UNIT 2

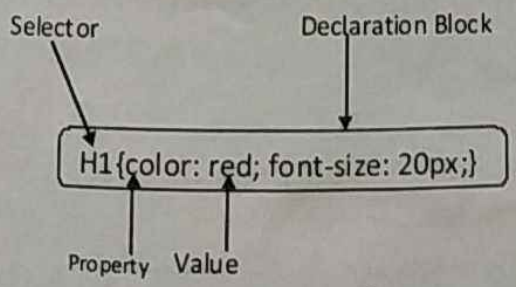
CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in HTML. It provides an additional feature to HTML. Using CSS, we can control the colour of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, borders and its colours, what background images or colours are used, as well as a variety of other effects in a web page. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

CSS Syntax

A CSS rule set contains a selector and a declaration block.



Selector: Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

1. color: red;
2. font-size: 20 px;

Each declaration contains a property name and value, separated by a colon.

Property: A Property is an of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "red" is assigned to color property.

```
Selector{
Property1: value1;
Property2: value2;
.....;}
```

CSS Selector

CSS selectors are used *to select the content you want to style*. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

1. The CSS Element Selector

The element selector selects the HTML element by name. For example

```
<html>
<head>
<style>
p{
  color: red;
  text-align: center;
font-family: algerian
}
</style>
</head>
<body>
<p>This is my css paragraph. </p>
</body>
</html>
```

Output

THIS IS MY CSS PARAGRAPH.

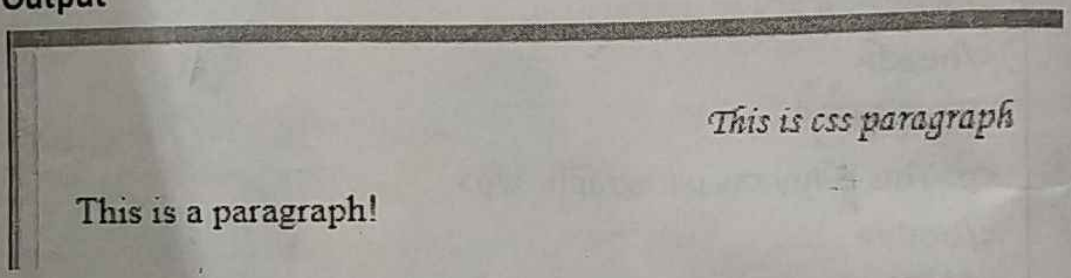
2. CSS Id Selector

The *id* selector selects the *id* attribute of an HTML element to select a specific element. An *id* is always unique within the page so it is chosen to select a single, unique element.It is written with the hash character (#), followed by the id of the element. For example

```
<html>
<head>
<style>
#p1{
  color: red;
```

```
    text-align: center;
    font-family: Monotype Corsiva;
}
</style>
</head>
<body>
<p id="p1">This is css paragraph </p>
<p>This is a paragraph! </p>
</body>
</html>
```

Output



3. CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character (.) followed by the class name. A class name should be unique. For example

```
<html>
<head>
<style>
.center {
   color: red;
   text-align: center;
   font-family: Monotype Corsiva;
}
```

```
</style>
</head>
<body>
<h1 class="center">This is css heading.</h1>
<p class="center">This is css paragraph.</p>
</body>
</html>
```

Output

This is css heading.

This is css paragraph.

CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector. For

Example

```
<html>
<head>
<style>
p.center {
  color: red;
  text-align: center;
  font-family: Monotype Corsiva;
}
</style>
</head>
<body>
<h1 class="center">This is a heading.</h1>
<p class="center">This is css paragraph.</p>
</body>
</html>
```

output

This is a heading.

This is css paragraph.

4. CSS Universal Selector

The universal selector is used as a wildcard character (*). It gives the style to all the elements on the page.

```
<html>
<head>
<style>
* {
   color: cyan;
   text-align: center;
   font-family: Monotype Corsiva;
}
</style>
</head>
<body>
<h1 >This is css heading.</h1>
<p >This is css paragraph.</p>
Normal text
</body>
</html>
```

Output

This is css heading.

This is css paragraph.

Normal text

5. CSS Group Selector

The grouping selector is used to give the style to a group or multiple elements with the same style definitions. It reduces the length of the code. Commas are used to separate each selector in grouping. For example

```
<html>
<head>
<style>
h1,h2,p {
  color: red;
  text-align: center;
  font-family: Monotype Corsiva;
}
</style>
</head>
<body>
<h1 >This is css heading.</h1>
<h2 >This is css heading.</h1>
<p >This is css paragraph.</p>
</body>
</html>
```

Output

This is css heading.

This is css heading.

This is css paragraph.

Adding CSS to HTML document

There are three different ways to add css to our HTML document. They are

1. Inline

Inline CSS is used to apply CSS on a single line or element. If you want to use inline CSS, use the *style* attribute to the tag.

Syntax:

<tag style="property1:value; property2:value;">

The main disadvantage of inline css is that we cannot reuse the style anywhere.

Usually, using of inline css is not recommended in your web pages because they cause problems and make the pages a lot more work to maintain. The only time we use them is when we want to check a style quickly during development

Example

```
<html>
<body>
<p style="color:blue;text-align:center;font-size:larger;">This is an
Inline css paragraph.</p>
</body>
 </html>
```

2. Internal

The internal style sheet is used to add a unique style for a single document. Internal styles are defined within the <style> element, inside the <head> section of an HTML page. For example

```
<html>
<head>
<style>
body {
  background-color: green;
}
```

```
h1 {
   color: white;
   border-style: solid;
   border-color: red;
}
</style>
</head>
<body>
<h1>The is an internal style sheet heading.</h1>
<p>This is a paragraph!!!</p>
</body>
</html>
```

3. External

The external style sheet is generally used when you want to make changes on multiple pages. Advantage of using external style sheet is to change the look of the entire web pages by changing just one file. The external style sheet is saved with .css extension

It uses the <link> tag on every pages and the <link> tag should be put inside the head section. For Example:

mystylesheet.css

```
body {
   background-color: green;
}
h1 {
   color: white;
   border-style: solid;
   border-color: red;
}
```

Following code is used to include .css file in our web page.

```
<head>
   <link rel="stylesheet" type="text/css" href="mystylesheet.css">
</head>
```

CSS Comments

CSS comments are generally written to explain our code.Comments are ignored by browsers.Comments are single or multiple lines statement and written within /*.....*/ .

```
h1 {
  color: white;
 /* border-style: solid;
  border-color: red;  */
}
```

CSS properties

Following are the basic CSS properties

- Text Properties
- Font Properties
- List Properties
- Border Properties

1. Text Properties

CSS Text properties are used for how to manipulate text in webpage. The following are the important text properties.

Text color

The *color* property is used to set the color of the text. The color is specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

Example

```
<head>
<style>
P{
   Color: green;
   }
```

```
</style>
</head>
<body>
<p>this is CSS text color</p>
</body>
```

Text align

The CSS text-align property is used for aligning elements left, right, center and justify.

```
<style>
 p {
  text-align: right;
 }
</style>
<p>This CSS text is aligned right</p>
```

Text Direction

The direction property is used to change the text direction of an element. Possible values are *ltr or rtl*.

```
<style>
 p {
  direction: rtl;
 }
</style>
```

Letter Spacing

The letter-spacing property is used to specify the space between the characters in a text. Using this we can increase or reduce the spacing. For reducing the space use minus numbers(eg. -5px)

```
<style>
 p {
  letter-spacing: 5px;
 }
</style>
```

Word Spacing

The word-spacing property is used to specify the space between the words in a text. Using this we can increase or reduce the spacing. For reducing the space use minus numbers(eg. -5px)

```
<style>
 p {
  word-spacing: 50px;
 }
</style>
```

Text Decoration

The text-decoration property is used to set or remove decorations from text. The value text-decoration: none; is often used to remove underlines from links.Possible values are *none, underline, overline, line-through*.

```
<style>
 .overline {
  text-decoration: overline;
 }
 .line-through {
  text-decoration: line-through;
 }
 .underline {
  text-decoration: underline;
 }
 a:link {
  text-decoration: none;
 }
</style>

<p class="overline">This text has a line over the top</p>
<p class="line-through">This text has a line through the middle</p>
<p class="underline">This text has a line underneath</p>
<a href="www.google.com" >This hyperlink has no underline</a>
```

2. Font Properties

CSS font properties enable you to change the look of your text. For example, you can assign a font family, apply bold or italic formatting, change the size and more.

Font Family

The font-family property allows you to set the font family.

```
<style>
 p {
  font-family: Georgia, Garamond, serif;
 }
</style>
```

Font Style

Possible values of font-style property are *normal, italic and oblique.*

```
<style>
 p {
  font-style: italic;
 }
</style>
```

Font Weight

This is used to how to set the font weight of an element. The font-weight property provides the functionality to specify how bold a font is. Possible values could be *normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900.*

```
<style>
 p {
  font-weight: bold;
 }
</style>
```

Font Size

This is used to set the font size of an element. The font-size property is used to control the size of fonts. Possible values could be *xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, size in pixels or in %.*

```
<style>
 p {
  font-size: 30px;
 }
</style>
```

Font Size Adjust

This property enables you to adjust the x-height to make fonts more legible. For more info, see the font-size-adjust page.

```
<style>
 p {
  font-size: 12px;
  font-size-adjust: 0.58;
 }
</style>
```

Font Property

The font property is a shorthand property that enables you to set all font properties at once.

```
<style>
 p {
  font: italic small-caps bold 20px Georgia, Garamond, serif;
 }
</style>
```

3. Backgrounds properties

Background-color

The *background-color* property is used to set the background color of an element.

```
<style>
 p {
   background-color: yellow;
 }
</style>
```

Background-image

The *background-image* property is used to set the background image of an element.

```
<style>
p {
   padding: 70px;
   background-image: url("/img/abc.jpg");
 }
</style>
```

Background-repeat

The *background-repeat* property is used to control the repetition of an image in the background. Possible values, of background-repeat are:

```
background-repeat: repeat-x;
background-repeat: repeat-y;
background-repeat: repeat;
background-repeat: space;
background-repeat: round;
background-repeat: no-repeat;
background-repeat:   space repeat;
```

By default, the repeated images are clipped to the size of the element, but they can be scaled to fit (using round) or evenly distributed from end to end (using space).

```
<style>
    body {
      background-image: url("/img/abc.jpg ");
      background-repeat: repeat;
    }
</style>
```

Example for how to repeat the background image vertically.

```
<style>
    body {
      background-image: url("/img/abc.jpg ");
      background-repeat: repeat-y;
    }
</style>
```

Example for how to repeat the background image horizontally.

```
<style>
    body {
      background-image: url("/img/abc.jpg ");
      background-repeat: repeat-x;
    }
</style>
```

Background Image Position

The *background-position* property is used to control the position of an image in the background. To set the background image position 100 pixels away from the left side.

```
<style>
    body {
      background-image: url("/img/abc.jpg");
      background-position:100px;
    }
</style>
```

Background Attachment

Background attachment determines whether a background image is fixed or scrolls with the rest of the page.Following example shows to set fixed background image.

```
<style>
    body {
      background-image: url("/img/abc.jpg");
      background-repeat: no-repeat;
      background-attachment: fixed;
    }
</style>
```

Following example shows to set scrolling background image.

```
<style>
    body {
      background-image: url("/img/abc.jpg");
      background-repeat: no-repeat;
      background-attachment:scroll;
    }
</style>
```

Background or shorthand property

Use the *background* property to set all the background properties at once.

```
<style>
 div {
  padding: 70px;
  height: 70px;
  width: 150px;
  background: url("/pix/samples/bg2.png") repeat fixed;
  overflow: auto;
 }
</style>
```

4. CSS - Links

You can set following properties of a hyper link –

- The :link signifies unvisited hyperlinks.
- The :visited signifies visited hyperlinks.
- The :hover signifies an element that currently has the user's mouse pointer hovering over it.
- The :active signifies an element on which the user is currently clicking.

Links can be styled with any CSS property (e.g. color, font-family, background, etc.).

```
<style type = "text/css">
  a:link {color: #000000}
  a:visited {color: #006600}
  a:hover {color: #FFCC00}
  a:active {color: #FF00CC}
</style>
```

The following example shows to create a css link button.

```
<html>
<head>
<style>
a:link, a:visited {
   background-color: #f44336;
   color: white;
   padding: 14px 25px;
   text-align: center;
   text-decoration: none;
   display: inline-block;
}
a:hover, a:active {
   background-color: red;
```

```
}
</style>
</head>
<body>

<a href="default.asp" target="_blank">This is a link</a>

</body>
</html>
```

5. CSS - Lists

The following properties, which can be used to control lists –

- list-style-type
- list-style-position
- list-style-image
- list-style

list-style-type

CSS list-style-type is using change the style of List Bullet. For example:

```
<style>
  ul{
        list-style-type: square;
  }
</style>
```

If you don't want any bullet in your list then set list-style-type: none;

```
<style>
  ul {
        list-style-type: none;
   }
```

Example for ordered list

```
<style>
ol{
   list-style-type:lower-alpha;
}
</style>
```

These are the values, that are used for an ordered list

Value	Description	Example
decimal	Number	1,2,3,4,5
decimal-leading-zero	0 before the number	01, 02, 03, 04, 05
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V
lower-greek	The marker is lower-greek	alpha, beta, gamma
lower-latin	The marker is lower-latin	a, b, c, d, e
upper-latin	The marker is upper-latin	A, B, C, D, E
hebrew	The marker is traditional Hebrew numbering	
armenian	The marker is traditional Armenian numbering	
georgian	The marker is traditional Georgian numbering	

list-style-position

It is possible to alter the indentation that takes place with your list items.

```
<style>
ul {
        list-style-position: inside;
       }
</style>

<style>
ol {
        list-style-position: outside;
      }
</style>
```

list-style-image

CSS lists allow you to insert an image in place of the normal bullets.

```
<style>
ul {
        list-style-image: url("arrow1.gif");
       }
</style>

<style>
ol {
        list-style-image: url("arrow2.gif");
        }
</style>
```

list-style Property

The *list-style* allows you to specify all the list properties into a single expression.

```
<style>
ul {
   list-style: inside   url("p1.jpg") ;
}
</style>
```

Pseudo-classes selectors

A CSS pseudo-class is a keyword added to selectors that specifies a special state of the element to be selected. For example: hover will apply a style when the user hovers over the element specified by the selector.

Pseudo-classes, together with pseudo-elements, let you apply a style to an element not only in relation to the content of the document tree, but also in relation to external factors like the history of the navigator (:visited, for example), the status of its content (like : checked on some form elements), or the position of the mouse (like :hover which lets you know if the mouse is over an element or not).

Syntax

```
selector:pseudo-class {
property: value;
}
```

Dynamic pseudo-classes

- :link
- :visited
- :hover
- :active
- :focus

UI element states pseudo-classes

- :enabled
- :disabled
- :checked

Structural pseudo-classes

- :first-child
- :nth-child(n)
- :nth-last-child(n)
- :nth-of-type(n)
- :nth-last-of-type(n)
- :last-child
- :first-of-type
- :last-of-type
- :only-child
- :only-of-type
- :root
- :empty

Other pseudo-classes

- :not(x)
- :target
- :lang(language)

```
/* We highlight the link when it is
hovered over (mouse over), activated (mouse down)
or focused (keyboard) */
a:hover,
a:active,
a:focus {
color: darkred;
text-decoration: none;
}
```

Pseudo-elements

Just like pseudo-classes, pseudo-elements are added to selectors but instead of describing a special state, they allow you to style certain parts of a document.Pseudo-elements are very much like pseudo-classes, but they have differences. They are keywords, this time preceded by two colons (::), that can be added to the end of selectors to select a certain part of an element.

For example, the ::first-line pseudo-element targets only the first line of an element specified by the selector.

Syntax

```
selector::pseudo-element {
property: value;
}
```

CSS classes can also be used with pseudo-elements "

```
selector.class:pseudo-element {
property: value;
}
```

Example

```
p::first-line {
text-decoration: underline; }
p::first-letter {
font-size: 5em; }
p::after {
content: url(/images/bullet.gif)      }
```

These are the some of the pseudo-elements

- ::after
- ::before
- ::first-letter
- ::first-line
- ::selection
- ::backdrop
- ::placeholder
- ::marker
- ::spelling-error
- ::grammar-error

Note that sometimes you will see double colons (::) instead of just one (:). This is part of CSS3 and an attempt to distinguish between pseudo-classes and pseudo-elements. Most browsers support both values.

::selection always starts with double colons (::). You can use only one pseudo-element in a selector. It must appear after the simple selectors in the statement.

The difference between pseudo-classes and pseudo-elements is that; basically a pseudo-class is a selector that assists in the selection of something that cannot be expressed by a simple selector, for example :hover. A pseudo-element however allows us to create items that do not normally exist in the document tree, for example "::after'.

JavaScript

JavaScript is a client side scripting language used to make web pages interactive. JavaScript was developed by *Brendan Eich* while he was working for Netscape Communications Corporation. On the client side, JavaScript is implemented as an interpreted language.

JavaScript is an open source & most popular client side scripting language supported by all browsers. JavaScript is used mainly for enhancing the interaction of a user with the webpage. JavaScript is now used for almost anything up to large applications and games, and can even be found in servers.

JavaScript was developed by Brendan Eich in 1995, which appeared in Netscape, a popular browser of that time. The language was initially called LiveScript and was later renamed JavaScript. JavaScript is a scripting language 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. JavaScript is a case-sensitive language.

The main advantage of JavaScript is that all modern web browsers support JavaScript. JavaScript runs on any operating system including Windows, Linux or Mac. Following are the benefits of using JavaScript:

1. It is widely supported in Web Browsers.
2. It gives easy access to the document objects and can manipulate most of them.
3. JavaScript is relatively secure- JavaScript can neither read from your local hard drive nor write to it, and you can't get a virus infection directly from JavaScript.

Simple JavaScript Program

You can place your javascript code in <script> tags(<script> and </script>). You have to use the type attribute within the <script> tag and set its value to text/javascript like this:

```
    <script type="text/javascript">
A simple hello world example:
<html>
<head>
  <title>—-My First JavaScript program—-</title>
  <script type="text/javascript">
        alert("Hello World!");
  </script>
</head>
<body>
</body></html>
```

We can include JavaScript code anywhere in an HTML document. That are follows "

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in an external file and then include in <head>...</head> section.

Note that we can include javascript code in <head> and <body> section together.

Javascript in <head> section

JavaScript is run based on event such as onclick, form load etc.

```
<html>
<head>
<script type="text/javascript">
        function hello() {
          alert("Hello World")
        }
</script>
</head>
<body>
<form >
<input type="button" onclick="sayHello()" value="Say Hello" />
</form>
</body>
</html>
```

JavaScript in <body>section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
        document.write("Hello World")
</script>
<p>This is web page body </p>
</body>
</html>
```

JavaScript in External File

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files. So you need to save your javascript program with the file extension ***.js***. The following example shows how to include javascript file in tour html file.

```
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
```

JavaScript variables and datatypes.

Variables are named containers to store values(a=10) and expressions(s=a+b).

Declaring Variables in JavaScript

Before using a variable, you first need to declare it. You have to use the keyword **var** to declare a variable like this:

```
var name;
```

Assign a Value to the Variable

You can assign a value to the variable either while declaring the variable or after declaring the variable.

```
var name = "John";
OR
var name;
name = "John";
```

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

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

```
<html>
<head>
<title>Variables!!!</title>
<script type="text/javascript">
        var one = 22;
        var two = 3;
        var sum= one + two;
   document.write("First No: = " + one + "<br />Second No: = " + two + " <br />");
   document.write(one + " + " + two + " = " + sum+ "<br/>");
</script>
</head>
<body>
</body>
</html>
```

There are two types of variables in javascript 1. Local Variable 2. Global Variable.

1. Local Variable

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

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

2. Global variable

Global variable is accessible from any function. A variable i.e. declared outside the function is known as global variable. For example:

```
<script>
var x=10; //  gloabal variable
function a(){
document.write(x);
}
a(); // calling the function
</script>
```

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:

```
var a=10;//holding number
var b="welcome";//holding string
```

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. 10
Boolean	represents boolean value either ' true' or 'false'.

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
RegExp	represents regular expression

These are explained in the next section.

Operators

An operator operates on operands. In JavaScript operators are used for compare values, perform arithmetic operations etc. There are various operators supported by JavaScript:

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

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

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:

```
<script type="text/javascript">
var a= 10;
var b= 3;
var sum= a + b;
var sub= a - b;
var mul=a * b;
var div=a / b;
var rem= a % b;
        document.write(a + " + " + b+ " = " + sum+ "<br/>");
        document.write(a + " - " + b+ " = " + sub+ "<br/>");
        document.write(a + " * " + b+ " = " + mul+ "<br/>");
        document.write(a + " / " + b+ " = " + div+ "<br/>");
        document.write(a + " % " + b+ " = " + rem+ "<br/>");
</script>
```

Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

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

Logical Operators

The following operators are known as JavaScript logical operators.

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

Assignment Operators

The following operators are known as JavaScript assignment operators.

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

JavaScript 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

Special Operators

The following operators are known as JavaScript special operators.

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

Decision Control Statements

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

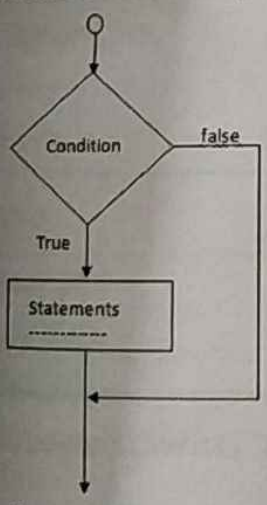
1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The synatx of JavaScript if statement is given below.

```
if(expression){
        statements;
        ----------
        ----------
}
```

Flowchart of JavaScript If statement



Example

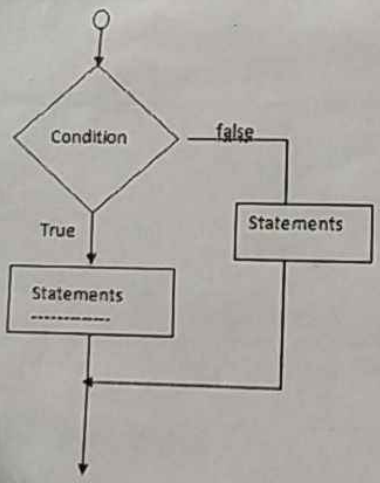
```
<script>
var a=15;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```

If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
        true block statements;
        ----------
}
else{
        false block statements;
        ----------
}
```

Flowchart of JavaScript If...else statement



Example

```
<script>
var a=10;
        if(a%2==0){
            document.write(a," is even number");
        }
        else{
            document.write(a," is odd number");
        }
</script>
```

If...else if statement

It evaluates the content only if expression is true from several expressions. The syntax of JavaScript if else if statement is given below.

```
if(expression1){
statements;
———
}
else if(expression2){
statements
———
}
else if(expression3){
statements
———
}
else{
statements
———
}
```

Example:

```
<script>
var a=3;
if(a>0){
document.write("a is positive");
}
else if(a<0){
document.write("a is negative");
}
else{
document.write("a is zero");
}
</script>
```

Switch statement

The **JavaScript switch statement** is used *to execute one code from multiple expressions*.The syntax of JavaScript switch statement is given below.

```
switch(expression){
case value1:
    statements
    ———
 break;
case value2:
    statements
    ———
 break;
    ———
    ———
default:
   default statements
   ——————————
}
Example:
<script>
var grade='B';
var result;
switch(grade){
case 'A':
        result="A Grade";
break;
case 'B':
        result="B Grade";
break;
case 'C':
        result="C Grade";
```

```
    break;
    default:
            result="No Grade";
    }
    document.write(result);
    </script>
```

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

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.

```
for (initialization; condition; increment)
{
  Body of the loop
}
```

For example:

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

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.

```
while (condition)
{
   Body of the loop
}
```

Example:

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

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.

```
do{
        body of the loop
}while (condition);
```

Let's see the simple example of do while loop in javascript.

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

for-in loop

A for-in loop iterates through the properties of an object and executes the loop's body once for each enumerable property of the object. Here is an example:

```
<script type="text/javascript">
var student = { name:"Bill", age: 25, degree: "Masters" };
for (var item in student) {
    alert(student[item]);
}
</script>
```

With each iteration JavaScript assigns the name of the property (a string value) to the variable item. In the example above these are: name, age, and degree.

Following example prints the properties of the web browser's **Navigator** object.

```
<script type="text/javascript">
        var aProperty;
        document.write("Navigator Object Properties<br /> ");

        for (aProperty in navigator) {
    document.write(aProperty);
                document.write("<br />");
        }
        document.write ("Exiting from the loop!");
</script>
```

Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code. The syntax of function is given below.

```
function functionName([arg1, arg2, ...argN]){
        body of the function
}
```

Example

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

JavaScript Function Arguments

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

```
<html>
<head>
<script>
function area(a){
alert("area of the square is "+a*a);
}
</script>
</head>
<body>
<form>
<input type="button" value="click" onclick="area(4)"/>
</form>
</body>
</html>
```

Function with Return Value

We can call function that returns a value and use it in our program.

```
<script>
function getInfo(){
        return "hello.... 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)
```

arg1, arg2, ... , argn represents the argument used by function.

functionBody - It represents the function definition.

Example:

```
<script>
var add=new Function("num1","num2","return num1+num2");
        document.write(add(2,5));
</script>
```

Javascript 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. You can access the items in an array by referring to

its **indexnumber** and the index of the first element of an array is zero. There are 3 ways to construct array in JavaScript

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

1) JavaScript array literal

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

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

Example

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

2) JavaScript Array directly (new keyword)

The syntax of creating array using *new* is given below:

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

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

```
<script>
var i;
var emp = new Array();
emp[0] = "alice";
emp[1] = "kishor";
emp[2] = "John";
for (i=0;i<emp.length;i++){
        document.write(emp[i] + "<br>");
}
</script>
```

JavaScript Array Methods

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

Methods	Description

concat()	It returns a new array object that contains two or more merged arrays.
	<script>
	var arr1=["C","C++","Python"];
	var arr2=["Java","JavaScript","Android"];
	var result=arr1.concat(arr2);
	document.writeln(result);
	</script>

includes()	It checks whether the given array contains the specified element. It returns true if an array contains the element, otherwise false.
	<script>
	var arr=["AngularJS","Node.js","JQuery"]
	var result=arr.includes("AngularJS");
	document.writeln(result);
	</script>

indexOf()	It searches the specified element in the given array and returns the index of the first match.
	<script>
	var arr=["C","C++","Python","C++","Java"];
	var result= arr.indexOf("C++");
	document.writeln(result);
	</script>

lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.
	<script>
	var arr=["C","C++","Python","C++","Java"];
	var result= arr.lastIndexOf("C++");
	document.writeln(result);
	</script>

pop()	It removes the last element and returns the array.
	<script>
	var arr=["AngularJS","Node.js","JQuery"];
	document.writeln("Orginal array: "+arr+"
	");
	document.writeln("Extracted element: "+arr.pop()+"
	");
	document.writeln("Remaining elements: "+ arr);
	</script>

push()	It adds one or more elements to the end of an array.
	<script>
	var arr=["AngularJS","Node.js"];
	arr.push("JQuery");
	document.writeln(arr);
	</script>

reverse()	It reverses the elements of given array.
	<script>
	var arr=["AngulaJS","Node.js","JQuery"];
	var rev=arr.reverse();
	document.writeln(rev);
	</script>

shift()	It removes and returns the first element of an array.
	<script>
	var arr=["AngularJS","Node.js","JQuery"];
	var result=arr.shift();
	document.writeln(result,"
	");
	document.writeln(arr);
	</script>

sort()	It returns the element of the given array in a sorted order.
	<script>
	var arr=["AngularJS","Node.js","JQuery","Bootstrap"]
	var result=arr.sort();
	document.writeln(result);
	</script>

unshift()	It adds one or more elements in the beginning of the given array.
	<script>
	var arr=["AngularJS","Node.js"];
	var result=arr.unshift("JQuery");
	document.writeln(arr);
	</script>

JavaScript Events

JavaScript events are items that transpire based on an action. A document event is the loading of an HTML document. A form event is the clicking on a button. Events are objects *with* properties.JavaScript defines five types of events which are form, image, *image* map, *link*, and window events. Events are associated with HTML tags. The definitions of the events described are as follows:

Form Events

- blur - The input focus was lost.
- change - An element lost the focus since it was *changed.*
- focus - The input focus was obtained.
- reset - The user reset the object, usually a form.
- select-Some text is selected
- submit - The user submitted an object, usually a form.

Image Events

- abort - A user action caused an abort.
- error - An error occurred.
- load - The object was loaded.

Image Map Events

- mouseOut - The mouse is moved from on top a *link.*
- mouseOver - The mouse is moved over a link.

Link Events

- click - An object was clicked.
- mouseOut - The mouse is moved from on top a link.
- mouseOver - The mouse is moved over a link.

Window Events

- blur - The input focus was lost.
- error - An error occurred.
- focus - The input focus was obtained,
- load - The object was loaded.
- unload - The object was exited.

Event Association

Events are associated with HTML tags. The definitions of the events described below are as follows:

- abort - A user action caused an abort of an image or *document* load.
- blur - A frame set, document, or form object such as a text field loses the focus for input.
- click - Happens when a link or image map is clicked on
- change - Happens when a form field is changed by the user and it loses the focus.
- error - An error happened loading a image or document
- focus - A frame set, document, or form object such as a text field gets the focus for input.
- load - The event happens when an image or HTML page has completed the load process in the browser.
- mouseOut - The event happens when the mouse is moved *from* on top of a link or image map
- mouseOver - The event happens when the mouse is placed *on* a link or *image* map.
- reset - The user reset the object which is usually a form.
- submit - The user submitted an object which is usually a form,
- unload - The object such as a frameset or HTML document was exited by the user.

The events for each HTML tag are as follows:

- <A>
  - click (onClick)
  - mouseOver [onMouseOver)
  - mouseOut (onMouseOut)
- <BODY>
  - blur(onBlur)
  - error(onError)
  - focus (onFocus)
  - load (onLoad)
  - unload (onUnload)

- <FORM>
  - submit (onSubmit)
  - reset (onReset)
- <FRAME>
  - blur (onBlur)
  - focus (onFocus)

<FRAMESET>
- blur [onBlur]
- error [onError]
- focus [onFocus]
- load [onLoad]
- unload [onUnload]

<IMG>
- abort [onAbort]
- error [onError]
- load [onLoad]

<INPUT TYPE = “button”>
- click [onClick]

<INPUT TYPE = “checkbox^
- click (onClick)

<INPUT TYPE = “reset”>
- click (onClick)

<INPUT TYPE = “submit”>
- click (onClick)

<INPUT TYPE = “text”>
- blur (onBlur)
- focus (onFocus)
- change (onChange)
- select (onSelect)

<SELECT>

- blur (onBlur)
- focus (onFocus)
- change(onChange)

<TEXTAREA>

- blur (onBlur)
- focus (onFocus)
- change(onChange)
- select (onSelect)

Event Handlers

Event handlers are created as follows:

onEvent = "Code to handle the event"

Example

```
<html>
<head>
<script type="text/javascript">
function popup
{
alert("Hello World");
}
</script>
</head>
<body>
<input type="button" value="Click Me!" onclick="popup()><br/>
<a href="#" onmouseover="" onMouseaut-"popup()">Hover
Me!</a>
</body>
</html>
```

POP UP BOXES

It is possible to make three different kinds of popup windows. There are three types of pop up boxes.

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

ALERT BOX

The syntax for an alert box is:

```
alert("yourtext");
```

The user will need to click "OK" to proceed. Typical use is when you want to make sure information comes through to the user.

```
Alert("ok");
```

CONFIRM BOX:

The syntax for a confirm box is:

```
confirm("yourtext");
```

The user needs to click either "OK" or "Cancel" to proceed.Typical use is when you want the user to verify or accept something.

- If the user clicks "OK", the box returns the value **true**.
- If the user clicks "Cancel", the box returns the value **false**

Example:

```
if (confirm("Do you agree")) {
        alert("You agree");
}
else{
        alert ("You do not agree")
}
```

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 "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

prompt("*sometext*","*defaultText*");

```
var uname = prompt("Please enter your username", "abc");
```

JavaScript Built in Objects

JavaScript supports Object Oriented Programming. Objects have a set of properties or attributes and functions. If an attribute contains a function, it is considered to be a method of the object; otherwise the attribute is considered a property. JavaScript supports both user defined objects and built-in objects. All user-defined objects and built-in objects are descendants of an object called **Object**. The **new** operator is used to create an instance of an object. Following statement shows how to create user defined objects.

```
var objectname=new object();
For example:
var student=new object();
```

JavaScript provides a special constructor function called **Object()** to build the object.

```
<html>
<head>
<title>javascript objects</title>
<script type = "text/javascript">
    var student = new Object();  // Create an object
    student.rollnumber = 02;  // Assigning properties to the object
    student.name  = "Rahul";

    document.write("Roll Number : " + student.rollnumber + "<br>");
    document.write("Name : " + student.name + "<br>");
```

```
</script>
</head>

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

JavaScript supports a number of built-in objects that extend the flexibility of the language. These objects are Date, Math, String, Array, and Object. Several of these objects are "borrowed" from the Java language specification, but JavaScript's implementation of them is different. If you're familiar with Java, you'll want to carefully examine JavaScript's built-in object types to avoid any confusion.

The JavaScript object model is a simple one. In addition to window-content objects, JavaScript supports "built-in" objects. These built-in objects are available regardless of window content and operate independently of whatever page your browser has loaded.

The built-in objects are Number, Boolean, String, Date, Math, Array, and RegExp.

Number object

A **number object** holds primitive numeric values. The **Number** object represents numerical date, either integers or floating-point numbers. Syntax of creating nuber object is

```
var val = new Number(number);
```

if you are passing non-number argument in the place of number, then it cannot be converted into a number, it returns **NaN** (Not-a-Number).

Properties of Number Object

1. MAX_VALUE

   The largest possible value a number in JavaScript can have 1.7976931348623157E+308

For example:

```
var val = Number.MAX_VALUE;
document.write ("Value of Number.MAX_VALUE : " + val );
```

2. **MIN_VALUE**

The smallest possible value a number in JavaScript can have 5E-324

For example:

```
var val = Number.MIN_VALUE;
alert("Value of Number.MIN_VALUE : " + val );
```

3. **NaN**

Equal to a value that is not a number. it is usually used to indicate an error condition for a function that should return a valid number. **isNaN()** global function is used to check if the value is an NaN value.

For example:

```
var dayOfMonth = 50;
        if (dayOfMonth < 1 || dayOfMonth > 31) {
          dayOfMonth = Number.NaN
          alert("Day of Month must be between 1 and 31.")
        }
        Document.write("Value of dayOfMonth : " + dayOfMonth );
```

4. **NEGATIVE_INFINITY**

A value that is less than MIN_VALUE.

For example:

5. **POSITIVE_INFINITY**

A value that is greater than MAX_VALUE

For example:

6. **prototype**

A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document. Syntax is:

```
object.prototype.name = value
```

For example:

```
var student = new Object();
student.prototype.contact_no=1234567890;
```

7. **constructor**

Returns the function that created this object's instance. By default this is the Number object.

JavaScript Number Object Methods

1. **toString()**

The toString() method returns a number as a string.

```
<script>
var n = 100;
document.write(n.toString()+"<br>");
document.write((25 + "abc").toString());
</script>
```

Output

100

25abc

2. **toExponential()**

toExponential() returns a string, with a number rounded and written using exponential notation. The parameter is optional. If you are not specifying the parameter; then it will not round.

```
<script>
var n = 5.4563;
document.write( n.toExponential() + "<br>");
 document.write( n.toExponential(2) + "<br>");
 document.write( n.toExponential(4) + "<br>");
 document.write( n.toExponential(6));
</script>
```

Output

5.4563e+0

5.46e+0

5.4563e+0

5.456300e+0

3. **toFixed()**

toFixed() returns a string, with the number written with a specified number of decimals:

Example

```
var n = 5.556;
document.write( n.toFixed(0));
document.write( n.toFixed(2));
document.write( n.toFixed(4));
```

output

6
5.56
5.5560

4. **toPrecision()**

toPrecision() returns a string, with a number written with a specified length:

Example

```
var n = 5.556;
document.write( n.toPrecision());
document.write( n.toPrecision(2));
document.write( n.toPrecision(4));
document.write( n.toPrecision(6));
```

output

5.556
5.6
5.556
5.55600

5. **valueOf()**

valueOf() returns a number as a number. This method is used internally in JavaScript to convert Number objects to primitive values.

Example

```
var n = 125;
x.valueOf();
```

output

125

There are 3 different JavaScript methods that can be used to convert variables to numbers. That are:

1. The Number()
2. The parseInt()
3. The parseFloat()

1. Number()

Number() can be used to convert JavaScript variables to numbers:

Example

```
Number(true);
Number(false);
Number("125");
Number("125.55");
Number("125,89");
```

Output

1
0
125
125.55
NaN

2. parseInt()

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

```
parseInt("125");        // returns 125
parseInt("125.55");     // returns 125
parseInt("100 200 300"); // returns 100
parseInt("30 minutes"); // returns 30
parseInt("minutes 30"); // returns NaN
```

output

125
125
100
30
NaN

3. parseFloat()

parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

```
parseFloat("125");       // returns 125
parseFloat("125.55");    // returns 125.55
parseFloat("100 200 300"); // returns 100
parseFloat("30 minutes"); // returns 30
parseFloat("minutes 30"); // returns NaN
```

output

125
125.55
100
30
NaN

JavaScript Boolean Object

JavaScript includes Boolean object to represent true or false. It can be initialized using new keyword.

Example

```
var bool = new Boolean(true);
alert(bool); // true
```

JavaScript treats empty string (""), 0, undefined and null as false. Everything else is true.

Example:

```
var bool1 = new Boolean(“”); // false
var bool2 = new Boolean(0); // false
var bool3 = new Boolean(undefined); // false
var bool4 = new Boolean(null); // false
var bool5 = new Boolean(NaN); // false
var bool6 = new Boolean(“some text”); // true
var bool7 = new Boolean(1); // true
```

Boolean Methods

Primitive or Boolean object includes following methods.

Method	Description
toSource()	Returns a string which represents the source code of a boolean object.
toString()	Returns a string of Boolean. Example: var result = (1 > 2); result.toString(); // returns “false”
valueOf()	Returns the value of the Boolean object. Example: var result = (1 > 2); result.valueOf(); // returns false

JavaScript String object

Of all JavaScript’s objects, the String object is the most commonly used. In JavaScript implementation, new string objects are created implicitly using a variable assignment. For example,

```
var myString = “This is a string”;
```

creates a string, with the specified text, called myString. There is no actual object called string, and attempting to instantiate a new String object using the new statement results in an error, as String (or string) is not a defined keyword.

```
var myString = new String();
myString = “This is a string”;
```

and

```
var myString = new String (“This is a string”);
```

String objects have one property: length. The length property returns the length of the string and uses the syntax string.length, where string is the name of the string variable.

Both of the following display 16.

```
alert ("This is a string".length)
```

and

```
var myString = "This is a string";
alert (myString.length);
```

While there may be just one string property, JavaScript supports a large number of methods that can be used with strings.

JavaScript String Methods & Properties

JavaScript string (primitive or String object) includes default properties and methods which you can use for different purposes.

String Properties

Length is the property of string object. It returns the length or number of characters of a string.

String Methods

- **charAt(position)**

  Returns the character at the specified position (in Number).

- **charCodeAt(position)**

  Returns a number indicating the Unicode value of the character at the given position (in Number).

- **concat([string,,])**

  Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.

- **indexOf(SearchString, Position)**

  Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.

- **lastIndexOf(SearchString, Position)**

Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.

- **localeCompare(string,position)**

Compares two strings in the current locale.

- **match(RegExp)**

Search a string for a match using specified regular expression. Returns a matching array.

- **replace(searchValue, replaceValue)**

Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.

- **search(RegExp)**

Search for a match based on specified regular expression.

- **slice(startNumber, endNumber)**

Extracts a section of a string based on specified starting and ending index and returns a new string

- **split(separatorString, limitNumber)**

Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.

- **substr(start, length)**

Returns the characters in a string from specified starting position through the specified number of characters (length).

- **substring(start, end)**

Returns the characters in a string between start and end indexes.

- **toLocaleLowerCase()**

Converts a string to lower case according to current locale.

- **toLocaleUpperCase()**

  Converts a sting to upper case according to current locale.

- **toLowerCase()**

  Returns lower case string value.

- **toString()**

  Returns the value of String object.

- **toUpperCase()**

  Returns upper case string value.

- **valueOf()**

  Returns the primitive value of the specified string object.

Math object

JavaScript's Math object provides advanced arithmetic and trigonometric functions, expanding on JavaScript's basic arithmetic operators (plus, minus, multiply, divide). The Math object in JavaScript is borrowed from Java. In fact, the implementation of the Math object in JavaScript closely parallels the Math class in Java, except that the JavaScript Math object offers fewer methods.

The Math methods are used in mathematical and trigonometric calculations. Handy Math-object methods include ceil, floor, pow, exp (exponent), max, min, round, and random. (Random is only available when using the X Window platform, however.)

The Math object is static, so you don't need to create a new Math object in order to use it. To access the properties and method of the Math object, you merely specify the Math object, along with the method or property you wish.

For example, to return the value of *pi*, you use:

```
var pi = Math.PI;
```

Similarly, to use a math method you provide the name of the method, along with the parameters you wish to use. For example, to round the value of *pi*, you'd use:

```
var pi = Math.PI;
var pieAreRound = Math.round(pi);      // it displays 3
```

Note that you must specify the Math object by name for each Math method/property you wish to use. JavaScript does not recognize the keywords PI and round all by themselves. Exception: you may use the with statement to associate the names of methods and properties with the Math object. This technique is a handy space-saver when you must use several Math properties and methods. The previous example can be written as

```
with (Math) {
    var pi = PI;
    var pieAreRound = round(pi);
    alert (pieAreRound)
}
```

Properties are used by *math.properyname*. Following are the different properties of and methods of *math* object.

- 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 square root of 1/2 (approx. 0.707)
- SQRT2 - Returns the square root of 2 (approx. 1.414)

Math object methods

- abs( ) - Returns the absolute value of a number
- acos( ) - Returns the arc cosine of a number
- asin( ) - Returns the arc sine of a number
- atan( ) - Returns the arc tangent of a number
- ceil( ) - Returns the least integer greater than or equal to a number
- cos( ) - Returns the cosine of a number

- exp( ) - Returns e (Euler's constant) to the power of a number
- floor( ) - Returns the greatest integer less than or equal to its argument
- log( ) - Returns the natural logarithm (base e) of a number
- max( ) - Returns the greater of two values
- min( ) - Returns the lesser of two values
- pow( ) - Returns the value of a number times a specified power
- random( ) - Returns a random number (X-platforms only)
- round( ) - Returns a number rounded to the nearest whole value
- sin( ) - Returns the sine of a number
- sqrt( ) - Returns the square root of a number
- tan( ) - Returns the tangent of a number

Date object

JavaScript Date object is used to determine the current time and date. A popular JavaScript application of the Date object is displaying a digital clock in a text box. The script uses the Date object to update the clock once every second. You also use the Date object to perform date math. For example, your script might determine the number of days between now and a certain future date. You can use this to display a "countdown," such as the number of days left of your company's big sale.

JavaScript treats the Date object like a constructor class. To use Date you must create a new Date object; you can then apply the various Date methods to get and set dates. (*The Date object has no properties.*) If you're familiar with the Date class in Java, you'll find the properties of the JavaScript Date object largely the same. The most commonly used methods are the get methods, which obtain the time and date of the value in the Date object. These methods are: getHours(), getMinutes(), getSeconds(), getYear(), getMonth(), getDate(), getDay().

(JavaScript's Date object also provides for setting the time and date of the Date object, but these are seldom used.)

Constructing a new Date object can take several forms. To return an object containing the current date and time, you use the Date object

without parameters. In the following, date_obj is a new object, containing the value of the current date and time, as set by the computer's system clock.

```
var date_obj = new Date();
```

Alternatively, you can specify a given date and time as part of the date constructor. Either of these methods is allowed — both set the new date object to January 1, 1997, at midnight local time.

```
var date_obj = new Date ("January 1 1997 00:00:00")
```

and

```
var date_obj = new Date (97, 0, 1, 12, 0, 0)
```

To use a Date method, append the method to the date object you previously created. For example, to return the current year, use:

```
var now = new Date();
var yearNow = now.getYear();
```

JavaScript Date Object Methods

- **getDate()** - Returns the day of month of a specified date
- **getDay()** - Returns the day of week of a specified date
- **getHours()** - Returns the hour of a specified date
- **getMinutes()** - Returns the minutes of a specified date
- **getMonth()** - Returns the month of a specified date
- **getSeconds()** - Returns the seconds of a specified date
- **getTime()** - Returns the number of seconds between January 1, 1970 and specified date
- **getTimeZoneoffset()** - Returns the time zone offset in minutes for the current locale
- **getYear()** - Returns the year of specified date
- **Parse()** - Returns the number of milliseconds in a data since January 1, 1970, 00:00:00
- **setDate()** - Sets the date

- **setHours()** - Sets the hours of a specified date
- **setMinutes()** - Sets the minutes of a specified date
- **setMonth()** - Sets the month of a specified date
- **setSeconds()** - Sets the seconds of a specified date
- **setTime()** - Sets the time of a specified date
- **setYear()** - Sets the year of a specified date
- **toGMTString()** - Converts a date to a string using GMT conventions
- **toLocaleString()** - Converts a date to a string using locale conventions
- **toString()** - Converts the value of a Date object or current location object to a string
- **UTC()** - Converts a comma-delimited date to the number of seconds since Jan 1, 1970.

Array Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Use the following syntax to create an **Array** object "

var fruits = new Array("apple", "orange", "mango");

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows "

var fruits = ["apple", "orange", "mango"];

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element

Array Properties

- **constructor**

  Returns a reference to the array function that created the object.

- **Index**

  The property represents the zero-based index of the match in the string

- **Input**

  This property is only present in arrays created by regular expression matches.

- **length**

  Reflects the number of elements in an array.

- **prototype**

  The prototype property allows you to add properties and methods to an object.

Array Methods

- **concat()**

  Returns a new array comprised of this array joined with other array(s) and/or value(s).

- **every()**

  Returns true if every element in this array satisfies the provided testing function.

- **filter()**

  Creates a new array with all of the elements of this array for which the provided filtering function returns true.

- **forEach()**

  Calls a function for each element in the array.

- **indexOf()**

  Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

- **join()**

  Joins all elements of an array into a string.

- **lastIndexOf()**

  Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

- **map()**

  Creates a new array with the results of calling a provided function on every element in this array.

- **pop()**

  Removes the last element from an array and returns that element.

- **push()**

  Adds one or more elements to the end of an array and returns the new length of the array.

- **reduce()**

  Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

- **reduceRight()**

  Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

- **reverse()**

  Reverses the order of the elements of an array — the first becomes the last, and the last becomes the first.

- **shift()**

  Removes the first element from an array and returns that element

- **some()**

  Returns true if at least one element in this array satisfies the provided testing function.

- **toSource()**
  Represents the source code of an object
- **sort()**
  Sorts the elements of an array
- **splice()**
  Adds and/or removes elements from an array.
- **toString()**
  Returns a string representing the array and its elements.
- **unshift()**
  Adds one or more elements to the front of an array and returns the new length of the array.

RegExp object

In JavaScript, a regular expression is simply a type of object that is used to match character combinations in strings. Or a regular expression is a sequence of characters that forms a search pattern. The search pattern can be used for text search and text replace operations.

There are two ways to create a regular expression:

If you expect your regular expression to remain constant (unchanging), it is best to use a **regex literal**. If your regular expression is dynamic, it is better to use the **regex constructor** method.

- **Regular Expression Literal** — This method uses slashes (/) to enclose the Regex pattern:

```
var regexLiteral = /one/;
```

- **Regular Expression Constructor** — This method constructs the Expression for you:

```
var regexConstructor = new RegExp("one");
```

A regular expression could be defined with the **RegExp ()** constructor, as follows "

```
var pattern = new RegExp(pattern, attributes);
or
var pattern = /pattern/attributes;
```

Here is the description of the parameters "

- **pattern** " A string that specifies the pattern of the regular expression or another regular expression.
- **attributes** " An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multi-line matches, respectively.

Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Brackets

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s

n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

RegExp Object Properties

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set
source	Returns the text of the RegExp pattern

RegExp object methods

test() method

One basic method is .test(), which returns a Boolean that is it searches a string for a pattern, and returns true or false, depending on the result. Syntax is:

RegExpobj.test()

- Returns true: the string contains a match of the regex pattern
- Returns false: no match found

```
str1 = "hello how are you";
str2 = "how are you";
ptrn = /hello/;
ptrn.test(str1); // it returns true because hello is in str1
ptrn.test(str2); // it returns false because hello is not in str1
```

exec()

The exec() method is a RegExp expression method. It searches a string for a specified pattern, and returns the found text as an object. If no match is found, it returns an empty *(null)* object.

```
<script>
var str="hello how are you";
var obj = /o/.exec(str);
document.write(obj);
</script>
```

It returns *o* because str contains *o*.

toString()

The toString() method returns the string value of the regular expression. The syntax of using toString() is:

RegExpObject.toString()

For Example:

```
<script>
function myFunction() {
 var patt = new RegExp("Hello World", "g");
 var res = patt.toString();
 document.getElementById("demo").innerHTML = res;
}
</script>
```

Output

/Hello World/g

*Note that we can also use some string functions for regular expressions(replace(), serach(), etc.).

Form Validation

Form validation is the process of making sure that data supplied by the user using a form, meets the criteria set for collecting data from the user. For 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 (in JavaScript or in any other language) to check whether the user entered a name containing alphabets only, a valid email address and a proper address.

Forms are used in webpages for the user to enter their required details that are further send it to the server for processing. A form is also known as web form or HTML form. Everyone must have filled an online form at some stage, a form usually asks for information related to name, phone no, address, credit-card no etc.

Incase you didn't provide information in the format specified by the form field or leave it empty, the message will appear and form cannot be submitted until you get it right.

This is done using a Javascript program on the client side, the Javascript program uses a **regular expression** pattern to test the input of each form field.

Accessing form data

If an HTML document contains more than one forms, they can be accessed as either by document.form_name where form_name is the value of the name attribute of the form element or by document.forms[i] where i is 0, 1,2,3.... and document.forms[0] refers to the first form of the document, document.forms[1] refers to the second form of the document and so on.

Elements of a form can be accessed by document.form_name.form_element where form_name is the value of the name attribute of the form element, form_element is the value of the name attribute of the form's element.

Following examples shows the different form validation using regular expression.

- **Checking for non-empty**

The following function can be used to check whether the user has entered anything in a given field. Blank fields indicate two kinds of values. A zero-length string or a NULL value.

```
function required()
{
var empt = document.forms["form1"]["text1"].value;
if (empt == "")
{
alert("Please input a Value");
return false;
}
else
{
alert('Code has accepted : you can try another');
return true;
}
}
```

- **Checking for all letters**

```
function allLetter(inputtxt)
 {
 var letters = /^[A-Za-z]+$/;
 if(inputtxt.value.match(letters))
  {
   return true;
  }
 else
  {
  alert("message");
  return false;
  }
 }
```

To get a string contains only letters (both uppercase or lowercase) we use a regular expression (/^[A-Za-z]+$/) which allows only letters. Next the match() method of string object is used to match the said regular expression against the input value.

- **Checking for all numbers**

To get a string contains only numbers (0-9) we use a regular expression (/^[0-9]+$/) which allows only numbers. Next, the match() method of the string object is used to match the said regular expression against the input value.

```
function allnumeric(inputtxt)
 {
   var numbers = /^[0-9]+$/;
   if(inputtxt.value.match(numbers))
   {
   alert('Your Registration number has accepted....');
   document.form1.text1.focus();
   return true;
   }
   else
   {
   alert('Please input numeric characters only');
   document.form1.text1.focus();
   return false;
   }
 }
```

- **Email Validation**

```
function ValidateEmail(mail)
{
 if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
.test(myForm.emailAddr.value))
 {
  return (true)
 }
  alert("You have entered an invalid email address!")
  return (false)
}
```

- **Phone No. Validation**

we validate a phone number of 10 digits with no comma, no spaces, no punctuation and there will be no + sign in front the number. Simply the validation will remove all non-digits and permit only phone numbers with 10 digits. Here is the function.

```
function phonenumber(inputtxt)
{
 var phoneno = /^\d{10}$/;
 if((inputtxt.value.match(phoneno))
    {
   return true;
    }
   else
    {
    alert("message");
    return false;
    }
}
```

A simple registration form validation

Example:

```
<body>
<h1 style="text-align: center"> REGISTRATION FORM </h1>
<form name="RegForm" action="/submit.php" onsubmit="return
GEEKFORGEEKS()" method="post">
  <p>Name: <input type="text" size=65 name="Name"></p><br>
  <p> Address: <input type="text" size=65 name="Address"> </p><br>
  <p>E-mail Address: <input type="text" size=65 name="EMail"> </p><br>
  <p>Password: <input type="text" size=65 name="Password"></p><br>
  <p>Telephone: <input type="text" size=65 name="Telephone"></p><br>
```

```
<p>SELECT YOUR COURSE
  <select type="text" value="" name="Subject">
    <option>BTECH</option>
    <option>BBA</option>
    <option>BCA</option>
    <option>B.COM</option>
    <option>GEEKFORGEEKS</option>
  </select></p><br><br>
<p>Comments: <textarea cols="55" name="Comment"> </textarea></p>
<p><input type="submit" value="send" name="Submit">
  <input type="reset" value="Reset" name="Reset">
</p>
</form>
</body>
```

Validating a form :

The data entered into a form needs to be in the right format and certain fields need to be filled in order to effectively use the submitted form. Username, password, contact information are some details that are mandatory in forms and thus need to be provided by the user.

Here JavaScript is used for form validation:

```
<script>
function GEEKFORGEEKS()
{
  var name = document.forms["RegForm"]["Name"];
  var email = document.forms["RegForm"]["EMail"];
  var phone = document.forms["RegForm"]["Telephone"];
  var what =  document.forms["RegForm"]["Subject"];
  var password = document.forms["RegForm"]["Password"];
  var address = document.forms["RegForm"]["Address"];
  if (name.value == "")
  {
```

```
    window.alert("Please enter your name.");
    name.focus();
    return false;
  }
  if (address.value == "")
  {
    window.alert("Please enter your address.");
    name.focus();
    return false;
  }
  if (email.value == "")
  {
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
  }
  if (email.value.indexOf("@", 0) < 0)
  {
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
  }
  if (email.value.indexOf(".", 0) < 0)
  {
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
  }

  if (phone.value == "")
  {
```

```
        window.alert("Please enter your telephone number.");
        phone.focus();
        return false;
    }

    if (password.value == "")
    {
        window.alert("Please enter your password");
        password.focus();
        return flase;
    }

    if (what.selectedIndex < 1)
    {
        alert("Please enter your course.");
        what.focus();
        return false;
    }

    return true;
}</script>
```

REVIEW QUESTIONS

Part A

1. What is the use of CSS?
2. Explain the syntax of defining CSS.
3. How to add comments in CSS?
4. What is javascript?
5. What are different javascript data types
6. What is the difference between for and for-in loop?
7. Define function in JavaScript.
8. What events in JavaScript.
9. What is regular expression?

Part B

1. Explain different types of CSS selectors?
2. Explain the different ways to adding CSS to HTML document
3. What are <link> and <style> elements?
4. What are pseudo class selectors?
5. What are the different ways to adding JavaScript to HTML document?
6. Explain the different decision control statements in JavaScript.
7. Explain the syntax of switch statement with an example.
8. Explain functions in JavaScript.
9. Explain arrays in JavaScript.
10. What are the different popup boxes in JavaScript?
11. Explain form validation with an example.
12. Explain any four string methods in JavaScript.

Part C

1. Explain different types of CSS properties.
2. Describe are the different types of operators in javascript.
3. Explain loops in JavaScript with examples.
4. Explain different JavaScript built-in objects.
5. Validate an e-mail registration from using JavaScript.