What is JavaScript?

- HTML and CSS concentrate on a static rendering of a page; things do not change on the page over time, or because of events.
- To do these things, we use scripting languages, which allow content to change dynamically.
- Not only this, but it is possible to interact with the user beyond what is possible with HTML.
- Scripts are programs just like any other programming language; they can execute on the client side or the server.

Advantages of client side scripting

- The web browser uses its own resources, and eases the burden on the server.
- It has fewer features than server side scripting.
- It saves network bandwidth.

Disadvantages of client side scripting

- Code is usually visible.
- Code is probably modifiable.
- Local files and databases cannot be accessed.
- User is able to disable client side scripting.

Differentiate between server side and client side scripting languages

Client-side scripting languages

- The client-side environment used to run scripts is usually a browser.
- The processing takes place on the end users computer.
- The source code is transferred from the web server to the user's computer over the internet and run directly in the browser.
- The scripting language needs to be enabled on the client computer.
- Sometimes if a user is conscious of security risks they may switch the scripting facility off.
- When this is the case a message usually pops up to alert the user when script is attempting to run.

Server-side scripting languages

- The server-side environment that runs a scripting language is a web server.
- A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages.
- This HTML is then sent to the client browser.
- It is usually used to provide interactive web sites that interface to databases or other data stores on the server.

- This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript.
- The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

What is difference between Java script and JAVA?

- Java is a statically typed language; JavaScript is dynamic.
- Java is class-based; JavaScript is prototype-based.
- Java constructors are special functions that can only be called at object creation; JavaScript "constructors" are just standard functions.
- Java requires all non-block statements to end with a semicolon; JavaScript inserts semicolons at the ends of certain lines.
- Java uses block-based scoping; JavaScript uses function-based scoping.
- Java has an implicit this scope for non-static methods, and implicit class scope; JavaScript has implicit global scope.

Embedded JavaScript

- JavaScript can be embedded in an HTML document.
- To embed it in HTML you must write:

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

- The script tag has effect of the stopping the JavaScript being printed out as well as indentifying the code enclosed.
- The JavaScript can be placed in the head section of your HTML or the body.

```
<html>
<body>
<script type="text/javascript">
        document.write("<h1>This is a heading</h1>");
</script>
</body>
</html>
```

- The Scripts placed in the body section are executed as the page loads and can be used to generate the content of the page.
- As well as the body section, JavaScript can also be placed in the head part.
- The advantages of putting a script in there are that it loads before the main body.

External JavaScript

- If you want to use the same script on several pages it could be a good idea to place the code in a separate file, rather than writing it on each.
- That way if you want to update the code, or change it, you only need to do it once.

- Simply take the code you want in a separate file out of your program and save it with the extension .js.

```
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

JavaScript Variables

- Variables in JavaScript behave the same as variables in most popular programming languages (C, C++, etc) do, but in JavaScript you don't have to declare variables before you use them.
- A variable's purpose is to store information so that it can be used later. A variable is a symbolic name that represents some data that you set.
- When using a variable for the first time it is not necessary to use "**var**" before the variable name.
- Variable names must begin with a letter.
- Variable names are case sensitive (y and Y are different variables).

```
var x=5;
var y=6;
var z=x+y;
```

- You can declare many variables in one statement. Just start the statement with var and separate the variables by comma:

```
var name="Doe", age=30, job="carpenter";
var name="Doe",
age=30,
job="carpenter";
```

- Variable declared without a value will have the value **undefined**.
- If you re-declare a JavaScript variable, it will not lose its value.
- The value of the variable *carname* will still have the value "Volvo" after the execution of the following two statements.

```
varcarname="Volvo";
varcarname;
```

JavaScript Operators

- Operators in JavaScript are very similar to operators that appear in other programming languages.
- The definition of an operator is a symbol that is used to perform an operation.
- Most often these operations are arithmetic (addition, subtraction, etc), but not always.

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
=	Assignment

```
<body>
<script type="text/JavaScript">
<!--
var two = 2
var ten = 10
varlinebreak = "<br />"
document.write("twoplusten=")
var result = two + ten
document.write(result)
//-->
</script>
</body>
```

Assignment	Equivalent to
X+=Y	X=X+Y
X-=Y	X=X-Y
X*=Y	X=X*Y
X/=Y	X=X/Y
X%=Y	X=X%Y

JavaScript Array

- An array is a special variable, which can hold more than one value at a time.
- The Array object is used to store multiple values in a single variable.
- An array can be created in three ways.
- The following code creates an Array object called myCars.

1. **Regular**

```
varmyCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2. **Condensed**

```
varmyCars=new Array("Saab","Volvo","BMW");
```

3. **Literal**

```
varmyCars=["Saab","Volvo","BMW"];
```

Access an Array

- You refer to an element in an array by referring to the **index** number.
- This statement access the value of the first element in myCars.

```
var name=myCars[0];
```

- This statement modifies the first element in myCars:

```
myCars[0]="Opel";
```

JavaScript Functions

- A function is a section of code that is separate from the main program.
- It is defined once but can be invoked many times.
- A function can be passed as parameters so that they can be used and a value can be returned back.
- There are some functions already built in to JavaScript, such as the Math.cos() function, which calculates the cosine of an angle.
- An example function could be:

```
functionmultByTen(x)
{
        return x*10;
}
```

- This can then be invoked by using the function's name complete with any parameters you want to pass:

```
mysum=multByTen(3)
```

- Below is an example of JavaScript function.

```
<html><body>
<script type="text/javascript">
var z= multXbyY(10,15);
document.write("The result is" +z);
functionmultXbyY(x,y) {
        document.write("x is" +x);
        document.write("y is "+y);
        return x*y;
}
</script>
</body></html>
```

JavaScript Conditions

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
    - **if statement** - use this statement to execute some code only if a specified condition is true
    - **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
    - **if...else if.. else statement** - use this statement to select one of many blocks of code to be executed
    - **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement

- Use the if statement to execute some code only if a specified condition is true.

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

If...else Statement

- Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

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

If...else if...else Statement

- Use the if....else if...else statement to select one of several blocks of code to be executed.

```
if (condition1) {
 code to be executed if condition1 is true
 }
else if (condition2)  {
 code to be executed if condition2 is true
 }
else {
 code to be executed if neither condition1 nor condition2 is true
 }
```

Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

```
switch(n)
{
case 1:
 execute code block 1
 break;
case 2:
 execute code block 2
 break;
default:
 code to be executed if n is different from case 1 and 2
}
```

The *default* Keyword

- Use the *default* keyword to specify what to do if there is no match.

Conditional Operator

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

```
variablename=(condition)?value1:value2
voteable=(age<18)?"Too young":"Old enough";
```

JavaScript Loops and Repetition

- Loops can execute a block of code a number of times.
- Loops are handy, if you want to run the same code over and over again, each time with a different value.

Different Kinds of Loops

- JavaScript supports different kinds of loops:
  - **for** - loops through a block of code a number of times
  - **for/in**- loops through the properties of an object
  - **while** - loops through a block of code while a specified condition is true
  - **do/while** - also loops through a block of code while a specified condition is true

The For Loop

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3)
 {
 the code block to be executed
 }
```

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

```
for (var i=0; i<5; i++)
 {
 x=x + "The number is " + i + "<br>";
 }
```

The For/In Loop

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

```
var person={fname:"John",lname:"Doe",age:25};
for (x in person)
 {
 txt=txt + person[x];
 }
```

The While Loop

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

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

The Do/While Loop

- The do/while 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.

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

JavaScript Objects

- JavaScript has several built-in objects, like String, Date, Array, and more.
- An object is just a special kind of data, with **properties** and **methods**.

Accessing Object Properties

- Properties are the values associated with an object.
- The syntax for accessing the property of an object is below

objectName.propertyName

- This example uses the length property of the String object to find the length of a string:

```
var message="Hello World!";
var x=message.length;
```

Accessing Objects Methods

- Methods are the actions that can be performed on objects.
- You can call a method with the following syntax.

```
objectName.methodName()
```

- This example uses the toUpperCase() method of the String object, to convert a text to uppercase.

```
var message="Hello world!";
var x=message.toUpperCase();
```

Creating JavaScript Objects

- With JavaScript you can define and create your own objects.
- There are 2 different ways to create a new object:
  - Define and create a direct instance of an object.
  - Use a function to define an object, then create new object instances.

Creating a Direct Instance

- This example creates a new instance of an object, and adds four properties to it:

```
person=new Object();
person.firstname="Narendra";
person.lastname="Modi";
person.age=24;
person.eyecolor="blue";
```

User- Defined Objects/How user defined objects are created in JavaScript?

- JavaScript allows you to create your own objects.

- The first step is to use the *new* operator.

```
VarmyObj= new Object();
```

- This creates an empty object.
- This can then be used to start a new object that you can then give new properties and methods.
- In object- oriented programming such a new object is usually given a constructor to initialize values when it is first created.
- However, it is also possible to assign values when it is made with literal values.

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

<script language="JavaScript" type="text/JavaScript">
person={
                firstname: "Ketan",
                lastname: "Chavda",
                age: 24,
                eyecolor: "blue"
        }
document.write(person.firstname + " is " + person.age + " years old.");
</script>

</body>
</html>
```

How a constructor can be used to populate data in the object?

- A constructor is pre defined method that will initialize your object.
- To do this in JavaScript a function is used that is invoked through the *new* operator.
- Any properties inside the newly created object are assigned using *this* keyword, referring to the current object being created.

```
<!DOCTYPE html>
<html>
<body>
<script>
function person(firstname, lastname, age)
{
        this.firstname=firstname;
        this.lastname = lastname;
        this. age = age;
}
var person1=new person("Narendra","Modi",24);
document.write(person1.firstname + " "+ person1.lastname +" "+ person1.age);
</script>
</body>
</html>
```

- In above the function *person*becomes the constructor invoked through the *new* keyword on assignment to the *person1* variable.
- Here the values are passed as parameters to the constructor.
- Inside the constructor the *this* keyword takes on the value of the newly created object and therefore applies properties to it.

Explain the event handling in JavaScript.

- Event handlers are attributes that force an element to "listen" for a specific event to occur.
- Event handlers all begin with the letters "on".
- There are two types of events in Javascript
    - Interactive i.g. onClick
    - Non-interactive i.g. onLoad
- The table below lists the HTML event handlers with descriptions.

Event Handler	Elements Supported	Description
Onblur	a, area, button, input, label, select, textarea	the element lost the focus
Onchange	input, select, textarea	the element value was changed
Onclick	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was clicked
Ondblclick	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was double clicked
Onfocus	a, area, button, input, label, select, textarea	the element received the focus
Onkeydown	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a key was pressed down
Onkeypress	All elements except br, font, frame, frameset,	a key was pressed and released

	head, html, iframe, isindex, meta, param, script, style, title	
Onkeyup	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a key was released
Onload	Frameset	all the frames have been loaded
Onload	Body	the document has been loaded
Onmousedown	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was pressed down
Onmousemove	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved within.
Onmouseout	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved away
Onmouseover	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer was moved onto
Onmouseup	All elements except br, font, frame, frameset, head, html, iframe, isindex, meta, param, script, style, title	a pointer button was released
Onreset	Form	the form was reset
Onselect	input, textarea	some text was selected
Onsubmit	Form	the form was submitted
Onunload	Frameset	all the frames have been removed
Onunload	Body	the document has been removed

Explain document object in JavaScript.

- There are lots of objects that descend from the *document* object forming a large sub-tree known as the Document Object Model (DOM), which has become standardized.
- The *document* object represents the HTML displayed in window.
- Using this object it is possible to access information about the document itself and also about the information content.
- Using this object it is possible to control certain parameters of the current document like background, foreground and link colors.

```
VarmyObj= new Object();
document.bgColor = "#9F2020";
documet.fgcolor = "#FAF519";
```

- It is possible to access any form information in document by using the *form[]* array.
- This contains all the *form* objects that are in the document.
- For example, a form can be constructed with :

```
<form name=”userDetails”>
<input type=”text” name= “ fname“/>
<input type=”text” name= “lname“/>
<input type=”submit” name= “ Submit“/>
```

- The form data can then be accessed with various DOM syntax constructions. The form itself can be accessed through:

  document.forms[0];

- It can also referred to by

  document.userDetails

- An individual element can then be accessed:

  document.userDetails.fname

- Below example shows the use of *document* object.

```
<html><head>
<title>Document Object Example</title>

<script type="text/javascript">
functionincrementCurrent() {
current = parseInt(document.forms["noteForm"].total.value);
document.forms["noteForm"].total.value = current + 1;
   }
</script>

</head><body>
<div id="mainDiv">
<h1>Document Object Example</h1>
<form id="noteForm">
        Current number of notes:
<input type="text" name="total" value="0" size="3"/>
<input type="button" value="Add a new note"
onclick="incrementCurrent()"/>
</form>
</div>
</body></html>
```

Why do you need validation? Show the use of regular expression in JavaScript to validate the email address with example.

- The idea behind JavaScript form validation is to provide a method to check the user enteredinformation before they can even submit it.
- JavaScript also lets you display helpful alerts to inform the user what information they have entered incorrectly and how they can fix it.

JavaScript Email Address validation using Regular Expression

- Using Regular expression we do checking for valid information.

```
<html>
<head>
<title>JavaScript Forms</title>
</head>

<body>
<form method="post" name="getinfo" onSubmit="return processForm()">
<input type="text" name="email"/>
<input type="submit" value="log in" name="Login"/>
</form>

<script language="JavaScript" type="text/JavaScript">
functionprocessForm()
{
varmyform= document.getinfo;
var check= myform.email.value;
Document.write(testEmail(check));
}
Function testEmail(chkMail)
{
varemailpattern = "^*\\w-_\.]*[\\w-_\.]\@[\\w]\.+[\\w]+[\\w+$";
var regex = new RegExp(emailpattern);
returnregex.test(chkMail);
}
</script>
</body>
</html>
```

- This will check for a valid email as describe.
- The testEmail() function returns true or false.

- The way it determines this end result is built on the template / pattern in the string *emailpattern*.
- This is used to work out the order of expected characters, how many times they repeat and specially occurring punctuation.
- The string in this case that is used as template is:

  "^*\\w-_\.][\\w-_\.]\@[\\w]\.+[\\w]+[\\w+$"*

- The first section is:

  ^[\\w-_\.]

- This sequence, beginning with ^, means check the first character is a word character represented by \\w.
- The next part is :

  **[\\w-_\.]*

- The * means that the next series of characters described can be represented many times or not at all.
- The characters themselves are the same as before; that is word characters, underscore, hyphen or period.

  \@[\\w]\.+

- This section begins by checking for the @ character.
- Following this should ne the word characters and then at least one 'dot'.
- In other words it would not accept a dot straight after the @ character.
- The last part is :

  [\\w]+[\\w]$

- The first set in square brackets makes sure that there are some characters after the dot and the last part checks that the last character is a word character.
- In this program after the string is declared, a regular expression object is created with the pattern:

  var regex = new RegExp(emailpattern);

- The pattern can then be tested against the incoming parameter with object's test method:

  returnregex.test(chkMail);

- This will return true or false depending on whether there is a match or not.

List the important built•in objects.

Built•in Objects:

- JS String
- JS Date

- JS Array
- JS Boolean
- JS Math
- JS RegExp

Example of inbuilt object

Math Object:

- The Math object allows you to perform mathematical tasks.
- The Math object includes several mathematical constants and methods.
- Syntax for using properties/methods of Math:

```
var x=Math.PI;
var y=Math.sqrt(16);
```

Math Object Properties

Property	Description
E	Returns Euler's number(approx.2.718)
LN2	Returns the natural logarithm of 2 (approx.0.693)
LN10	Returns the natural logarithm of 10 (approx.2.302)
LOG2E	Returns the base-2 logarithm of E (approx.1.442)
LOG10E	Returns the base-10 logarithm of E (approx.0.434)
PI	Returns PI(approx.3.14)
SQRT1_2	Returns the squareroot of 1/2(approx.0.707)
SQRT2	Returns the squareroot of 2(approx.1.414)

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between-PI/2 and PI/2radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of Ex
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm(base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x