comments in JavaScript:

As programs become more complex. Need to add comments on what the code does and explains why. Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them. There are two types

1. Single line comment- it starts with two forward slash characters(//). The rest of the line is a comment. It may occupy a full line of its own or follow a statement.
2. Multi-line comment- Multiline comments start with a forward slash and an asterisk /* and end with an asterisk and a forward slash */. The content of comments is ignored, so if we put code inside /* … */, it won't execute.

JavaScript Variables:

JavaScript variables are containers for storing data values. JavaScript is dynamically typed (also called loosely typed) scripting language. That is, in javascript variables can receive different data types over time. Data types are basically typed of data that can be used and manipulated in a program.

Variables in JavaScript are containers which hold reusable data. It is the basic unit of storage in a program.

•The value stored in a variable can be changed during program execution.

•A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.

•In JavaScript, all the variables must be declared before they can be used.

To declare a variable, you use the **"var"** keyword followed by the variable name. A variable name can be any valid identifier. Values are assigned to the variable either while declaring the variable or after

declaring the variable.

Declaration:

Var variablename;

Example: var temp;

Here we have declared the “temp” variable, which allows us to store any kind of data.

Assign value to variable

1. **Var temp;**

   Temp=10;

   Here we first declare the "temp" variable and then store the value of 10 in it. Now this variable is considered an integer variable because it stores an integer value.

2. **Var temp=10;**

   Here we have stored the value of 10 in the “temp” variable at the time of declaration. Now this variable is considered an integer variable because it stores an integer value.

- You can declare two or more variables using one statement, each variable declaration is separated by a comma (,)

  Note:

  Distinguish between undefined and undeclared variables.

  An undefined variable is a variable that has been declared. Because we have not assigned it a value, the variable used the undefined as its initial value.

  In contrast, an undeclared variable is the variable that has not been declared.

- There are two types of variables in JavaScript:

  a. **local variable**

     A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.

  b. **Global variable.**

     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.

- **JavaScript variable hoisting**

  When executing JavaScript code, the JavaScript engine goes through two phases:

  1. Parsing
  2. Execution

In the parsing phase, The JavaScript engine moves all variable declarations to the top of the file if the variables are global, or to the top of a function if the variables are declared in the fucntion. In the execution phase, the JavaScript engine assigns values to variables and execute the code. Hoisting is a mechanism that the JavaScript engine moves all the variable declarations to the top of their scopes, either function or global scopes.

- **'let' and 'const' keywords**:

From ES6, you can use the "let" keyword to declare one or more variables. The " let " keyword is similar to the "var" keyword. However, a variable is declared using the **let** keyword is block-scoped, not function or global-scoped like the **var** keyword.

A block in JavaScript is denoted by curly braces ({}). For instance, the **if...else**, **do...while**, or **for** loop statement creates a block.

Whenever you declare a variable in a function, the variable is visible only within the function. You can't access it outside the function. **var** is the keyword to define variable for function-scope accessibility.

In ES6, "const" and "let" keywords allow developers to declare variables in the block scope, which means those variables, exist only within the corresponding block. i.e A block scope is the area within if, switch conditions or for and while loops. Generally speaking, whenever you see {curly brackets}, it is a block.

Example:

```
function foo()
{
    if(true)
    {
        var fruit1 = 'apple';       //exist in function scope
        const fruit2 = 'banana';    //exist in block scope
        let fruit3 = 'strawberry';  //exist in block scope
    }
    console.log(fruit1);
    console.log(fruit2);
    console.log(fruit3);
}

foo();
```

Output:

```
//apple
//error: fruit2 is not defined
```

//error: fruit3 is not defined

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.

1. **primitive data types**

There are five types of primitive data types in JavaScript. They are

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

2. **non-primitive data types**

The non-primitive data types are

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

Example:

```
<html>
<head>
<title>Example </title>
</head>
<body>
<script type="text/javascript">
var a = 17;
```

```
                var b = "suresh";
                var c = "";
                var d = null;
                document.write("Type of a = " + (typeof a));
                document.write("<br>");
                document.write("Type of b = " + (typeof b));
                document.write("<br>");
                document.write("Type of c = " + (typeof c));
                document.write("<br>");
                document.write("Type of d = " + (typeof d));
                document.write("<br>");
                document.write("Type of e = " + (typeof e));
                document.write("<br>");
            </script>
        </body>
    </html>
```

Output:

Type of a = number

Type of b = string

Type of c = string

Type of d = object

Type of e = undefined.

Object:

The above primitive data types are only their values contain only a single thing i.e a string or a number or whatever. In JavaScript, an object is a collection of properties, where each property is defined as a key-value pair of various data and more complex entities.

JavaScript is an object-based language. Everything is an object in JavaScript. JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where key is a string, and value can be anything.

There are 3 ways to create objects.

1. **By object literal**

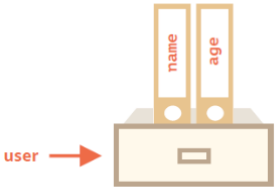
   The syntax of creating object using object literal is

   object= {property1:value1, property2:value2.....propertyN:valueN}

   here value and property is separated by colon(:)

example:

var user={name: “ ravi”, age:30}



In the above example, we are created **“user”** object and it has two properties those are **name** and **age**, their values are Ravi and 30. Property values are accessible using the dot notation

Document.write(user.name)

Document.write(user.age)

It can allow performing add, remove and read files operations at any time

2. **By creating instance of Object directly**

Syntax: var objectname = new Object()

Here “ new keyword ” is used to create an object

Example:

```
<script>
    var std=new Object();
    std.id=101;
    std.name="Ravi ";
    std.marks=86;
    document.write(std.id+" "+ std.name+" "+ std.marks);
</script>
```

Output:

101 Ravi 86

3. **By using an object constructor**

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The “ This keyword” refers to the current object.

Example:

```
<html>
    <body>
        <script>
            function emp(id,name,salary)
```

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

Output:

103 Vimal Jaiswal 30000

Screen Output & Keyboard Input:

When you load a document in your Web browser, it creates a number of JavaScript objects with properties and capabilities based on the HTML in the document and other information. These objects exist in a hierarchy that reflects the structure of the HTML page itself. The pre-defined objects that are most commonly used are the window and document objects. Some of the useful Objects are:

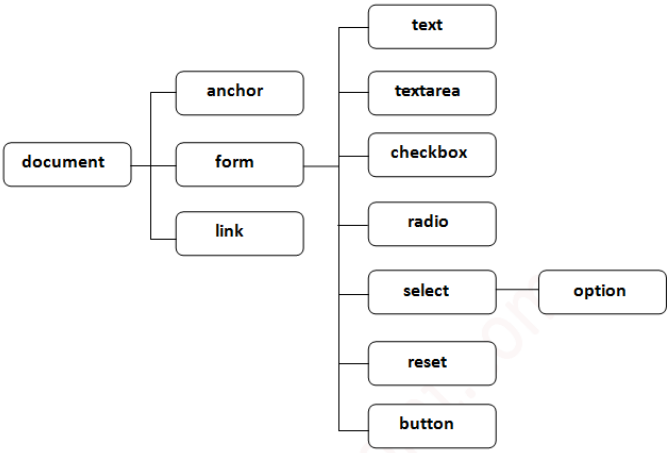
1. Document
2. Window
3. Browser
4. Form
5. Math
6. Date

1. The Document Object

The document object represents the whole html document. When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

Properties of Document object:

The properties of document object that can be accessed and modified by the document object.



Methods of document object:

Method	Description
write("string")	Writes the given string on the document.
writeln("string")	Writes the given string on the document with newline character at the end.
getElementById()	Returns the element having the given id value.
getElementsByName()	Returns all the elements having the given name value.
getElementsByTagName()	Returns all the elements having the given tag name.
getElementsByClassName()	Returns all the elements having the given class name.

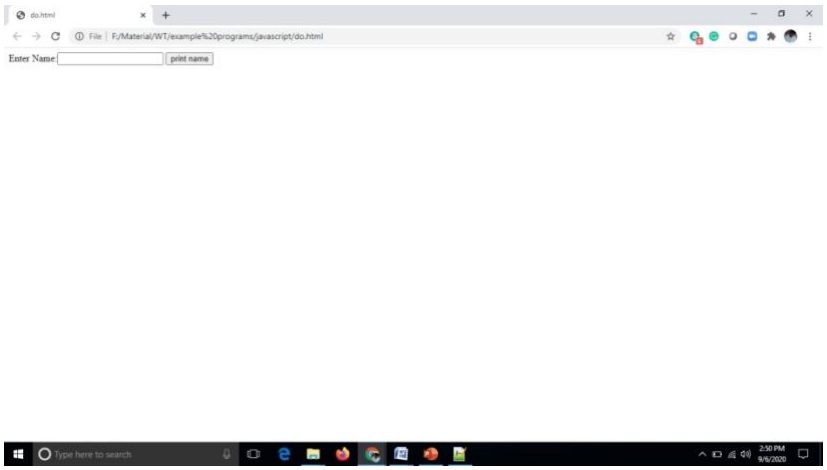
Example Program:

```
<html>
    <head>
        <script type="text/javascript">
            function printvalue()
            {
                var name=document.form1.name.value;
                alert("Welcome: "+name);
            }
        </script>
    </head>
    <body>
        <form name="form1">
```

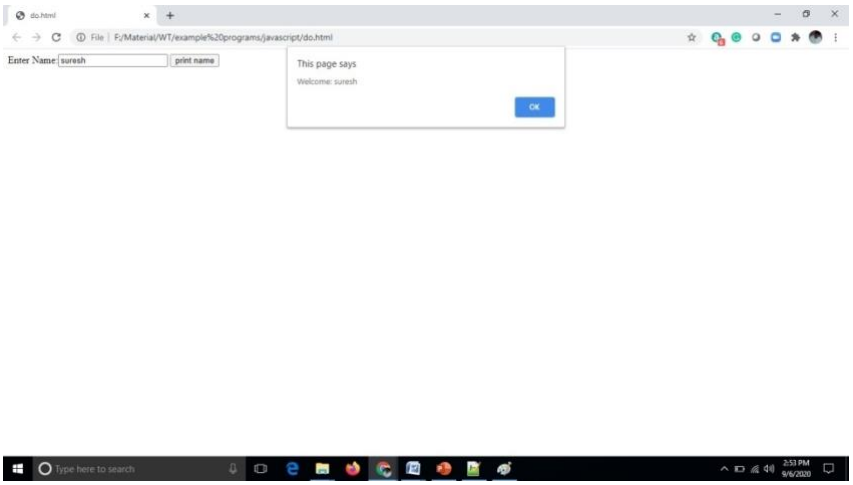
```
                Enter Name:<input type="text" name="name"/>
                <input type="button" onclick="printvalue()" value="print name"/>
            </form>
        </body>
    </html>
```

Result:

Input:



Output:



2. The Window Object

The Browser Object Model (BOM) is used to interact with the browser. The default object of browser is window means you can call all the functions of window by specifying window or directly. The window object is used to create a new window and to control the properties of window.

Window is the object of browser; it is not the object of javascript. The javascript objects are string, array, date etc.

Methods:

Method	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.
open()	Opens the new window.
close()	Closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Window object supports three types of message boxes.

1. Alert box
2. Confirm box
3. Prompt box

1. Alert box:

Alert box is used to display warning/error messages to user. It displays a text string with OK button.

Syntax:

window. Alert (“Message”);

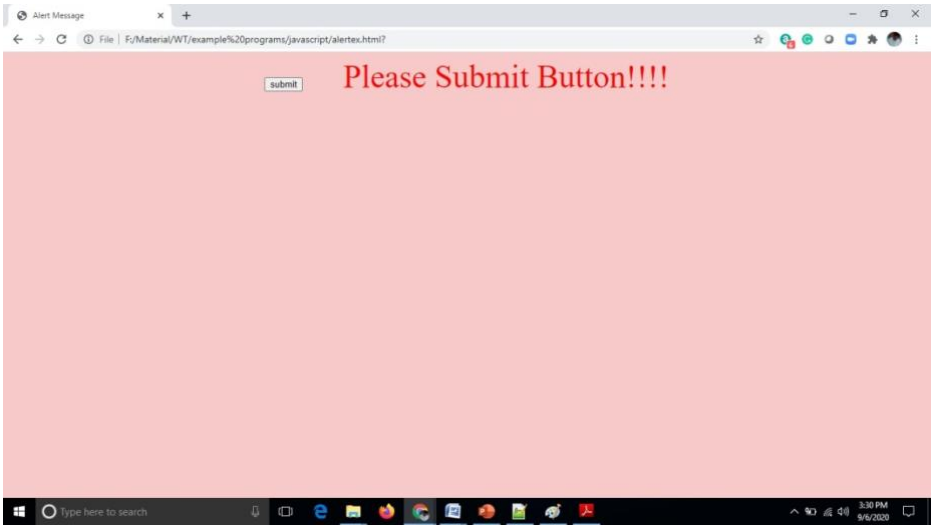
Example:

```
<html>
    <head>
        <title> Alert Message</title>
        <script type="text/javascript">
            function msg()
            {
                window.alert("Hai! Welcome to maya world!!!");
            }
        </script>
        <style>
            body{background-color: #F7CAC9}
            font{color:red}
        </style>
    </head>
```

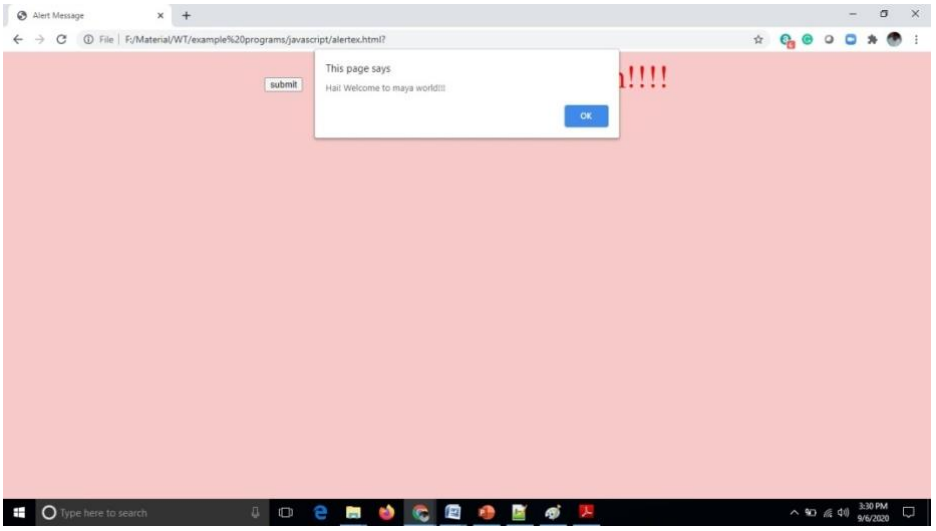
```
<body ><center>
<font size=”24”>
<form>
    <input type="submit" value="submit" onclick="msg()">    
    Please Submit Button!!!!</button>
</form>
</font>
</center>
</body>
</html>
```

Result:

Input:



Output:



2. Confirm Box

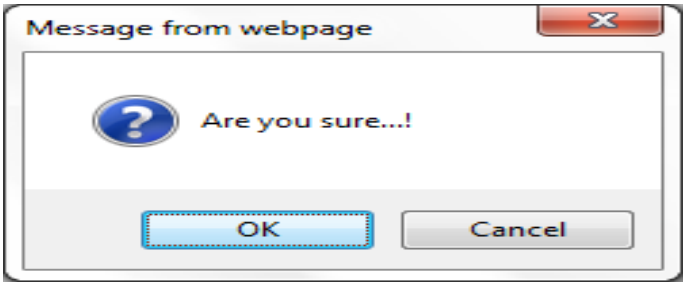
it is useful when submitting form data. This displays a window containing message with two buttons: OK and Cancel. Selecting Cancel will abort the any pending action, while OK will let the action proceed.

Syntax

window.confirm("String");

example:

```
<script type="text/javascript">
    function msg(){
        var v= confirm("Are u sure?");
        if(v==true){
            alert("ok");
        }
        else{
            alert("cancel");
        }
    }
</script>
<input type="button" value="delete record" onclick="msg()"/>
```



3. Prompt box:

It used for accepting data from user through keyboard. This displays simple window that contains a prompt and a text field in which user can enter data. This method has two parameters: a text string to be used as a prompt and a string to use as the default value. If you don't want to display a default then simply use an empty string.

Syntax

Variable=window.prompt("string","defaultvalue");

Example:

```
<script type="text/javascript">
    function msg(){
    var v= prompt("Who are you?");
```

```
        alert("I am "+v);


        }
</script>
<input type="button" value="click" onclick="msg()"/>
```

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. There are following types of operators in JavaScript.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operator
- Increment decrement operators
- Conditional/Ternary operator
- Bitwise operators

1. Arithmetic Operators

Arithmetic operators are 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 (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

2. Relational operators

The JavaScript comparison operator 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

3. Logical operators

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

4. 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

5. Conditional/Ternary operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax:

variablename = (condition) ? value1:value2

Example:

```
var voteable = (age < 18)? "Too young":"Old enough";
```

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

6. **Increment decrement operators**

JavaScript borrows increment and decrement operators from the C language. Increasing or decreasing a number by one is among the most common numerical operations. Both of increment and decrement operators have two versions: prefix and postfix.

You place the prefix version of the increment or decrement operators before a variable and the postfix versions after the variable.

Operator	Description	Example
++	Increment	x++ or ++x
--	Decrement	x-- or --x

The only difference between the postfix and prefix is that JavaScript doesn't evaluate them until the containing statement has been evaluated.

7. Bitwise operators

Bitwise operators treat arguments as 32-bit integer numbers and work on the level of their binary representation.

These operators are not JavaScript-specific. They are supported in most programming languages. These operators are used very rarely, when we need to fiddle with numbers on the very lowest (bitwise) level.

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

We won't need these operators any time soon, as web development has little use of them, but in some special areas, such as cryptography, they are useful.

Control Statements:

The working of control Statements in JavaScript as same as java language.

S.No.	Statement	Description
1.	switch case	A block of statements in which execution of code depends upon different cases. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.
2.	If else	The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
3.	While	The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.
4.	do while	Block of statements that are executed at least once and continues to be executed while condition is true.
5.	for	Same as while but initialization, condition and increment/decrement is done in the same line.
6.	for in	This loop is used to loop through an object's properties.
7.	continue	The continue statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.
8.	break	The break statement is used to exit a loop early, breaking out of the enclosing curly braces.
9.	return	Return statement is used to return a value from a function.

Arrays:

An array is an object that can store a collection of items. Arrays become really useful when you need to store large amounts of data of the same type.

An array is an ordered set of data elements which can be accessed through single variable name. You can access the data elements either sequentially by reading from the start of the array, or by using their index.

The basic operations that are performed on arrays are:

1) Creating arrays

2) Adding elements to an array

3) Accessing Array members

1. Creating an array

There are 3 ways to construct array in JavaScript. It also allows mixed data types

1. By array literal
2. By creating instance of Array directly

3. By using an Array constructor

1. By array literal

Syntax:

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

Example:

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

2. By creating instance of Array directly

Syntax:

var arrayname=new Array();

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

Example:

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

3. By using an Array constructor

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

Syntax:

var arrayname=new Array(value1,value2,….,valueN);

Example:

var emp=new Array("ramesh","suresh","rajesh");

2) Adding elements to an array

What happens if you want to add an item to an array that which is already full?

Many languages struggle with this problem but javascript has a really good solution – The interpreter simply extends the array and inserts the new item anywhere.

Example:

Var data=[10,20.3,”abd”,”Suresh”];

Data[4]=”kumar”;

Data[77]=”stupid”;

A new element is added at position 4 as well as at position 77

3) Accessing array members:

You can access the items in an array by referring to its “index number “ and the index of the first element of an array is zero.

<html>

<head>

```
<title>Accessing array elements </title>
</head><body>
<script type=”text/javascript”>
Var data=[“Wednesday”, “WT Lab”, “suresh”,77];
Var i=data.length;
For (var count =0; count<i ;count++)
{
Document.writeln(data[count] + “ , “);
}
Document.close();
</script></body></html>
```

Methods

The Array object has many properties and methods which help developers to handle arrays easily and efficiently.

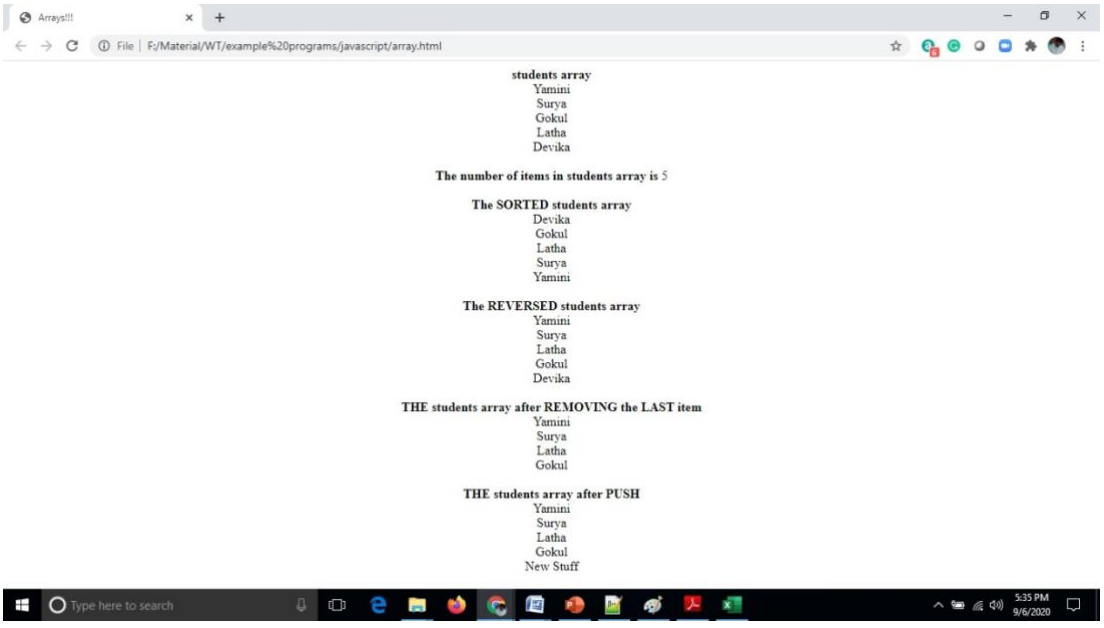
S.no	Method	Description
1.	Length	know the number of elements in an array
2.	reverse	reverse the order of items in an array
3.	sort	sort the items in an array
4.	pop	remove the last item of an array
5.	shift	remove the first item of an array
6.	push	add a value as the last item of the array.

Example:

```
<html>
<head>
<title>Arrays!!!</title>
<script type="text/javascript">
    var students = new Array("Yamini", "Surya", "Gokul", "Latha", "Devika");
    Array.prototype.displayItems=function(){
        for (i=0;i<this.length;i++){
            document.write(this[i] + "<br />");
        }
    }
    document.write("<center>");
    document.write("<b>students array</b><br />");
    students.displayItems();
```

```
        document.write("<b><br />The number of items in students array is </b>" + students.length
+ "<br />");
        document.write("<b><br />The SORTED students array<br /></b>");
        students.sort();
        students.displayItems();
        document.write("<b><br />The REVERSED students array<br /></b>");
        students.reverse();
        students.displayItems();
        document.write("<b><br />THE students array after REMOVING the LAST item<br
/></b>");
        students.pop();
        students.displayItems();
       document.write("<b><br />THE students array after PUSH<br /></b>");
       students.push("New Stuff");
        students.displayItems();
        document.write("</center>");
</script>
</head>
<body>
</body>
</html>
```

Output:



Functions

A function is a block of code which will be executed only if it is called. If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

Defining a Function

To create a function we can use a function declaration.

Synatx:

```
function functionname()
{
        lines of code to be executed
}
```

The function keyword goes first, and then goes the name of the function, then a list of parameters between the parentheses and finally the code of the function, also named "the function body", between curly braces. JavaScript Functions can have 0 or more arguments.

Example:

```
function welcome ()
{
        document.write ("This is a simple function. <br />");
}
```

Here we have defined the function with the "function" keyword, then defined function name as "welcome", then within the function, displaying the simple message "This is a Simple function". If you want to display a simple message you need to call it.

Calling a Function

Calling the function actually performs the specified actions. When you call a function, this is usually within the BODY of the HTML page, and you usually pass a parameter into the function on which the function will act.

Here's an example of calling the same function:

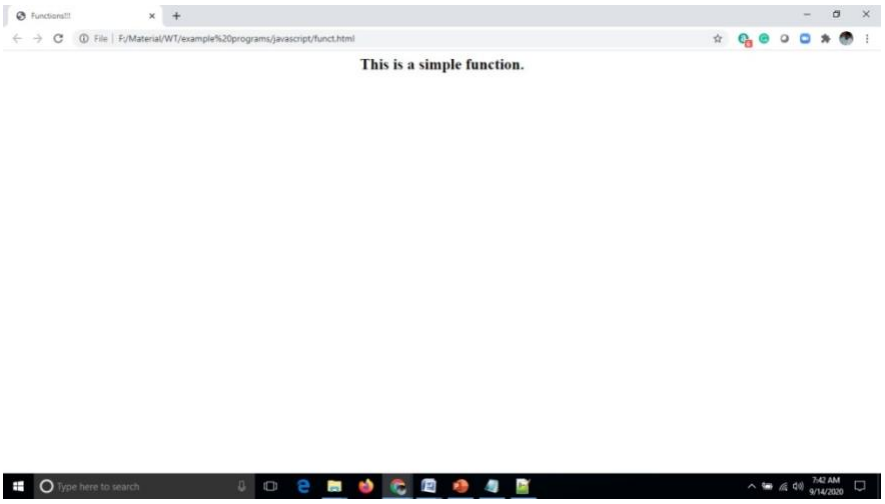
```
welcome();
```

Complete Example:

```
<html>
    <head>
        <title>Functions!!!</title>
        <script type="text/javascript">
        function welcome()
        {
```

```
                document.write("This is a simple function.<br />");
            }
            welcome();
            </script>
        </head>
        <body>
        </body>
    </html>
```

Output:



Returning a value

Every function in JavaScript returns undefined, unless otherwise specified. To specify a return value for a function, we use the return statement followed by an expression or a value.

return expression;

We can also use the **"return"** statement without value to exit the function prematurely.

JavaScript Constructor Method

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

The constructor keyword is used to declare a constructor method.

The class can contain one constructor method only.

JavaScript allows us to use parent class constructor through super keyword.

Example:

```
<html>
    <body>

        <script>
```

```
class Employee {
        constructor() {
            this.id=101;
            this.name = "Martin Roy";
        }
    }
    var emp = new Employee();
    document.writeln(emp.id+" "+emp.name);
</script>
</body>
</html>
```

JavaScript Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser.

When JavaScript code is included in HTML, it reacts over these events and allows the execution. This process of reacting over the events is called Event Handling. Thus, it handles the HTML events via Event Handlers.

Common HTML Events

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

Examples:

1. **Using onLoad:**

```
<html>
    <head>
        <title> function demo </title>
    </head>
    <body onLoad="about(34,'KK',420)">
        <script language="javascript">
```

```
                function about(age,name,behaviour)
                {
                        document.write("Age is::<br>" +age);
                        document.write("Name is ::" +name);
                        document.write("Behaviour is ::"  +behaviour);
                        document.close();
                }
            </script>
        </body>
    </html>
```

2. **Using onClick**:

```
    <html>
        <head>
            <title> using on click </title>
        </head>
        <body>
            <script type="text/javascript">
                functioncallme()
                {
                    alert("Hello!! How Are You!!!!")
                }
            </script>
            <button onclick="callme()"> Click ME </button>
        </body>
    </html>
```

Form Validation:

Form validation is the process of verifying whether the data provided by a user using an HTML form is as we expect it.

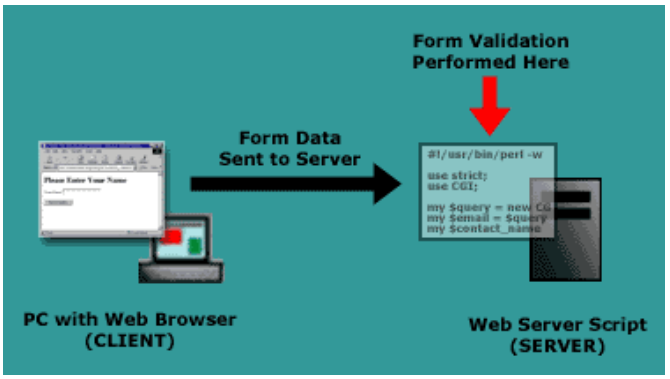
Example:

If you are using a registration form, and you want your user to submit name, email id and address, you must use a code to check whether the user entered a name containing alphabets only, a valid email address and a proper address.

There are two main methods for validating forms:

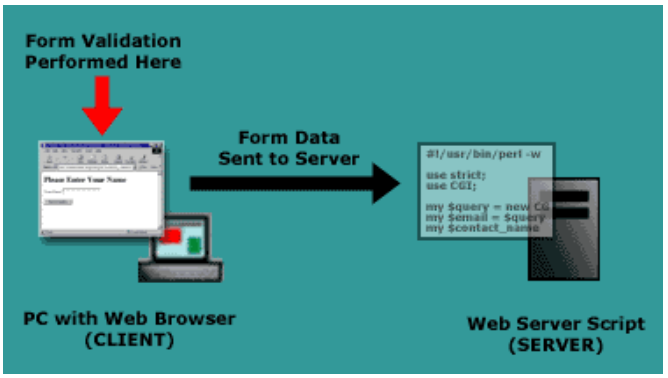
1. Server-side

   Server-side validation is more secure but often more tricky to code



2. Client-side.

   Client-side (JavaScript) validation is easier to do and quicker too. The browser doesn't have to connect to the server to validate the form, so the user finds out instantly if they've missed out that required field!.



The form validation process typically consists of two parts

1. **The required fields validation** which is performed to make sure that all the mandatory fields are filled in,
2. **Data format validation** which is performed to ensure that the type and format of the data entered in the form is valid.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Form Validation with JavaScript

1. Let's first create a simple HTML form that we will validate on client-side using JavaScript when the user clicks on the submit button.

Regular Expression:

A regular expression is a string that describes a pattern e.g., email addresses and phone numbers.

In JavaScript, regular expressions are objects. JavaScript provides the built-in **RegExp** type that allows you to work with regular expressions effectively.

Regular expressions are useful for searching and replacing strings that match a pattern. They have many useful applications.

For example, you can use regular expressions to extract useful information in web scraping like product prices. Or you can use regular expressions to validate form fields like email addresses and phone numbers.

Regular Expression is defined or created by following two ways

1. pattern is placed in between forward-slash character(/)

   ex: var re=/hi/;

   here we are define a regular expression "re" and stores value "hi"

2. by using RegExp Constructor

   ex: let re = new RegExp('hi');

   Both regular expressions are the instances of the RegExp type. They match the string 'hi'.

RegExp object has many useful methods.

1. test() method that allows you to test if a string contains a match of the pattern in the regular expression.

The test() method returns true if the string argument contains a match.

Using pattern flags

Besides a pattern, a RegExp object also accepts an optional flag parameter. Flags are settings that change the searching behavior.

1) The ignore flag (i)

Regular expressions have many flags. For example, the ignore flag ignores cases when searching. It is represented by **" i "**

By default, searches are case-sensitive. For example /hi/ only matches the string hi not Hi, or HI.

To search for a string with any cases, you use the i flag:

Example:

```
let re = /hi/i;
let result = re.test('Hi John');
console.log(result);
```

here it return "true" i.e the /hi/i will match any string hi, Hi, and HI.

Example 2:

```
let re = new RegExp('hi','i');
let result = re.test('HI John');
console.log(result);
```

here regular expression object is created with the help of regular expression

constructor and apply ignore case so it return “true” i.e the /hi/i will match any string hi, Hi, and HI.

2) The global flag (g)

In general the regular expression , the RegExp object only checks if there is a match in a string and returns the first match.

When the gloabal(g) flag is available, the RegExp looks for all matches and returns all of them. It’s possible to combine flags e.g., gi flags combine the ignore (i) and the global flag (g) flags.

Methods:

The exec() method of the RegExp performs a search for a match in a string and returns an array that contains detailed information about the match.

The exec() method returns null if it could not find any match. However, it returns a single match at once. To get all matches, you need to execute the exec() multiple times.

Searching strings

The method str.match(regexp) returns all matches of regexp in the string str. To find all matches, you use the global flag (g). And to find the matches regardless of cases, you use the ignore flag (i).

Example:

```
let str = "Are you Ok? Yes, I'm OK";
let result = str.match(/OK/gi);
console.log(result);
```

Output:

```
["Ok", "OK"]
```

Replacing strings

The following example uses the replace() method to replace the first occurrence the string 'Ok' in the string str:

Example:

```
let str = "Are you OK? Yes, I'm OK.";
let result = str.replace('Ok','fine');
console.log(result);
```

Output:

```
Are you fine? Yes, I'm OK
```

DYNAMIC HTML

DHTML is combination of HTML, CSS and JavaScript. It gives pleasant appearance to web page.

Difference between HTML and DHTML

HTML	DHTML
HTML is used to create static web pages.	DHTML is used to create dynamic web pages.
HTML is consists of simple html tags.	DHTML is made up of HTML tags+cascading style sheets+javascript.
Creation of html web pages is simplest but less interactive.	Creation of DHTML is complex but more interactive.

Positioning Moving and Changing Elements:

It is possible to position elements on the page in both Microsoft and Netscape's implementation of Dynamic HTML. By dynamically altering the positions of elements over time, we can create simple animation effects

Program:

```
<html>
    <head>
        <title> Position Moving </title>
        <script type="text/javascript">
            functionmoveleft()
            {
                document.getElementById('image').style.position="absolute";
                document.getElementById('image').style.left="0";
            }
            functionmoveright()
            {
                document.getElementById('image').style.position="absolute";
                document.getElementById('image').style.right="0";
            }
        </script>
    </head>
    <body>
        <img id="image" src=" Penguins.jpg" onclick="moveleft()" or
        onclick="moveright()" />
    </body>
</html>
```

Imp Questions:

1. How Object creation and Modification done in JavaScript?
2. Explain about Control statements used in JavaScript with examples.
3. Explain Built-In Objects
4. Illustrate the concept of Screen Output and Keyboard Input.
5. State the two approaches of Pattern matching used in JavaScript.
6. Explain the following terms with examples (i) Absolute Positioning (ii) Relative Positioning.