SUBJECT CODE : 3161611

As per New Syllabus of

GUJARAT TECHNOLOGICAL UNIVERSITY

Semester - VI (IT) Professional Elective - III

ADVANCED WEB PROGRAMMING

Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in

P.E.S. Modern College of Engineering, Pune

TECHNICAL PUBLICATIONS

SINCE 1993

An Up-Thrust for Knowledge

ADVANCED WEB PROGRAMMING

Subject Code : 3161611

Semester - VI (Information Technology) Professional Elective - III

Published by :

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :
Yogiraj Printers & Binders
Sr. No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-90770-24-3

Course 18

PREFACE

The importance of **Advanced Web Programming** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Advanced Web Programming**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
A. A. Puntambekar

Dedicated to God.

SYLLABUS

Advanced Web Programming - 3161611

Credits	Examination Marks				Total Marks
C	Theory Marks		Practical Marks		
	ESE (E)	PA (M)	ESE (V)	PA (I)	
4	70	30	30	20	150

1. Refreshing Java Script and CSS

CSS syntax, benefits, Responsive design, Bootstrap introduction, Java script syntax, Java script inbuilt objects, Error handling and event handling, DOM, Asynchronous Programming. **(Chapter - 1)**

2. Introduction to Angular JS

Basics and Syntax of Angular JS, Features, Advantages, Application Structure, Basics of routes and navigation, MVC with Angular JS, Services. **(Chapter - 2)**

3. Angular JS in Details

Modules, Directives, Routes, Angular JS Forms and Validations, Data binding, Creating single page website using Angular JS. **(Chapter - 3)**

4. Introduction to Node JS

Setup Node JS Environment, Package Manager, Features, Console Object, Concept of Callbacks. **(Chapter - 4)**

5. Node JS in details

Events and Event Loop, timers, Error Handling, Buffers, Streams, Work with File System, Networking with Node (TCP, UDP and HTTP clients and servers), Web Module, Debugging, Node JS REST API, Sessions and Cookies, Design patterns, caching, scalability. **(Chapter - 5)**

6. Database Programming with Node JS and MongoDB

Basics of MongoDB, Data types, Connect Node JS with MongoDB, Operations on data (Insert, Find, Query, Sort, Delete, Update) using Node JS. **(Chapter - 6)**

TABLE OF CONTENTS

Part II : JavaScript

Notes

1 Refreshing Java Script and CSS

Syllabus

CSS syntax, benefits, Responsive design, Bootstrap introduction, Java script syntax, Java script inbuilt objects, Error handling and event handling, DOM, Asynchronous Programming.

Contents

Part I : CSS and Bootstrap

1.1 Introduction to CSS

- The CSS stands for **Cascading Style Sheet.**
- The Cascading Style Sheet is a markup language used in the web document for **presentation purpose.**
- The primary intension of CSS was to separate out the web content from the web presentation.
- Various elements such as text, font and color are used in CSS for presentation purpose. Thus CSS specification can be applied to bring the styles in the web document.

1.1.1 Benefits of CSS

Following are important benefits of CSS :

(1) **Improved Control Over Formatting :** The web page developer (called as web author) have better control over formatting of web contents.

(2) **Improved Site Maintainability :** It becomes easy to maintain the site because the formatting is centralized into one CSS file.

(3) **Improved Accessibility :** The CSS driven sites are more accessible and give the user significantly enriched experience of accessibility of the web page.

(4) **Improved Download Speed :** As all the formatting is centralized into one CSS file, all presentation will be quicker to download.

(5) **Output Flexibility :** CSS can be used to ~adopt a page for different output media. This approach to CSS page design is often referred to as responsive design.

Review Question

1. *What is CSS? Enlist the benefits of CSS.*

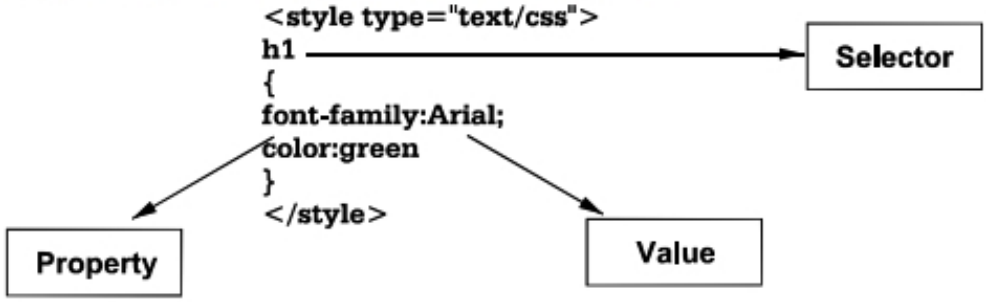
1.2 CSS Syntax and Structure

The style specification is specified differently for each different level. For instance :

- For inline cascading style sheets the **style** appears in side the tag for defining the value.

```
<p style="font-size: 30pt ;font-family:Script">
```

- For the document cascading style sheet the **style** specification appear as the content of a style element within the header of a document.



The **type** attribute tells the browser that what it is reading is text which is affected by the Cascading Style Sheet. The type specification is necessary because there is one more specification used in JavaScript.

- The External style sheet makes use of **style** specification in the same manner as in document cascading style sheet.

The most commonly used CSS properties are enlisted in the following table –

Property Type	Property
Fonts	font
	font-family
	font-size
	font-style
	font-weight
	@font-face
Text	letter-spacing
	line-height
	text-align
	text-decoration
	text-indent
Color and background	background
	background-color
	background-image
	background-position
	background-repeat
	color

Borders	border border-color border-width border-style border-top border-top-color border-top-width
Spacing	padding padding-bottom, padding-left, padding-right, padding-top margin margin-bottom, margin-left, margin-right, margin-top
Sizing	height max-height max-width min-height min-width width
Layout	bottom, left, right, top clear display float overflow position visibility z-index

Lists	list-style
	list-style-image
	list-style-type

1.3 Location of Styles

There are three levels of cascading style sheets -

- Inline style sheet
- Document level style sheet
- External level style sheet

1.3.1 Inline Style Sheet

- The inline cascading style sheet is a kind of style sheet in which the styles can be applied to HTML tags. This tag can be applied using following rule -

```
Tag
{
property: value
}
```

- For example :

```
<p style="font-family: Arial;color:red" >
```

Here for the tag **p** two properties are used such as **font-family** and **color** and those are associated with the values such as Arial and red respectively.

- Note that if we want to use **more than one property** then we have to use separator such as **semicolon.** In the following HTML document we have used cascading style sheet-

HTML Document [InlineStyle.html]

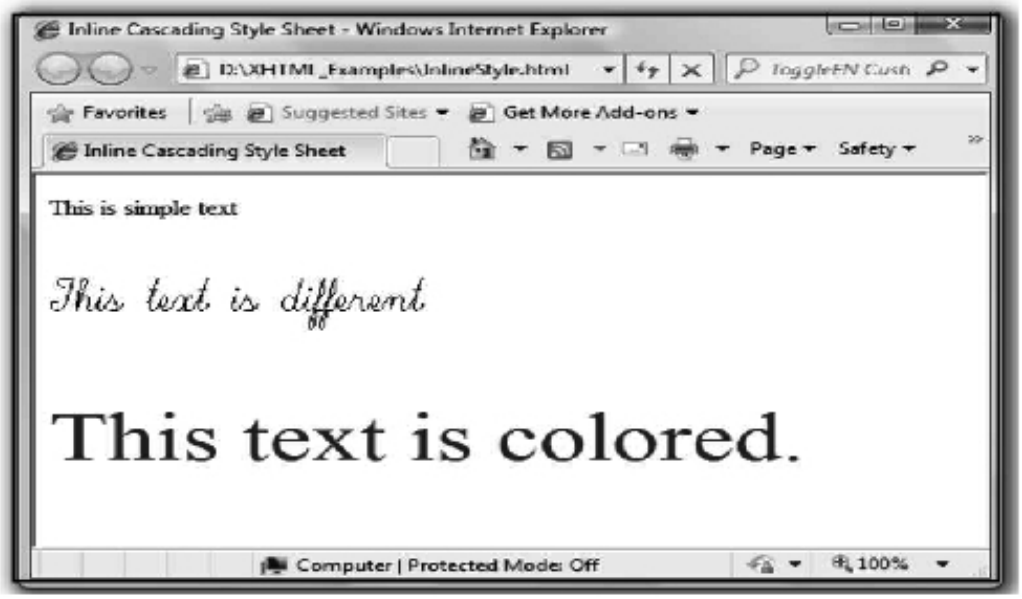
```
<!DOCTYPE html >
<html>
    <head>
        <title>Inline Cascading Style Sheet</title>
    </head>
    <body>
        <p>This is simple text</p>
        <p style="font-size: 30pt ;font-family:Script">This text is different </p>
        <p style="font-size: 40pt ;color:#ff0000">This text is colored.</p>
    </body>
</html>
```

Script Explanation

1) In this document, in the body section the style sheets are created.
2) In this section first of all we have displayed a simple sentence This is simple text. There is no style for this sentence.
3) In the next line, we have applied style in which **font-size** is set to the size of 30 point and **font-family** is set by the font name "Script" .
4) Then colored text will be displayed by **color:#ff0000**". The first two digits ff denote the red color, there is no green and blue color as the values next to ff are 00. Hence the text will be displayed in red.

This can be very well illustrated in output.

Output



1.3.2 Document Level Style Sheet

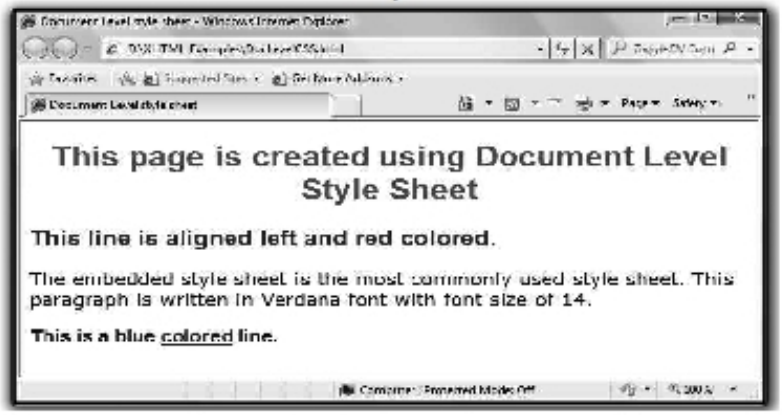
- This type style sheet appears only in the head section and in the body section newly defined Selector tags are used with the actual contents.
- For example : In the following HTML script we have defined **h1, h2, h3** and **p** selectors. For each of these **selectors** different property and values are set. Such setting will help us to represent our web page in some decorative form.
- The most important thing while writing document level style sheet is that we should mention the **style type="text/css"** in the head section. By this the browser will come to know that the program is making use of cascading style sheet.

HTML Document [DocLevelCSS.html]

```
<!DOCTYPE html>
<html >
    <head>
    <title>Document Level style sheet</title>
    <style type="text/css">
        h1
        {
         font-family:Arial;
         color:green
        }
        h2
        {
         font-family:Arial;
         color:red;
         left:20px
        }
        h3
        {
         font-family:arial;
         color:blue;
        }
        p
        {
         font-size:14pt;
         font-family:verdana
        }
    </style>
    </head>
    <body>
        <h1>
          <center>
          This page is created using Document Level Style Sheet
          </center>
         </h1>
        <h2>
          This line is aligned left and red colored.
        </h2>
        <p>
          The embedded style sheet is the most commonly used style sheet.
          This paragraph is written in Verdana font with font size of 14.
        </p>
        <h3>
        This is a blue <a href="colorname.html">colored</a> line.
```

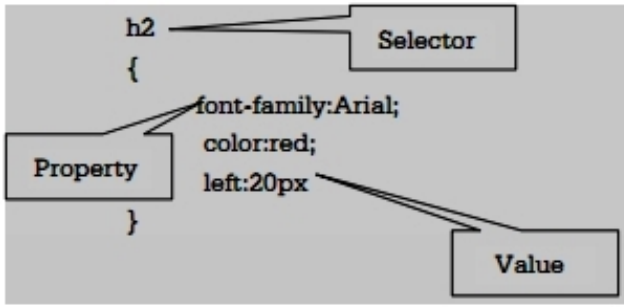
```
    </h3>
</body>
</html>
```

Output



Script Explanation

1) In above program, we have defined all the selectors in the head sections only. And these HTML elements are then used along with the web page contents.
2) The setting defined in the selectors will affect the web page contents. For example we have defined the selector h2 as



and then in the body section we have written -

```
<h2> This line is aligned left and red colored.</h2>
```

3) Now as h2 defines font to be "Arial" with color as "red" having left alignment of 20 pixels, the sentence "This line is aligned left and red colored." will be displayed in Arial font, which is red colored and aligned from left by 20 pixels. Surely we can see this effect on our web browser.

1.3.3 External Stylesheet

- Sometimes we need to apply particular style to more than one web documents in such cases external style sheets can be used.

- The central idea in this type of style sheet is that the desired style is stored in one .css file. And the name of that file has to be mentioned in our web pages.
- Then the styles defined in .css file will be applied to all these web pages.
- Here is a sample program in which external style sheet is used.

Step 1 : Create an HTML document

HTML Document[ExtCSS.html]

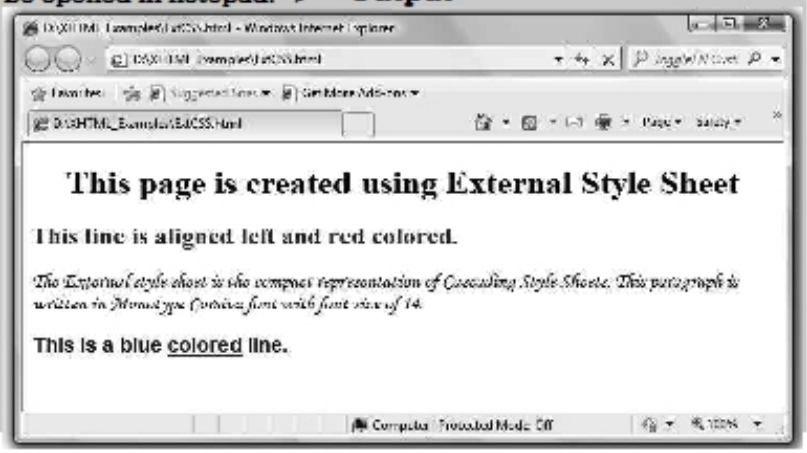
```
<!DOCTYPE html>
<html >
    <head>
    <link rel="stylesheet" type="text/css" href="ex1.css" />
    </head>
    <body>
<h1 class="special"> <center> This page is created using External Style
Sheet</center>  </h1>
        <h2>
          This line is aligned left and red colored.
        </h2>
        <p>
        The External style sheet is the compact representation of Cascading Style Sheets.
        This paragraph is written in Monotype Corsiva font with font size of 14.
        </p>
        <h3>
          This is a blue <a href="colorname.html">colored</a> line.
        </h3>
    </body>
</html>
```

Step 2 : Create a css file which contains the styles that can be applied to different HTML elements present in the above HTML document.

The cascading style sheet ex1.css can be

```
<!- - The file name ex1.css and can be opened in notepad.-->
h1
{
 font-family:Arial
}
h2
{
 font-family:times new roman;
 color:red;
left:20px
}
h3
```

Output



```
{
font-family:arial;
color:blue;
}
p
{
 font-size:14pt;
 font-family:Monotype Corsiva
}
```

Script Explanation

When we want to link the external style sheet then we have to use **<link>** tag which is to be written in the head section.

- **link** tells the browser some file must be linked to the page.
- **rel=stylesheet** tells the browser that this linked thing a style sheet.
- **href="** " denotes the path name of style sheet file.
- **type="text/css"** tells the browser that what it is reading is text which is affected by the cascading style sheet.

1.4 Selectors

1.4.1 Simple Selector Form

- The simple selector form is a single element to which the property and value is applied.
- We can also apply property and value to group of elements.
- Following is an illustration for simple selector form.

HTML Document[SimpleSel.html]

```
<!DOCTYPE html>
<html >
    <head>
    <title>Simple Selector Form</title>
    <style type="text/css">
        h1
        {
         font-family:Arial;
         color:green;
        }
```

```
        h2,h3
        {
         font-family:Monotype Corsiva;
         color:red;
         font-size: 28pt;
        }
    </style>
    </head>
    <body>
        <h1>India is My Country</h1>
        <h2>All the Indians are my brothers and sisters</h3>
        <h3>I love my country</h3>
    </body>
</html>
```

We can apply style to more than one selector

Output

Similarly the style can also be applied to the elements at specific positions.

For example :

```
body b p {font-size:18pt;}
```

Note that there are more than one element to which the style is applied and these elements are separated by the white spaces.

1.4.2 Class Selectors

- Using class selector we can assign different styles to the same element.
- These different styles appear on different occurrences of that element.

For example

HTML Document[ClassSel.html]

```
<!DOCTYPE html>
<html>
<head>
<title>Class Selector Form</title>
<style type="text/css">
    h1.RedText
    {
     font-family:Monotype Corsiva;
     color:red;
     font-size: 14pt;
    }
    h1.BlueText
    {
     font-family:Arial;
     color:blue;
     font-size: 10pt;
    }
</style>
</head>
<body>
    <h1 class ="RedText">India is My Country</h1>
    <h1 class="BlueText">All the Indians are my brothers and sisters</h1>
    <h3>I love my country</h3>
</body>
</html>
```

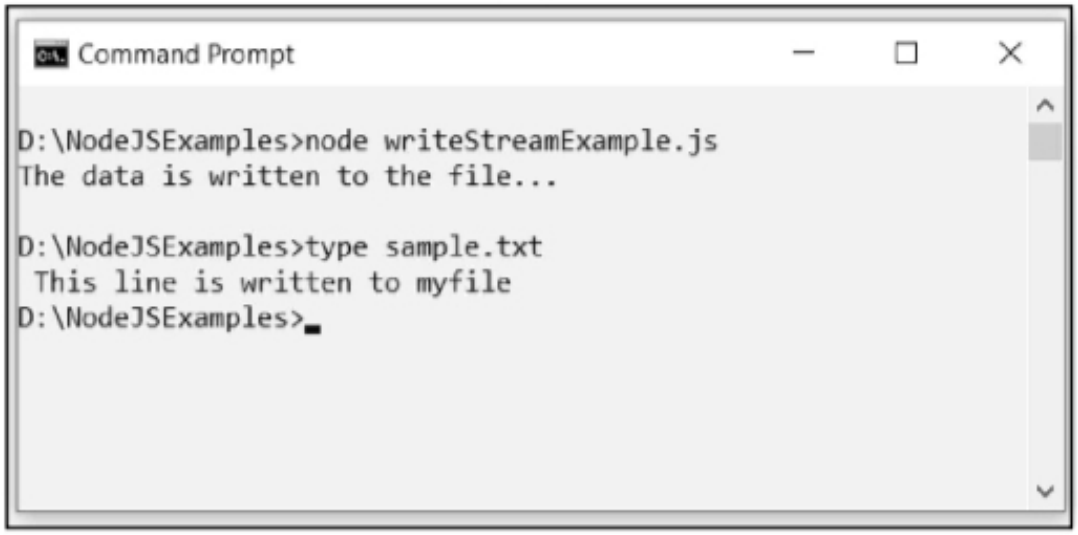
Output

Following Nodejs script shows how to write data to the string,

writeStreamExample.js

```
var fs = require("fs");
var str = " This line is written to myfile";
var ws = fs.createWriteStream('sample.txt');
ws.write(str,'utf8');
ws.end();
console.log("The data is written to the file...");
```

Output



The above code is simply modified to handle the error event as follows -

```
var fs = require("fs");
var str = " This line is written to myfile";
var ws = fs.createWriteStream('sample.txt');
ws.write(str,'utf8');
ws.end();
console.log("The data is written to the file...");

ws.on('error',function(err) {
        console.log(err.stack);
});
```

Read Operation

For reading from the stream we use createReadStream. Then using the **data** event the data can be read in chunks.

```
</tr>
<tr class="three"><td>PQR</td><td>5</td>
</tr>

</body>
</html>
```

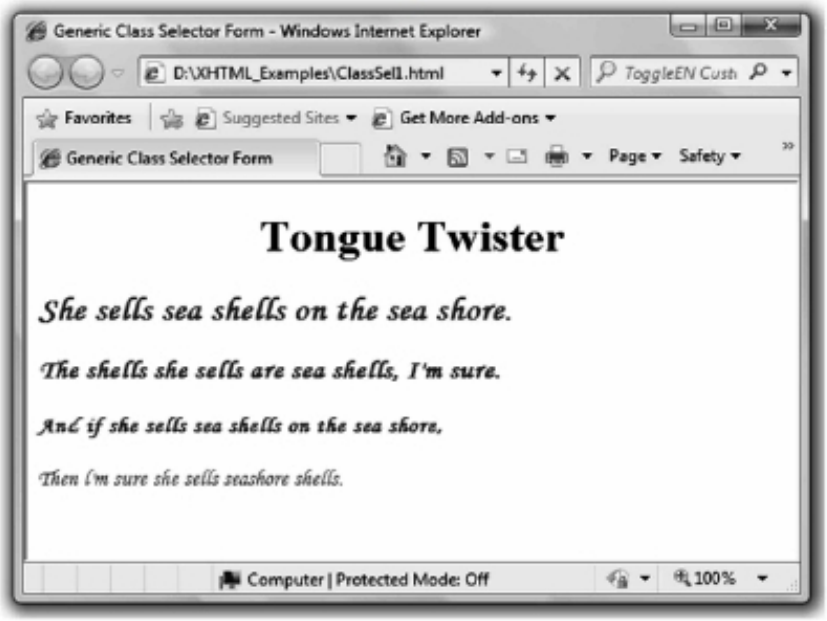
1.4.3 Generic Selectors

- We define the class in generalised form.
- In the sense, that particular class can be applied to any tag.
- Here is the HTML document which makes use of such generic selector.

HTML Document [ClassSel1.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Generic Class Selector Form</title>
    <style type="text/css">
        .RedText
        {
         font-family:Monotype Corsiva;
         color:red;
        }
        </style>

    </head>
    <body>
    <center>
    <h1> Tongue Twister</h1>
    </center>
        <h2 class="RedText">
        She sells sea shells on the sea shore.
    </h2>
    <h3 class="RedText">
        The shells she sells are sea shells, I'm sure.
    </h3>
    <h4 class="RedText">
        And if she sells sea shells on the sea shore,
        </h4>
    <p class="RedText">
    Then I'm sure she sells seashore shells.
    </p>
    </body>
</html>
```

Output



Note that the class selector must be preceded by a dot operator.

1.4.4 Id Selectors

- The **id selector** is similar to the **class selector** but the only **difference between the two** is that class selector can be applied to more than one elements where as using the id selector the style can be applied to the one specific element.
- The syntax of using id selector is as follows -

```
#name_of_id {property:value list;}
```

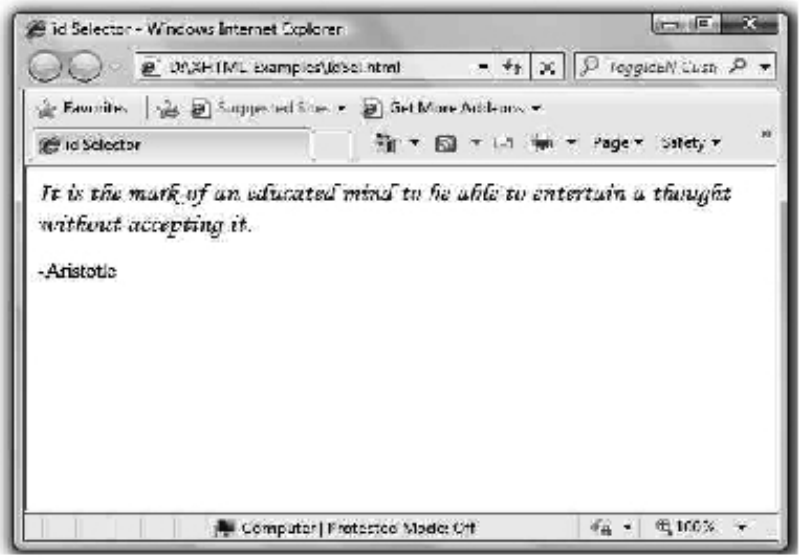
- Following HTML document makes use of id selector

HTML Document[IdSel.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>id Selector</title>
    <style type="text/css">
        #top
        {
         font-family:Monotype Corsiva;
         color:blue;
         font-size:16pt;
        }
        </style>
    </head>
    <body>
```

```
    <div id="top">
        It is the mark of an educated mind to be able to
        entertain a thought without accepting it.
    </div>
    <p>
        -Aristotle
    </p>
    </body>
</html>
```

Output



1.4.5 Universal Selectors

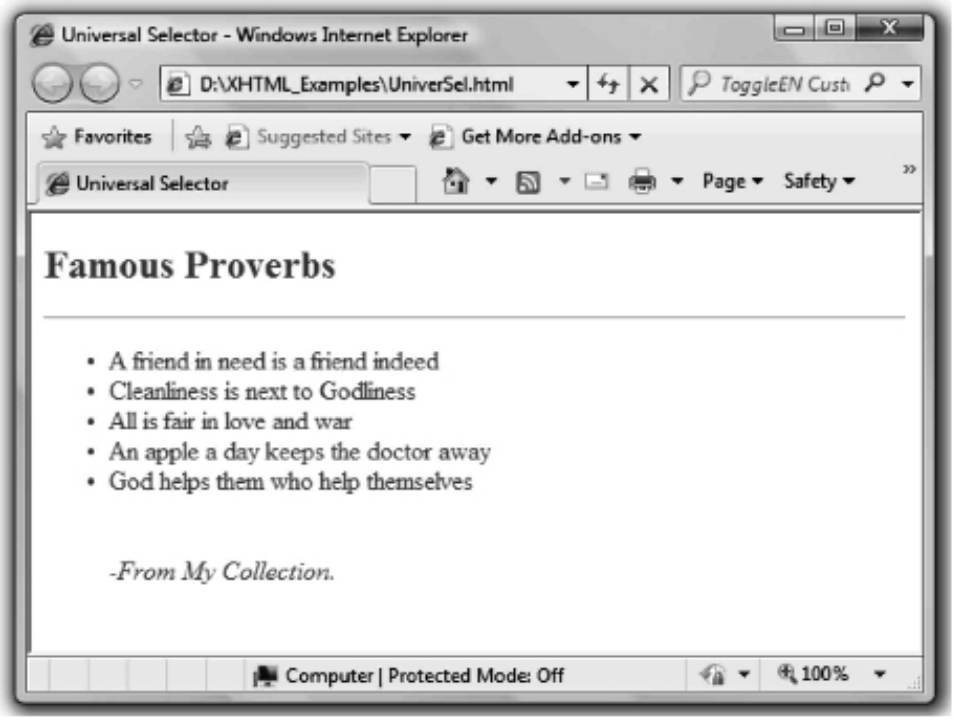
This selector is denoted by * (asterisk). This selector can be applied to all the elements in the document.

HTML Document[UniverSel.html]

```
<!DOCTYPE html>
<html >
    <head>
    <title>Universal Selector</title>
    <style type="text/css">
        * {
         color:green;
          }
        </style>
    </head>
<body>
    <h2> Famous Proverbs</h2>
```

```
    <hr/>
    <ul type="disc">
    <li>A friend in need is a friend indeed</li>
    <li>Cleanliness is next to Godliness</li>
    <li>All is fair in love and war</li>
    <li>An apple a day keeps the doctor away</li>
    <li>God helps them who help themselves</li>
    <br/><br/>
    <em>
       -From My Collection.
    </em>
    </body>
</html>
```

Output



As we have defined the universal selector that sets the green color. Hence the text that is appearing on the above web page is in green color.

1.4.6 Attribute Selector

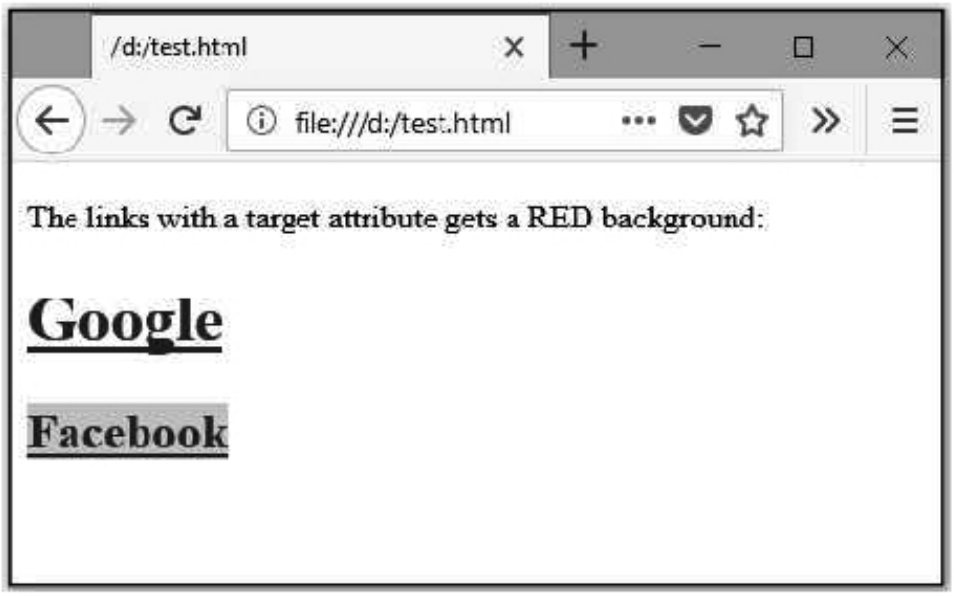
- CSS allows authors to specify rules that match elements which have certain attributes defined in the source document.
- The syntax is

```
[att] - Match when the element sets the "att" attribute, whatever the value of the attribute.
[att=val] - Match when the element's "att" attribute value is exactly "val".
```

- Example : In the following example all the <a> elements get selected with a target attribute.

```
<!DOCTYPE html>
<html>
<head>
<style>
        a[target] {
        background-color: red;
    }
</style>
</head>
<body>
<p>The links with a target attribute gets a RED background:</p>
<h1><a href="https://www.google.com">Google</a> </h1>
<h2><a href="http://www.facebook.com" target="_blank">Facebook</a></h2>
</body>
</html>
```

Output



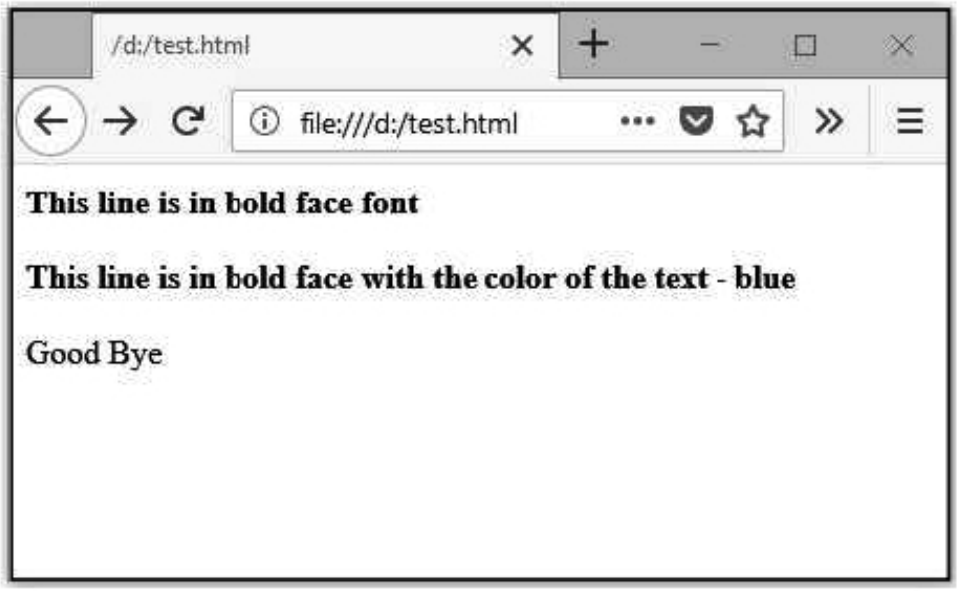
1.4.7 Contextual Selector

- "Contextual selectors" in CSS allow you to specify different styles for different parts of your document.
- You can assign styles directly to specific HTML tags, or you can create independent classes and assign them to tags in the HTML.

For example –

```
<!DOCTYPE html>
<html>
<head>
<style>
    p b {
        font-family: Times, serif; /* Font family */
        font-weight: bold; /* heavy faced type*/
        color: blue; /* blue colour of the text */
    }
</style>
</head>
 <body>
 <div><b>This line is in bold face font</b></div>
 <p><b>This line is in bold face with the color of the text - blue </b></p>
  <p> Good Bye</p>
 </body>
</html>
```

Output



1.5 Background

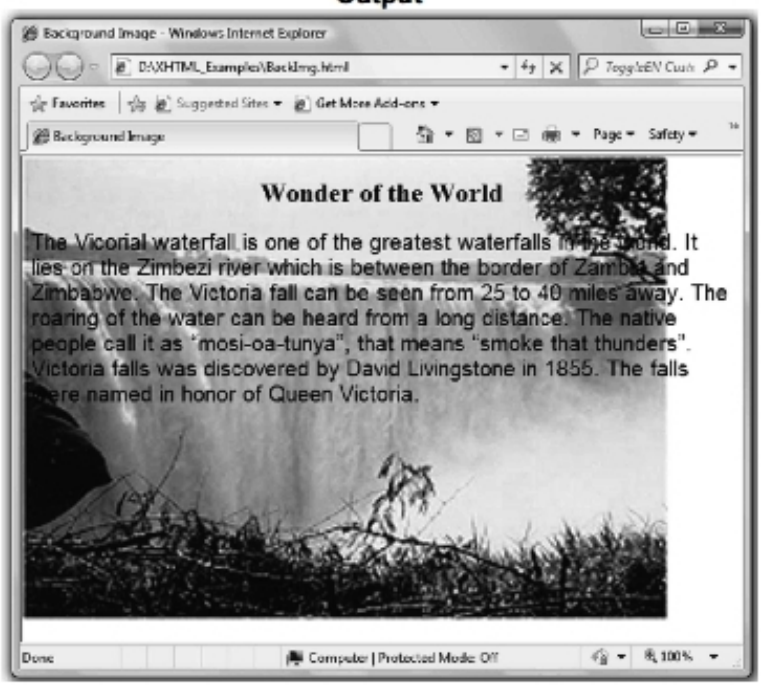
Using **background-image** property an image can be set as background.

HTML Document[BackImg.html]

```
<!DOCTYPE html>
<html>
   <head>
```

```
    <title>Background Image</title>
    <style type="text/css">
        body {background-image:url(victoriafalls.jpg);
            background-repeat:no-repeat; }
        p
        {
         font-family:Arial;
         font-size:20px;
        }
    </style>
    </head>
    <body>
    <center>
    <h2>Wonder of the World</h2>
    </center>
    <p>
    The Vicorial waterfall is one of the greatest waterfalls in the world.
    It lies on the Zimbezi river which is between the border of Zambia and Zimbabwe.
    The Victoria fall can be seen from 25 to 40 miles away.
    The roaring of the water can be heard from a long distance.
    The native people call it as "mosi-oa-tunya", that means "smoke that thunders".
    Victoria falls was discovered by David Livingstone in 1855.
    The falls were named in honor of Queen Victoria.
    </p>
    </body>
</html>
```

Output



It is expected that the foreground text over the image should be visible. That means the color of the background image must not be the same as the color of the text written over it.

- **Background-repeat property**

  The **background-image** property sets the background image with repetition. If we want to control the repetition of the image then we must set the **background-repeat** property. The values of this property could be no-repeat, repeat-x, repeat-y or repeat(default). The repeat-x means the background image gets repeated over the x-axis(horizontally) and repeat-y means the background image gets repeated over the y-axis(vertically).

- **Background-position property**

  Various values for back-ground position property could be **top, bottom, left** and **right**

Example 1.5.1 *Write a CSS rule that places a background image halfway down the page, tilting it horizontally. The image should remain in place when the use scrolls up or down.*

Solution :

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
p
{
 font-size:40px;
 color:magenta;

}
body
{

background-image:url(baby.jpg);
background-repeat:repeat-x;
background-attachment:fixed;
}
</style>
</head>
<body>
<p>
Twinkle, twinkle, little star, How I wonder what you are.Up above the world so high, Like a diamond in the sky.Twinkle, twinkle, little star, How I wonder what you are!
```

```
When the blazing sun is gone, When there's nothing he shines upon,Then you show your little light,
Twinkle, twinkle, through the night.
Twinkle, twinkle, little star, How I wonder what you are!
In the dark blue sky so deep Through my curtains often peep For you never close your eyes
Til the morning sun does rise
Twinkle, twinkle, little star How I wonder what you are Twinkle, twinkle, little star How I wonder
what you are
</p>
</body>
</html>
```

1.6 Color and Color Properties

Different browsers have different ability to deal with the colors. In the next subsequent sections we will discuss how to use colors using CSS.

1.6.1 Color Groups

There are **three groups of colors**. Those are as given below -

1. This is the smallest group of colors. In this group there are **16** colors. These are named colors. The colors in this group are enlisted as follows -

Sr.No.	Color Name	Hexadecimal Value
1	black	000000
2	red	FF0000
3	lime	00FF00
4	blue	0000FF
5	yellow	FFFF00
6	fuchsia	FF00FF
7	aqua	00FFFF
8	white	FFFFFF
9	silver	C0C0C0
10	gray	808080
11	maroon	800000
12	purple	800080

13	green	008000
14	olive	808000
15	navy	000080
16	teal	008080

These colors get displayed as it is on the web browsers. And all the web browsers support these colors.

2. There is a set of **216** colors which is called **web palette**. The elements of such colors are red, green and blue. The values of these elements can be 00, 33, 66, 99, CC and FF.
3. There are 16 millions colors. These are 24-bit colors.

Syntax : Color : colorname :

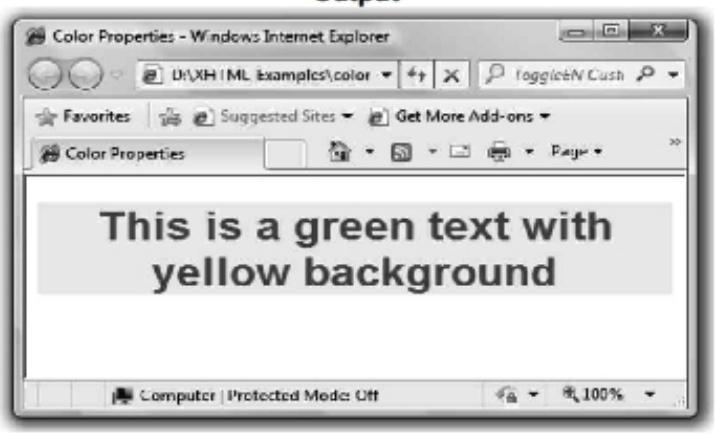
1.6.2 Color Properties

Using **color** property we can set the foreground color and using **background-color** the background color can be set. The use of these properties is shown in XHTML document.

HTML Document[colorProperties.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Color Properties</title>
    <style type="text/css">
        h1
        {
         font-family:Arial;
         color:green;
         background-color:yellow;
        }
    </style>
    </head>
    <body>
        <center>
        <h1>This is a green text with yellow background</h1>
        </center>
    </body>
</html>
```

Output



1.7 Manipulating Texts and Fonts

The font properties can be setting different types of fonts, styles and sizes.

1.7.1 Font Families

The **font-family** denotes different types of fonts such as Arial, Times New Roman, Script, monospace and so on. Following are some examples of generic fonts

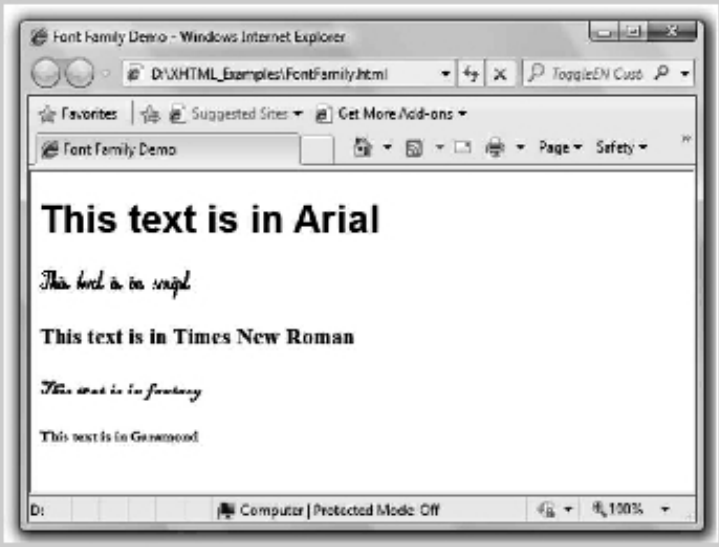
Generic Font Name	Example
sans-serif	Arial, Helvetica, Futura
cursive	Zapf-chancery
fantasy	Critter, Cottonwood
monospace	Courier, Prestige
serif	Times New Roman,Garamond

Let us see usage of **font-family** property in HTML document.

HTML Document[FontFamily.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Font Family Demo</title>
    <style type="text/css">
       h1
```

Output




```
{
        font-family:Arial;
       }
       h2
       {
        font-family:script;
       }
       h3
       {
        font-family:'Times New Roman';
       }
       h4
       {
        font-family:fantasy;
       }
       h5
       {
        font-family:Garamond;
       }
    </style>
    </head>
    <body>
       <h1>This text is in Arial </h1>
       <h2>This text is in script </h2>
       <h3>This text is in Times New Roman</h3>
       <h4> This text is in fantasy</h4>
       <h5> This text is in Garamond</h5>

    </body>
</html>
```

If the font name has more than one word then the name should be enclosed by the single quotes. Although quotes are not mandatory, it is a good practice to use quotes for specifying the font name.

1.7.2 Font Sizes

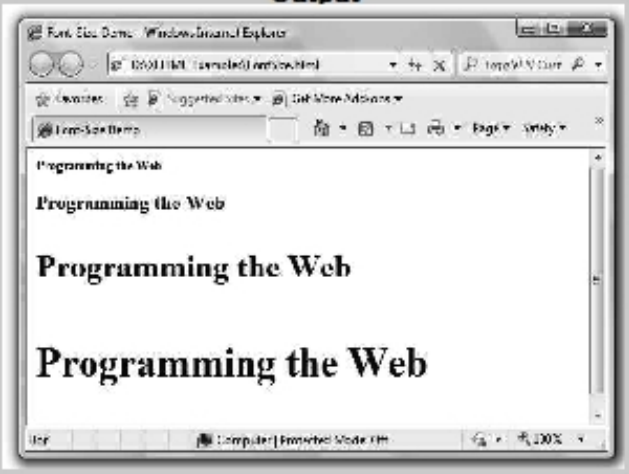
This property is used to specify the size of the font. One can specify the font size in points (pts), pixels (px) or in percentage (%). We can also specify the font size using the relative values such as **small, medium, large**. Use of font size in such a way is relative. And the disadvantage of using such relative sizes is that one cannot have the strict control over the font-size. Different browsers may have different font size values.

In the following HTML document we will use the font-size property.

HTML Document[FontSize.html]

```
<!DOCTYPE html>
<html>
    <head>

<title>Font-Size Demo</title>
    <style type="text/css">
        h1
        {
         font-size:10pt;
        }
        h2
        {
         font-size:20px;
        }
        h3
        {
         font-size:xx-large;
        }
        h4
        {
         font-size:250%;
        }

    </style>
    </head>
    <body>
        <h1>Programming the Web</h1>
        <h2>Programming the Web</h2>
        <h3>Programming the Web</h3>
        <h4>Programming the Web</h4>
    </body>
</html>
```

Output



1.7.3 Font Variants

The font variation can be achieved by making setting the font in upper case or in lower case.

We use **font-variant** property for this purpose.

Value	Meaning
normal	The font can be displayed in normal form.
small-caps	The font can be displayed in small capital letters.

HTML Document[FontVariant.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Font-Variant Demo</title>
    <style type="text/css">
       h1
{
        font-variant:small-caps;
       }
    </style>
    </head>
    <body>
       <h1>Programming the Web</h1>
    </body>
</html>
```

Output

1.7.4 Font Styles

Various font styles are -

- normal
- italic
- oblique

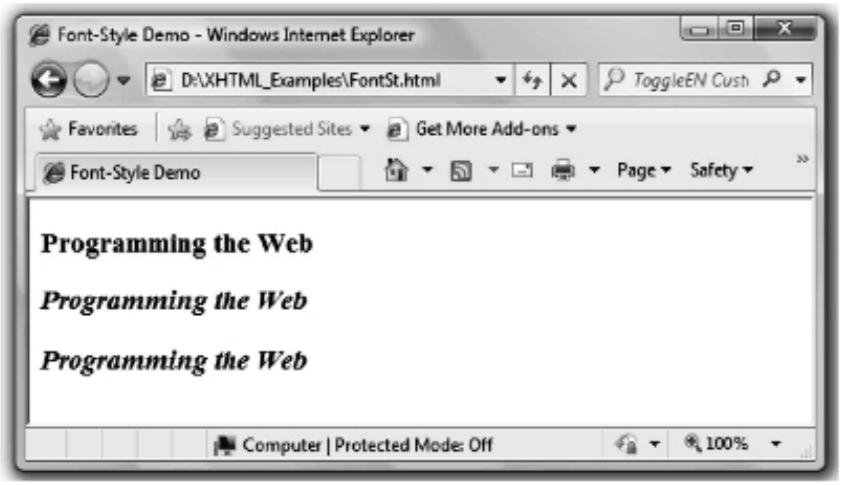
The illustration is as given below -

HTML Document[FontSt.html]

```
<!DOCTYPE html>
<html
    <head>
```

```
    <title>Font-Style Demo</title>
    <style type="text/css">
        h3.normal { font-style:normal;}
        h3.italic { font-style:italic;}
        h3.oblique { font-style:oblique;}
    </style>
    </head>
    <body>
        <h3 class="normal">Programming the Web</h3>
        <h3 class="italic">Programming the Web</h3>
        <h3 class="oblique">Programming the Web</h3>
    </body>
</html>
```

Output



1.7.5 Font Weights

Various font styles are -

- Normal (by default)
- Bold
- Bolder
- Lighter

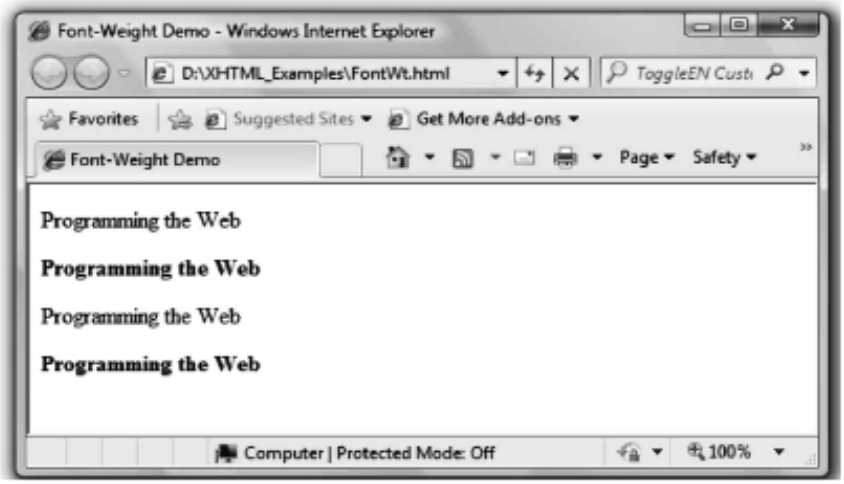
The illustration is as given below -

HTML Document[FontWt.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Font-Weight Demo</title>
    <style type="text/css">
```

```
        p.normal { font-weight:normal;}
        p.bold { font-weight:bold;}
        p.bolder { font-weight:bolder;}
        p.lighter { font-weight:lighter;}
    </style>
    </head>
    <body>
        <p class="normal">Programming the Web</p>
        <p class="bold">Programming the Web</p>
        <p class="lighter">Programming the Web</p>
        <p class="bolder">Programming the Web</p>
    </body>
</html>
```

Output



1.7.6 Font Shorthands

We can specify more than one font properties in a list

For example

HTML Document[Fontshrthnd.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Font Properties</title>
    <style type="text/css">
        p.myfont {font-family:Courier New;
            font-style:italic;
            font-weight:bold;
            font-size:28pt;
          }
```

```
    </style>
    </head>
    <body>
        <p class="myfont">I Love my Country!!!</p>
          </body>
</html>
```

Output

1.7.7 Text Decoration

Using text decoration property we can include special features in the text. Various properties of text decoration are

- underline
- overline
- line-through

HTML Document[TxtDecor.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Text Decoration</title>
    <style type="text/css">
        p.linethrough {text-decoration:line-through;}
        p.underline {text-decoration:underline;}
        p.overline {text-decoration:overline;}
        p.normal {text-decoration:normal;}
```

```
    </style>
    </head>
    <body>
        <p class="linethrough"> This text is having linethrough </p>
        <p class="overline"> This text is having overline </p>
        <p class="underline"> This text is having underline </p>
        <p class="normal"> This text is a normal text </p>

    </body>
</html>
```

Output

1.7.8 Alignment of Text

Using cascading style sheets we can align the text. This alignment can be done using following properties such as -

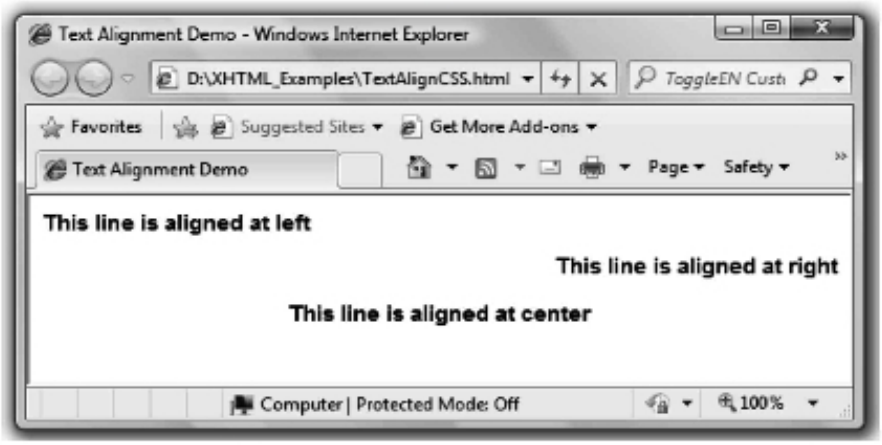
Property	Value	Meaning
text-align	left	Aligning the text to the left.
text-align	right	Aligning the text to the right
text-align	center	Alignment of text at the centre.
text-indent	value in inches	Desired indentation of the text can be applied.

Following is script which uses various text alignment properties.

HTML Document[TextAlignCSS.html]

```
<!DOCTYPE html>
<html>
    <head>
      <title>Text Alignment Demo</title>
      <style type="text/css">
       h1
       {
        font-family:Arial;
        font-size:15px;
        text-align:left;
       }
       h2
       {
        font-family:Arial;
        font-size:15px;
        text-align:right;
       }
       h3
       {
        font-family:Arial;
        font-size:15px;
        text-align:center;
       }
    </style>
    </head>

    <body>
        <h1>This line is aligned at left</h1>
        <h2>This line is aligned at right</h2>
        <h3>This line is aligned at center</h3>
    </body>
</html>
```

Output



1.8 Lists

There are two types of lists - unordered list and ordered lists. The unordered lists are given by bullets. These bullets are of various types such as square and circle disk.

For displaying the lists we use **list-style-type** property. For unordered list we can use property values such as

- Circle
- Disk
- Square
- None

Let us see their illustration in the HTML document.

HTML Document [list1.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>Unordered List Demo</title>
    <style type="text/css">
       ul {list-style-type:square}
    </style>
    </head>
    <body>
     <h2>Following are the useful vegetables...</h2>
     <ul>
       <li>Carrot</li>
       <li>Cucumber</li>
       <li>Spinach</li>
    </ul>
```

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

Output

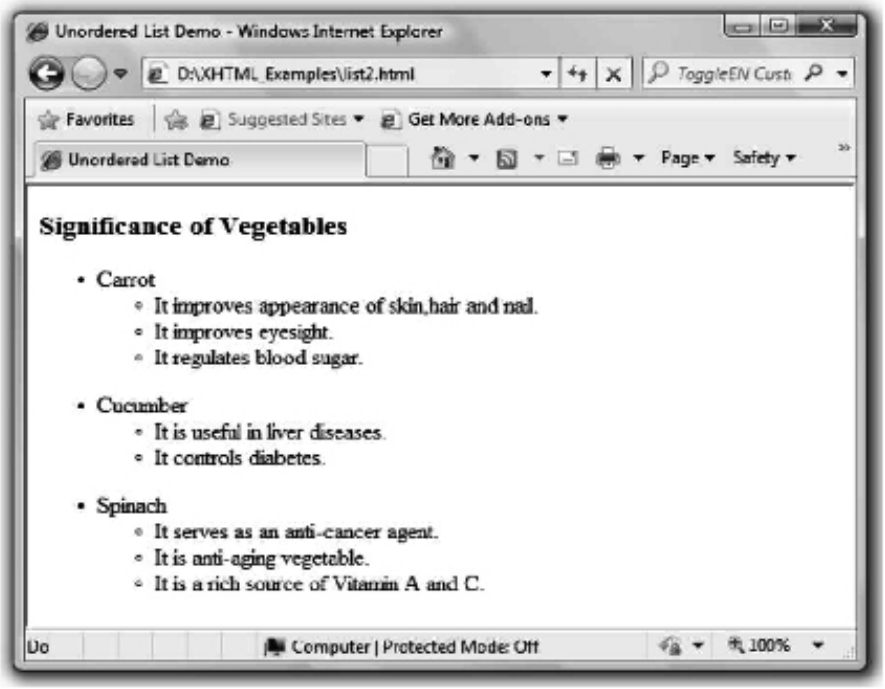
There is another way of representing the unordered list. It is illustrated by following HTML document.

HTML Document[list2.html]

```
<!DOCTYPE html>
<html >
    <head>
    <title>Unordered List Demo</title>
    <style type="text/css">
    li.disk {list-style-type:disk}
    li.circle {list-style-type:circle}
    </style>
    </head>
    <body>
    <h3> Significance of Vegetables</h3>
    <ul>
     <li class="disk">Carrot
      <ul>
       <li class="circle">It improves appearance of skin,hair and nail.</li>
       <li class="circle">It improves eyesight.</li>
       <li class="circle">It regulates blood sugar.</li>
      </ul>
     </li>
```

```
    </ul>
    <ul>
     <li class="disk">Cucumber
      <ul>
       <li class="circle">It is useful in liver diseases.</li>
       <li class="circle">It controls diabetes.</li>
      </ul>
     </li>
    </ul>
    <ul>
     <li class="disk">Spinach
      <ul>
       <li class="circle">It serves as an anti-cancer agent.</li>
       <li class="circle">It is anti-aging vegetable.</li>
       <li class="circle">It is a rich source of Vitamin A and C.</li>
      </ul>
     </li>
    </ul>
    </body>
</html>
```

Output

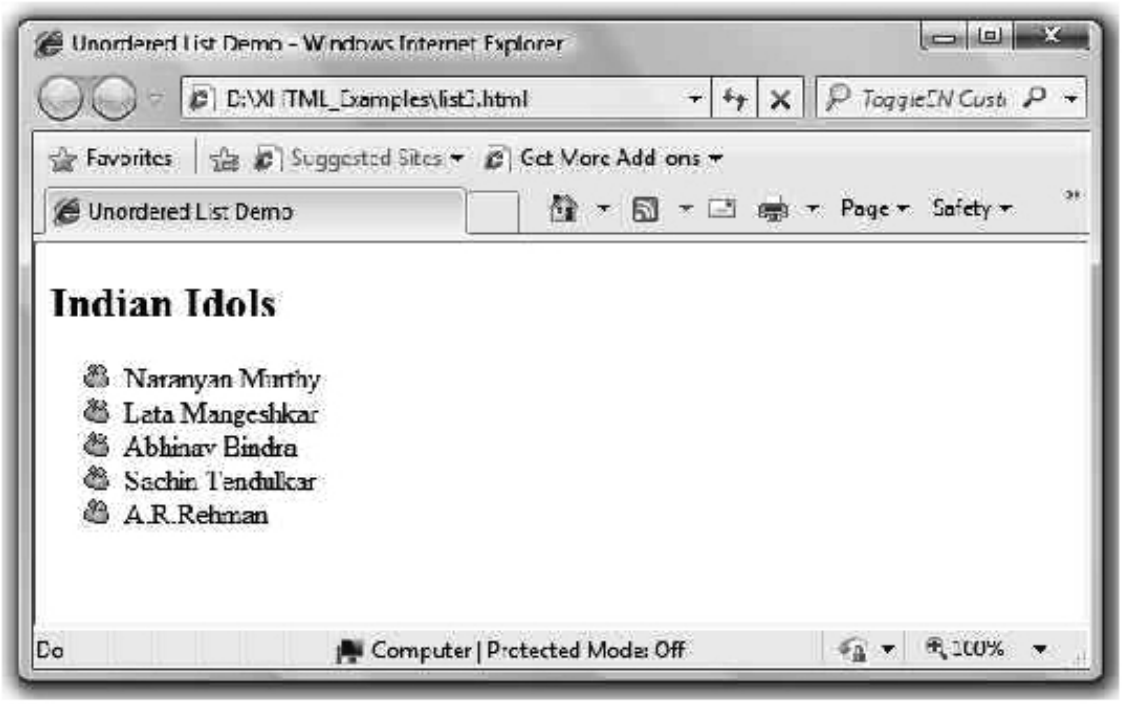


We can also display small images as the list item bullet. For example

HTML Document[list3.html]

```
<!DOCTYPE html>
<html>
    <head>
        <title>Unordered List Demo</title>
        <style type="text/css">
          li.image {list-style-image:url(people.gif)}
            </style>
    </head>
    <body>
      <h2>Indian Idols</h2>
      <ul>
          <li class="image">Naranyan Murthy</li>
          <li class="image">Lata Mangeshkar</li>
          <li class="image">Abhinav Bindra</li>
          <li class="image">Sachin Tendulkar</li>
          <li class="image">A.R.Rehman</li>
      </ul>
      </body>
</html>
```

Output



Let us now make use of ordered list.

For an ordered list various property values are enlisted below.

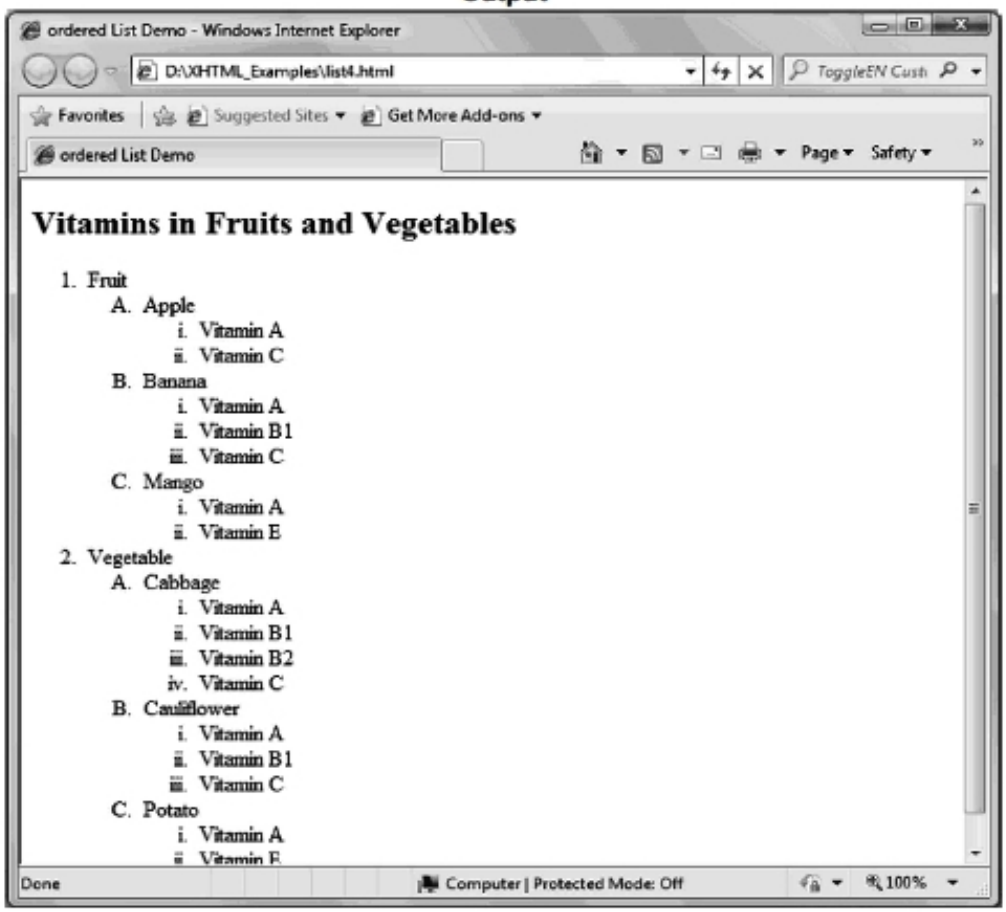
Property Value	Sample preview
decimal	1,2,3,4
upper-alpha	A,B,C,D
lower-alpha	a,b,c,d
upper-roman	I,II,III,IV
lower-roman	I,ii,iii,iv

HTML Document[list4.html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>ordered List Demo</title>
    <style type="text/css">
     ol  {list-style-type:decimal;}
     ol  ol  {list-style-type:upper-alpha;}
     ol  ol  ol {list-style-type:lower-roman;}
    </style>
    </head>
    <body>
    <h2>Vitamins in Fruits and Vegetables</h2>
    <ol>
     <li>Fruit
      <ol>
       <li>Apple
        <ol>
         <li>Vitamin A</li>
         <li>Vitamin C</li>
        </ol>
       </li>
       <li>Banana
        <ol>
          <li>Vitamin A</li>
          <li>Vitamin B1</li>
          <li>Vitamin C</li>
```

```
        </ol>
        </li>
        <li>Mango
         <ol>
          <li>Vitamin A</li>
          <li>Vitamin E</li>
         </ol>
        </li>
        </ol>
 </li>
    <li>Vegetable
      <ol>
       <li>Cabbage
        <ol>
          <li>Vitamin A</li>
          <li>Vitamin B1</li>
          <li>Vitamin B2</li>
          <li>Vitamin C</li>
        </ol>
       </li>
       <li>Cauliflower
        <ol>
          <li>Vitamin A</li>
          <li>Vitamin B1</li>
          <li>Vitamin C</li>
        </ol>
       </li>
       <li>Potato
         <ol>
          <li>Vitamin A</li>
          <li>Vitamin E</li>
         </ol>
        </li>
       </ol>
     </li>

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

Output



1.9 Responsive Design

- **Definition :** "Responsive design" refers to the idea that your website should display equally well in everything from widescreen **monitors to mobile phones.**
- It is called responsive web design because you can resize, hide, shrink, enlarge, or move the contents of the web page to make it look good on any screen.
- The responsive web design makes use of HTML and CSS only. It is neither a program nor a JavaScript.
- Using the responsive design approach the web developer can give the best experience for all the user. For example -

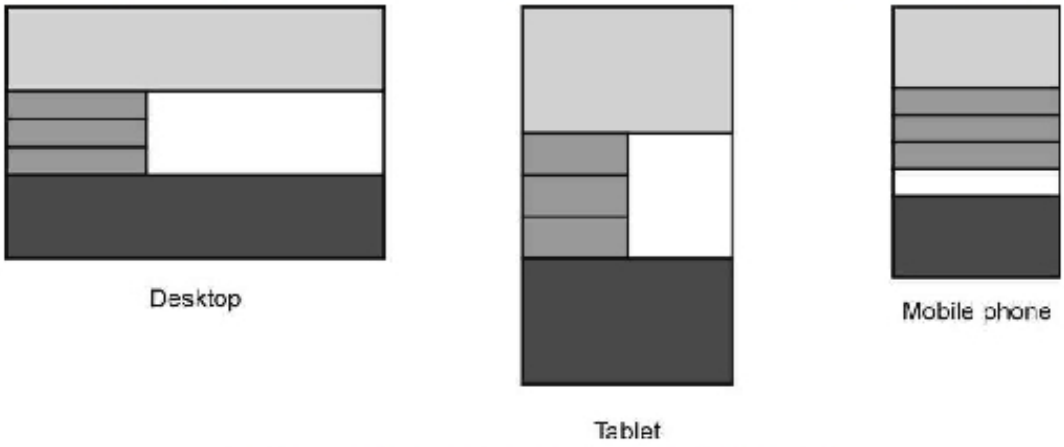


Fig. 1.9.1 Responsive Layout for Different Devices

- There are four **key components** that make responsive design work –
  - Liquid Layouts
  - Scaling Images to the viewport size.
  - Setting viewports via <meta> tag.
  - Customizing the CSS for different viewports using media queries.
- Making the use of **liquid layouts** means specifying the widths in percentage. This is the most obvious part of any responsive web design approach. **Scaling images** to the viewport size means you have to specify following rule:

```
img {
  Width:100%;
}
```

- It only shrinks or expands the visual display of the image to fit the size of the browser window, never expanding beyond its actual dimensions.

1.9.1 Setting Viewports

- Viewport means user's **visible area** of web page.
- The viewport varies from mobile phone to desktop computers.
- HTML5 allows the web developer to take control over the viewport using <meta> tag.

- Following line must be included in your HTML document to set the viewport :

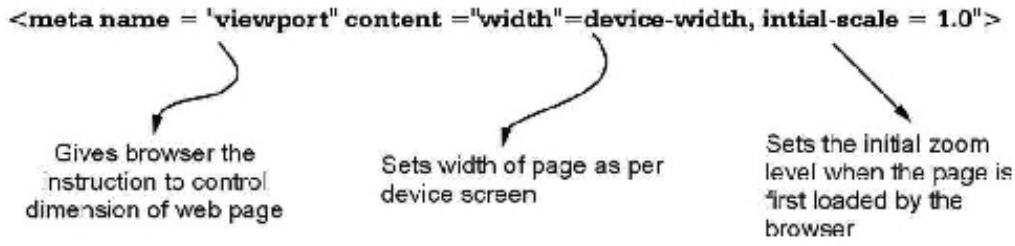


Fig. 1.9.2

The HTML document structure will be :

```
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        ...
    </style>
   </head>
   <body>
         ...
    </body>
  </html>
```

- By setting the viewport in above manner, the page is telling the browser that no scaling is needed, and to make the viewport as many pixels wide as the device screen width.
- This means that if the device has a screen that is 320 px wide, the viewport width will be 320 px; if the screen is 480 px then the viewport width will be 480 px.

1.9.2 Media Queries

- Another key component of responsive web design is media queries. Media queries is for different style rules for different size devices such as mobiles, desktops and so on.
- It uses the **@media** rule to include a block of CSS properties only if a certain condition is true.
- For example – Following HTML document makes use of media queries for different types of devices(such as mobile phones, tablets and desktops)

HTML Document

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

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
   background-color: red;
}
/*Rules for Phones*/
@media only screen and (max-width: 480px)
{
   body { background-color: green; }
}
/*Rules for Tablets*/
@media only screen and (min-width: 481px)
         and (max-width: 768px)
{
   body { background-color: blue; }
}
/*Rules for Desktops */
@media only screen and (min-width: 769px)
{
   body { background-color: yellow; }
}
</style>
</head>
<body>
<h2>Resize the browser window and
    observe the change in color.</h2>
</body>
</html>
```

Output

Review Questions

1. *Write a short note on – Responsive design in HTML*
2. *Explain with illustrative example , how to set viewport?*
3. *What is media queries? Explain its use with the help of example.*

1.10 Bootstrap Introduction

- Bootstrap is open source, front end web framework. It is used for designing web-site and web applications.
- Bootstrap helps the developer to create responsive designs. Responsive web design means creating the web site which automatically adjusts itself to look good on all the devices. These devices range from small phones to large desktops screens.
- Bootstrap includes HTML, CSS and JavaScript based design templates which includes forms, buttons, tables, navigation, modals and so on.

Features of Bootstrap

Setting up Environment

There are **two ways to use Bootstrap** in a webpage.

1) The **first way** is to use a CDN or Content Delivery Network. Using bootstrap CDN means that we will not download and store the bootstrap files in our server or local machine. We will just include the bootstrap CSS and JavaScript links on our page.
2) The **second way** is downloading and storing a copy Bootstrap files in our web server or local machine. We can download the Bootstrap from https://getbootstrap.com/

I have created a folder named **bootstrap-3.3.7** and downloaded my Bootstrap in this folder. After getting downloaded I can see three folders in it namely css, fonts and js

- If we want to **use the CDN** then we will include Bootstrap CSS as follows -

```
<link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

- If we want to use our **own downloaded copy** then we will use

```
<link rel="stylesheet"
href="bootstrap-3.3.7/css/bootstrap.min.css">
```

Similarly the jquery can be used as

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.0.0/jquery.min.js"></script>
                                        Or
<script src="jquery-3.0.0.min.js"></script>
```

Components of Bootstrap

1. **HTML 5 :** The Bootstrap uses the HTML elements and CSS properties, hence the code is contained within the <doctype>.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
... ...
                Script declarations
</head>
<body>
...
...
</body>
</html>
```

2. **The mobile-first :** Bootstrap 3 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework.

For proper rendering the <meta> tag inside the <head> section is specified. It is specified as follows -

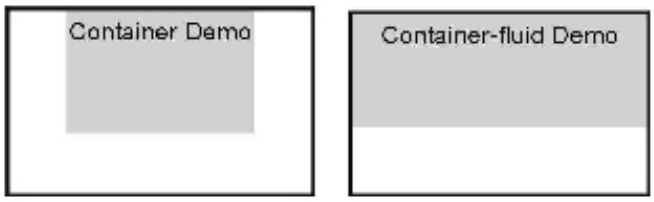
```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

The width=device-width part sets the width of page which is based on the screen-width of the device.

The initial-scale=1 part sets the initial zoom level when the page is loaded on the browser.

3. **Container :** The container element is for wrapping the site contents. There are **two classes** for the container -
   i) **container :** It provides a responsive fixed width container.
   ii) **container-fluid :** It provides full width container occupying the entire width of the viewport.

For example



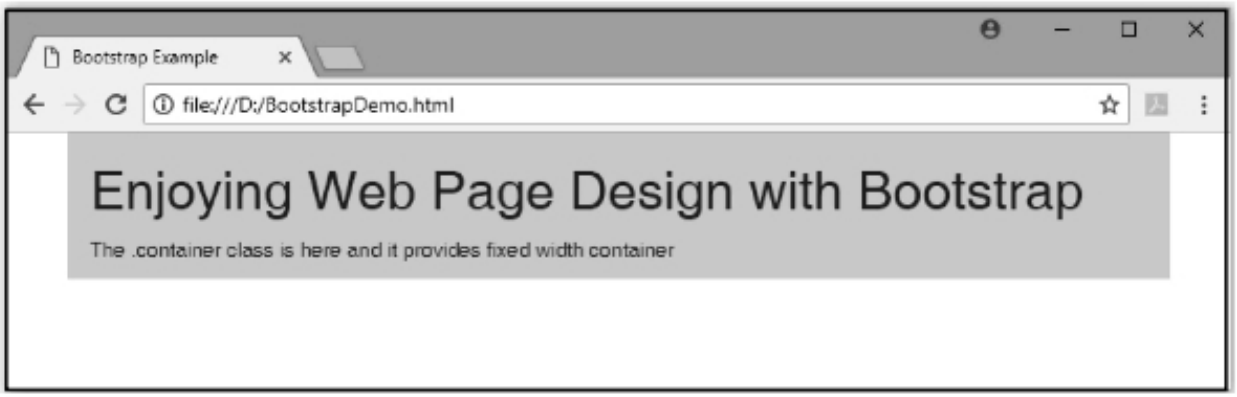
How to write first Bootstrap Code ?

Step 1 : Open a Notepad and type the following code. Save it using the filename extension **.html**

BootstrapDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container" style="background-color:grey;">
 <h1 >Enjoying Web Page Design with Bootstrap</h1>
 <p>The .container class is here and it provides fixed width container</p>
</div>
</body>
</html>
```

Output



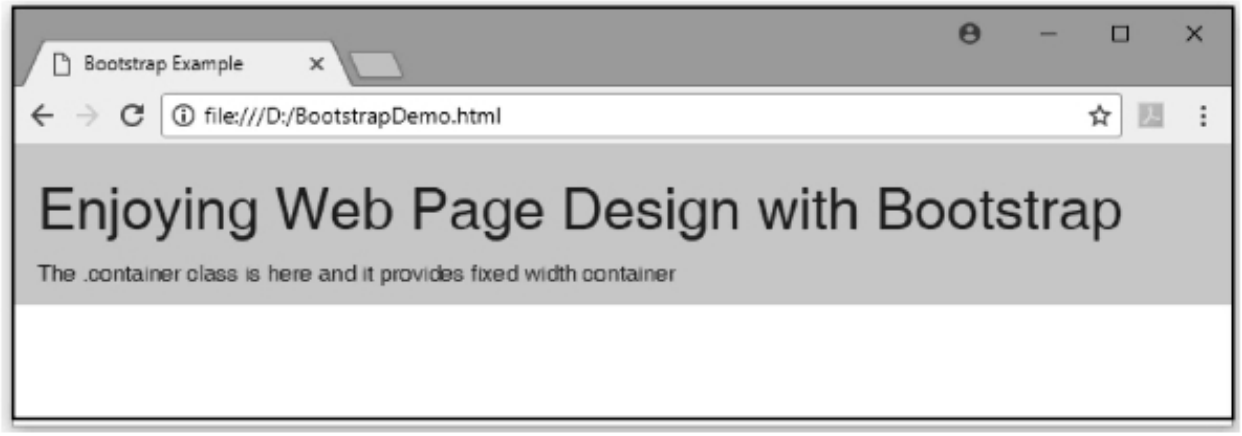
Similarly if we change the element class to **container-fluid** we will get the content to be occupied the entire web page. It can be demonstrated by modifying the above script as follows -

BootstrapDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container-fluid" style="background-color:grey;">
 <h1 >Enjoying Web Page Design with Bootstrap</h1>
 <p>The .container class is here and it provides fixed width container</p>
</div>

</body>
</html>
```

Output



1.10.1 Grid System

- In Bootstrap there are 12 columns across the page. It is not always necessary to have 12 columns we can group the columns and wider columns can be created.
- The Bootstrap's Grid system is **responsive**. That means as we resize the browser window, the columns arrangement gets changed as per the screen size.

- There are four classes of Bootstrap's Grid system and those are -

Class	Purpose
xs	It is used for phones.
sm	It is used for tablet screen.
md	It is used for small laptops.
lg	It is for laptops and desktops.

The above classes can be combined to create more dynamic and flexible display.

Structure of Bootstrap Grid

The typical structure of Bootstrap Grid is as follows -

```
<div class="row">
 <div class="col-*-*"></div>
 <div class="col-*-*"></div>
</div>
<div class="row">
 <div class="col-*-*"></div>
 <div class="col-*-*"></div>
 <div class="col-*-*"></div>
</div>
...
...
```

In above structure we adding the row by

```
<div class="row">
```

And then adding the column for that row using

```
<div class="col-*-*"></div>
```

Here first * indicates the class as xs or sm or md or so on. The second * indicates the number of columns. We can add at the most 12 columns.

Let us understand how to use Grid system with a simple example

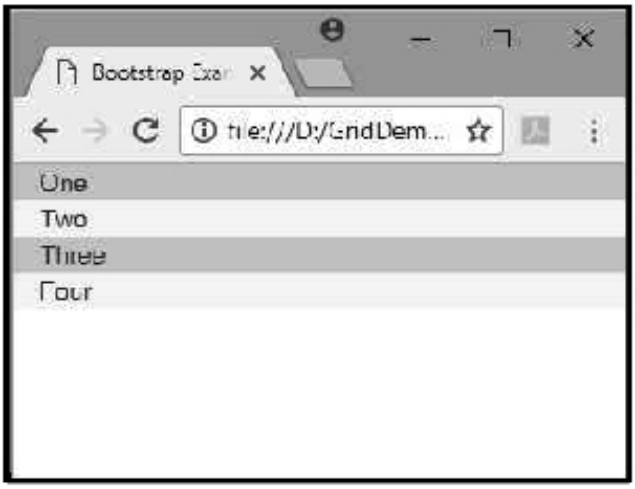
GridDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>

<div class="container-fluid">
<div class="row">
   <div class="col-sm-3" style="background-color:grey;">One</div>
   <div class="col-sm-3" style="background-color:aqua;">Two</div>
   <div class="col-sm-3" style="background-color:grey;">Three</div>
   <div class="col-sm-3" style="background-color:aqua;">Four</div>
 </div>
</div>
</body>
</html>
```

Output

If we resize the browser window we get the columns to be stacked one upon other



Similarly we can create the columns of unequal width, if we change the **<col-sm>**. For example –

```
<div class="col-sm-8" style="background-color:grey;">One</div>
 <div class="col-sm-4" style="background-color:aqua;">Two</div>
```

1.10.2 Typography

- In this section we will learn how to display the text in different style using Bootstrap elements.
- By **default font size** is 14 px.
- The HTML elements will be displayed with different style in Bootstrap.

Heading

The tags <h1> to <h6> are used to display the heading. For example

HeadingDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>
<div class="container">
        <h1> This line is of 36px </h1>
        <h2> This line is of 30px </h2>
        <h3> This line is of 24px </h3>
        <h4> This line is of 18px </h4>
        <h5> This line is of 14px </h5>
        <h6> This line is of 12px </h6>
</div>
</body>
</html>
```

Output

The small element

The <small> element is used to create the small light text in heading. For example

HeadingDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>
<div class="container">
    <h1> This line is of <small>36px</small> </h1>
    <h2> This line is of <small>30px</small> </h2>
</div>
</body>
</html>
```

Output

The mark element

The <mark> element can be used to highlight the important text present in the line. Here is the simple illustration

MarkDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>

<div class="container">
        <p> This line is an <mark>important</mark> message </p>
</div>
</body>
</html>
```

Output

The <pre> element

For displaying the text on multiple lines we can use <pre> tag. Here is the simple example of this tag

PreDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>

<div class="container">
        <pre>
        Humty Dumty
        sat on a wall.
        Jack and Jill
        went up the hill.
        Twinkle Twinkle
        little star
        How I wonder
        What you are!!!
        </pre>
</div>
</body>
</html>
```

Output

Colors and Background

- There are some contextual classes using which specific meaning can be represented through the colors. The classes for text colors are -

.text-muted, .text-primary, .text-success, .text-info, .text-warning, .text-danger

- Similarly we can align the text to left, right or center using text-left, text-right and text-center.
- These are called contextual classes which provides the meaning through text.
- Let us use these classes in following demo application

TextDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
 <div class="container">
        <p class="text-left">This line is left aligned</p>
        <p class="text-right">This line is right aligned</p>
        <p class="text-center">This line is at center </p>
        <p class="text-muted">This line is using muted class</p>
        <p class="text-primary">This line is using primary class</p>
```

```
        <p class="text-success">This line is using success class</p>
        <p class="text-info">This line is using info class</p>
        <p class="text-warning">This line is using warning class</p>
        <p class="text-danger">This line is using danger class</p>
</div>
</body>
</html>
```

Output

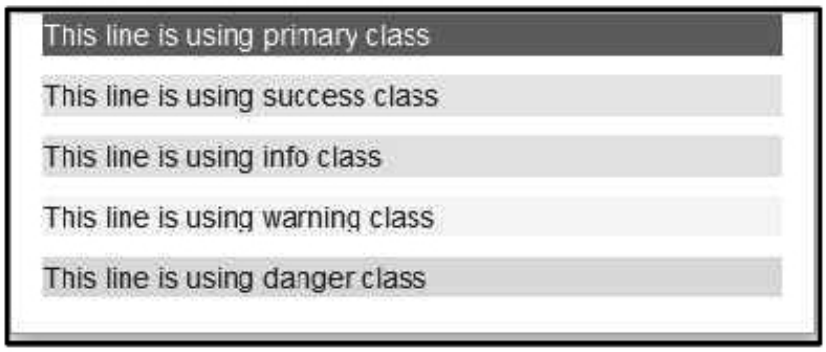
Similarly we can set background color for these classes as follows :

TextBGDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>
<div class="container">
        <p class="bg-primary">This line is using primary class</p>
        <p class="bg-success">This line is using success class</p>
```

```
        <p class="bg-info">This line is using info class</p>
        <p class="bg-warning">This line is using warning class</p>
        <p class="bg-danger">This line is using danger class</p>
</div>
</body>
</html>
```

Output



1.10.3 Tables

The <table> tag is used to create the table. The **<tr>** tag is used for **rows** and **<td>** tag is used for **columns**. The heading in the table can be denoted within <thead><th> </th> </thead> tag while remaining contents of the table are specified in <tbody> tag.

The following example demonstrates the use of table tag

TableDemo1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
        <table class="table">
                <thead>
                        <tr><th> Name </th><th> City </th></tr>
                </thead>
                <tbody>
                        <tr>
```

```
                    <td>AAA</td>
                    <td>Pune</td>
                </tr>
                <tr>
                    <td>BBB</td>
                    <td>Mumbai</td>
                </tr>
                <tr>
                    <td>CCC</td>
                    <td>Chennai</td>
                </tr>
            </tbody>
</div>
</body>
</html>
```

Output

For having the grids to the table, we simply change the

```
<table class="table">
```

to

```
<table class="table table-bordered">
```

1.10.4 Images

For displaying the images the **<img src="">** tag is used. You have to specify the image file name within the double quotes.

There are three classes that are used to display the image. These classes are -

1. **img-rounded :** The image is displayed in a border-circle of radius 6px.
2. **img-circle :** The image is displayed in circle of radius 500px
3. **img-thumbnail :** The image is displayed in a rectangle having padding and grey border.

Here is illustrative example of displaying images

ImgDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<img src = "bookImg.jpg" class = "img-rounded">
<img src = "bookImg.jpg" class = "img-circle">
<img src = "bookImg.jpg" class = "img-thumbnail">
</body>
</html>
```

Output

1.10.5 Button

In Bootstrap we can create the buttons of different styles. Various button classes are used as follows -

.btn, .btn-default, .btn-primary, .btn-success, .btn-info, .btn-warning, .btn-danger, .btn-link

The example code is as follows -

ButtonDemo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
 <button type="button" class="btn">Basic</button>
 <button type="button" class="btn btn-default">Default</button>
 <button type="button" class="btn btn-primary">Primary</button>
 <button type="button" class="btn btn-success">Success</button>
 <button type="button" class="btn btn-info">Info</button>
 <button type="button" class="btn btn-warning">Warning</button>
 <button type="button" class="btn btn-danger">Danger</button>
 <button type="button" class="btn btn-link">Link</button>
</body>
</html>
```

Output

We can also create active or disabled buttons using active and disabled class. The illustration is as follows –

ButtonDemo1.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
 <button type="button" class="btn btn-success">Success</button>
 <button type="button" class="btn btn-success active">Active</button>
 <button type="button" class="btn btn-success disabled">Disabled</button>
</body>
</html>
```

Output

1.10.6 Form

In Bootstrap there are three types of form layouts.

1. Vertical Form 2. Horizontal Form 3. Inline Form

For using the form we must add the element as ** **. The required controls such as <input>, <select>, <textarea> and so on can be added within this class.

Let us discuss and learn how to create the above mentioned types of forms –

1. Vertical or Basic Form

To create the vertical or basic form use following steps –

Step 1 : Add a role to form element as -

```
<form role="form">
```

Step 2 : Inside the wrap all the controls

Step 3 : While adding any textual control such as <text>, <textarea> and <select> add the class **"form-control"**

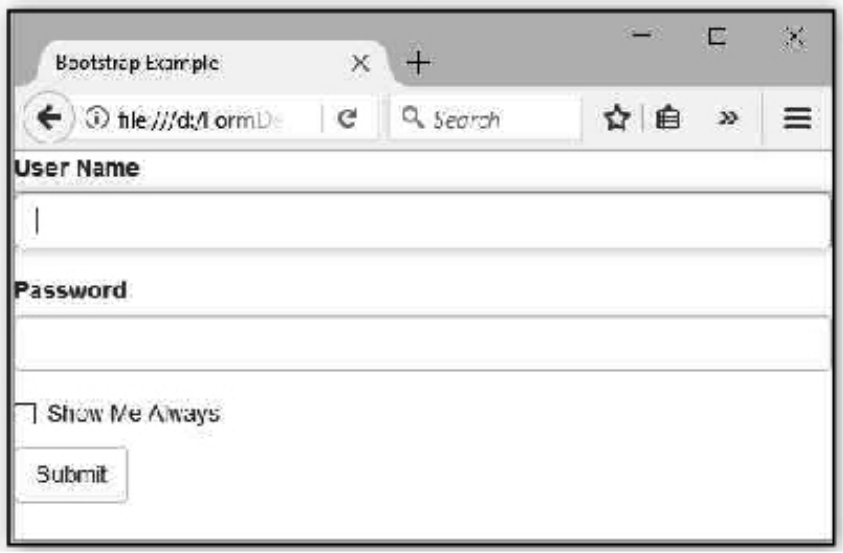
Here is example code –

FormDemo1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>
<form role = "form">


        <div class = "form-group">

        <label for = "name">User Name</label>

        <input type = "text" class = "form-control" id = "name">
        </div>


        <div class = "form-group">

        <label for = "pwd">Password</label>

        <input type = "password" class = "form-control" id = "pwd">

        </div>

        <div class = "checkbox">

        <label><input type = "checkbox"> Show Me Always</label>

        </div>
```

```
        <button type = "submit" class = "btn btn-default">Submit</button>

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

Output



2. Horizontal Form

To create the vertical or basic form use following steps –

Step 1 : Add a role to form element as -

```
<form role="form"> and form class="form-horizontal"
```

Step 2 : Inside the wrap all the labels and controls

Step 3 : While adding any textual control such as <text>, <textarea> and <select> add the class "form-control"

Step 4 : Add a class of "control-label" to the labels.

Here is example code –

FormDemo2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

```
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<form class ="form-horizontal" role = "form">
        <div class = "form-group">
        <label class="control-label col-sm-4" for = "name">User Name</label>
        <div class="control-label col-sm-8">
                <input type = "text" class = "form-control" id = "name">
        </div>
        </div>
        <div class = "form-group">
        <label class="control-label col-sm-4" for = "pwd">Password</label>
        <div class="control-label col-sm-8">
                <input type = "password" class = "form-control" id = "pwd">
        </div>
        </div>
        <div class="form-group">
        <div class="col-sm-offset-4 col-sm-8">
        <div class = "checkbox">
        <label><input type = "checkbox"> Show Me Always</label>
        </div>
        </div>
        </div>
        <div class="form-group">
        <div class="col-sm-offset-4 col-sm-8">
        <button type = "submit" class = "btn btn-default">Submit</button>
        </div>
        </div>
</form>
</body>
</html>
```

3. Inline Form

In an inline form, all of the elements are inline, left-aligned, and the labels are alongside.

To create the inline form use following steps –

Step 1 : Add a role to form element as -

```
        <form role="form">
```

And the class as "form-inline" to the <form> element

Step 2 : Inside the wrap all the controls

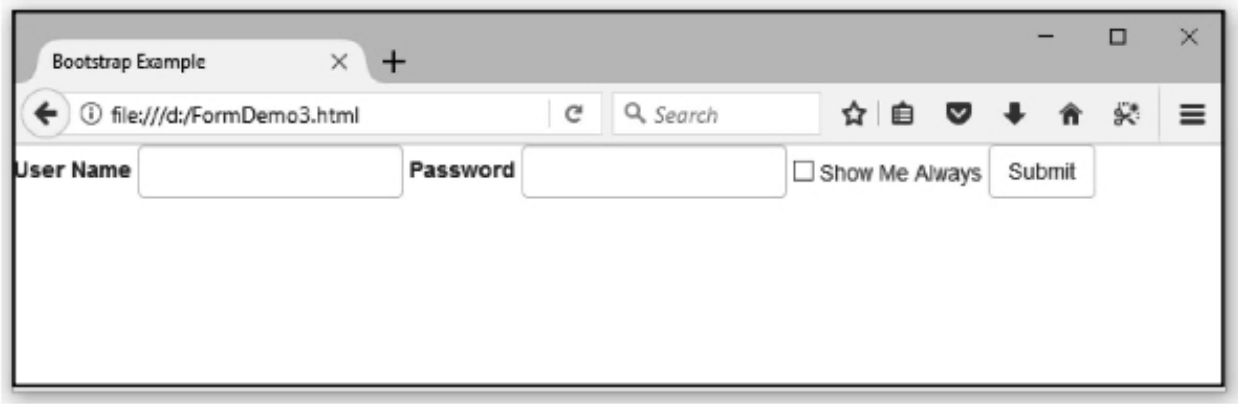
Step 3 : While adding any textual control such as <text>, <textarea> and <select> add the class **"form-control"**

Here is example code –

FormDemo3.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>
<body>
<form class="form-inline" role = "form">
        <div class = "form-group">
        <label for = "name">User Name</label>
        <input type = "text" class = "form-control" id = "name">
        </div>


        <div class = "form-group">
        <label for = "pwd">Password</label>
        <input type = "password" class = "form-control" id = "pwd">
        </div>

        <div class = "checkbox">
        <label><input type = "checkbox"> Show Me Always</label>
        </div>

        <button type = "submit" class = "btn btn-default">Submit</button>

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

Output



Part II : JavaScript

1.11 Java Script Syntax

- JavaScript is a client side scripting language.
- JavaScript was developed by Netscape in 1995. At that time its name was LiveScript. Later on Sun Microsystems joined the Netscape and then they developed **LiveScript**. And later on its name is changed to **JavaScript**.

The JavaScript can be directly embedded within the HTML document or it can be stored as external file.

Directly embedded JavaScript

The syntax of directly embedding the JavaScript in the HTML is

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

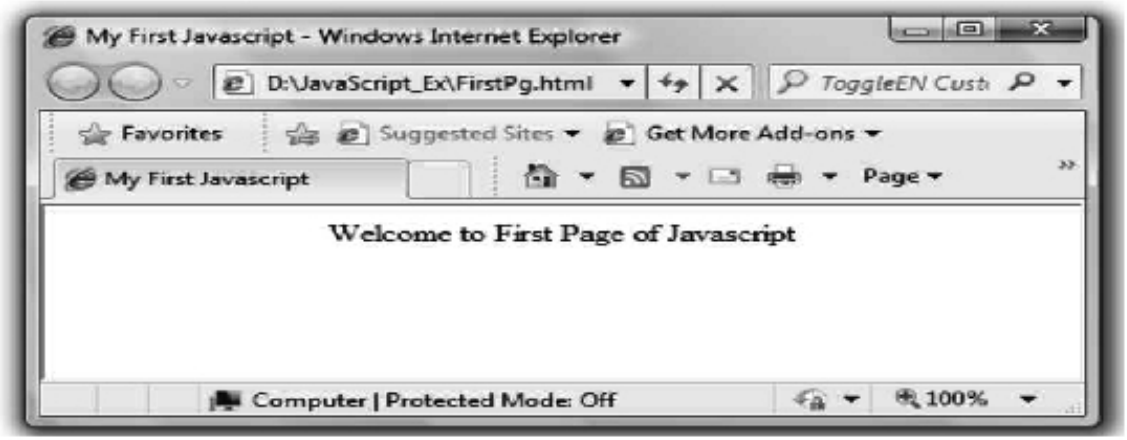
Example 1.11.1 *Write a JavaScript to display Welcome message in JavaScript*

Solution :

```
<!DOCTYPE html >
<html >
    <head>
        <title> My First Javascript </title>
    </head>
    <body>
        <center>
```

```
        <script type="text/javascript">
        /*This is the First JavaScript*/
        document.write(" Welcome to First Page of Javascript");
        </script>
        </center>
    </body>
</html>
```

Output



Script Explanation :

In above script

(1) We have embedded the javaScript within

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

(2) A comment statement using /* and */. Note that this type of comment will be recognized only within the <script> tag. Because, JavaScript supports this kind of comment statement and not the XHTML document.

(3) Then we have written document.write statement, using which we can display the desired message on the web browser.

Indirectly Embedding JavaScript

If we want to embed the JavaScript indirectly, that means if the script is written in some another file and we want to embed that script in the HTML document then we must write the script tag as follows -

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

Javascript is which is to be embedded is in the file MyPage.js

```
        …
        …
</script>
```

We will follow the following steps to use the external JavaScript file.

Step 1 : Create an XHTML document as follows -

XHTML Document[FirstPg.html]

```
<!DOCTYPE html >
<html>
        <head>
                <title> My First Javascript </title>
        </head>
                <body>
                        <center>
                        <script type="text/javascript" src="my_script.js">
                        </script>
                        </center>
                </body>
        </html>
```

This is an external javascript file,it can be specified with the attribute **src**

Step 2 :

JavaScript[my_script.js]

```
        /*This is the First JavaScript*/
                        document.write(" Welcome to First Page of Javascript");
```

Step 3 : Open the HTML document in Internet Explorer and same above mentioned output can be obtained.

Advantages of indirectly embedding of JavaScript

1. Script can be hidden from the browser user.
2. The layout and presentation of web document can be separated out from the user interaction through the JavaScript.

Disadvantages of indirectly embedding of JavaScript

1. If small amount of JavaScript code has to be embedded in XHTML document then making a separate JavaScript file is meaningless.
2. If the JavaScript code has to be embedded at several places in XHTML document then it is complicated to make separate JavaScript file and each time invoking the code for it from the XHTML document.

1.12 Java Script Inbuilt Objects

In JavaScript object is a collection of properties. These properties are nothing but the members of the classes from Java or C++. For instance - in JavaScript the object Date() is used which happens to the member of the class in Java.

1.12.1 Math Objects

- For performing the mathematical computations there are some useful methods available from **math** object.
- For example if we want to find out minimum of two numbers then we can write -

```
document.write(math.min(4.5,7.8));
```

The above statement will result in 4.5. Thus using various useful methods we can perform many mathematical computations.

- Here are some commonly used methods from **math** object.

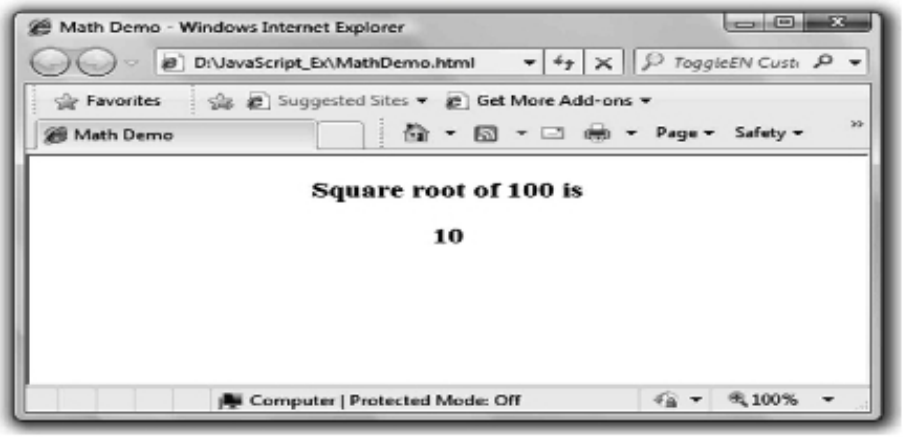
Method	Meaning
sqrt(num)	This method finds the square root of num.
abs(num)	This method returns absolute value of num.
ceil(num)	This method returns the ceil value of num. For example ceil(10.3) will return 11.
floor(num)	This method returns the floor value of num. For example floor(10.3) will return 10.
log(num)	This method returns the natural logarithmic value of num. For example log(7.9) will return 2.
pow(a,b)	This method will compute the ab. For example pow(2,5) will return 32.
min(a,b)	Returns the minimum value of a and b.
max(a,b)	Returns the maximum value of a and b.
sin(num)	Returns the sine of num.
cos(num)	Returns the cosine of num.
tan(num)	Returns the tangent of num.
exp(num)	Returns the exponential value i.e. enum .

JavaScript Program[MathDemo.html]

```
<!DOCTYPE html>
<html >
 <head>
  <title>Math Demo</title>
```

```
</head>
<body>
 <center>
  <h3>Square root of 100 is </h3>
  <script type="text/javascript">
   var num=100;
   document.write("<h3>"+Math.sqrt(num)+"</h3>");
  </script>
 </center>
</body>
</html>
```

Output



1.12.2 Number Objects

Various properties of number object are -

Property	Meaning
MAX_VALUE	Largest possible number gets displayed.
MIN_VALUE	Smallest possible number gets displayed.
NaN	When not a number then NaN is displayed.
PI	The value of PI gets displayed.
POSITIVE_INFINITY	The positive infinity gets displayed.
NEGATIVE_INFINITY	The negative infinity gets displayed.

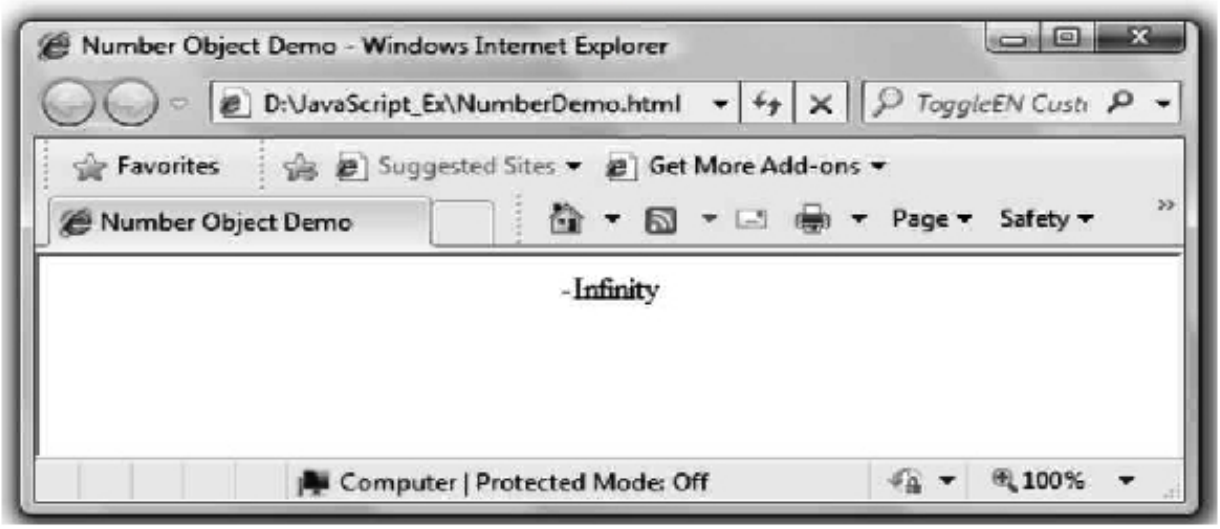
Using **Number.***property_name* we can display the property value. Following JavaScript uses the property of negative infinity.

JavaScript[NumberDemo.html]

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

```
<head>
  <title>Number Object Demo</title>
</head>
<body>
 <center>
  <script type="text/javascript">
  document.write(Number.NEGATIVE_INFINITY);
  </script>
 </center>
</body>
</html>
```

Output



1.12.3 Date Objects

- This object is used for obtaining the date and time.
- This date and time value is based on computer's local time (system's time) or it can be based on GMT(Greenwich Mean Time).
- Nowadays this GMT is also known as UTC i.e. Universal Co-ordinated Time. This is basically a world time standard.
- Following are the commonly used methods of Date object.

Method	Meaning
getTime()	It returns the number of milliseconds. This value is the difference between the current time and the time value from 1st January 1970.
getDate()	Returns the current date based on computers local time.
getUTCDate()	Returns the current date obtained from UTC.

getDay()	Returns the current day. The day number is from 0 to 6 i.e. from Sunday to Saturday.
getUTCDay()	Returns the current day based on UTC. The day number is from 0 to 6 i.e. from Sunday to Saturday.
getHours()	Returns the hour value ranging from 0 to 23, based on local time.
getUTCHours()	Returns the hour value ranging from 0 to 23, based on UTC timing zone.
getMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on local time.
getUTCMilliseconds()	Returns the milliseconds value ranging from 0 to 999, based on UTC timing zone.
getMinutes()	Returns the minute value ranging from 0 to 59, based on local time.
getUTCMinutes()	Returns the minute value ranging from 0 to 59, based on UTC timing zone.
getSeconds()	Returns the second value ranging from 0 to 59, based on local time.
getUTCSeconds()	Returns the second value ranging from 0 to 59, based on UTC timing zone.
setDate(value)	This function helps to set the desired date using local timing or UTC timing zone.
setHour(hr,minute,second,ms)	This function helps to set the desired time using local or UTC timing zone. The parameters that can be passed to this function are hour,minute,seconds and milliseconds. Only hour parameter is compulsory and rest all are the optional parameters.

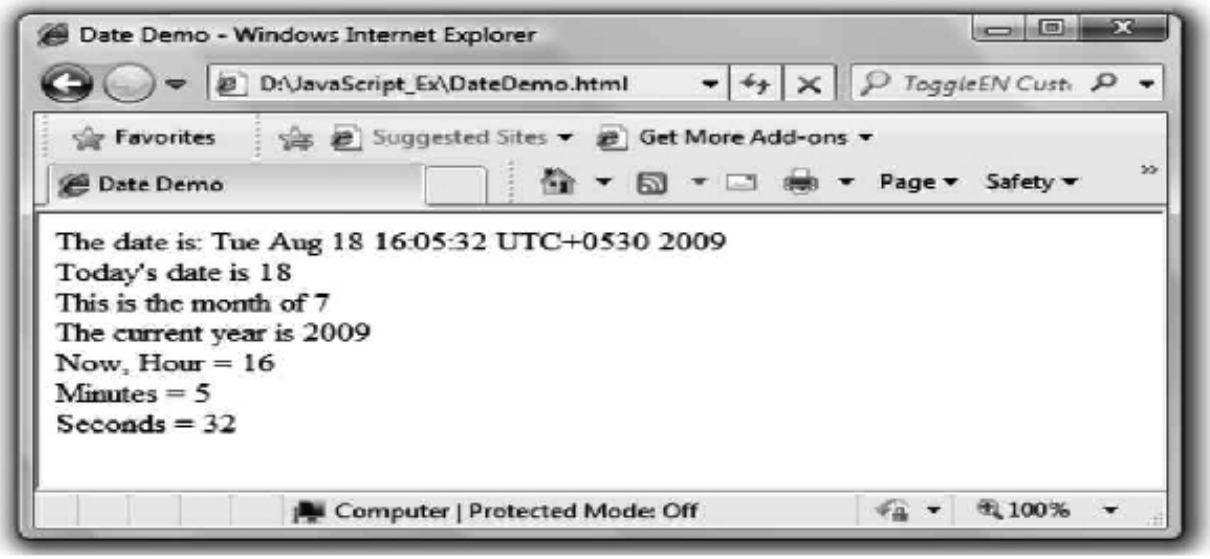
In the following web document we are making use of **Date()** object and some useful methods of it.

JavaScript[DateDemo.html]

```
<!DOCTYPE html>
<html>
<head>
  <title>Date Demo</title>
</head>
<body>
   <script type="text/javascript">
```

```
    var my_date=new Date();
    document.write("The date is: "+my_date.toString()+"<br>");
    document.write("Today's date is "+my_date.getDate()+"<br>");
    document.write("This is the month of "+my_date.getMonth()+"<br>");
    document.write("The current year is "+my_date.getFullYear()+"<br>");
    document.write("Now, Hour = "+my_date.getHours()+"<br>");
    document.write("Minutes = "+my_date.getMinutes()+"<br>");
    document.write("Seconds = "+my_date.getSeconds()+"<br>");
   </script>
</body>
</html>
```

Output



Script Explanation :

1. In the above script we have created an instance *my_date* of the object **Date().**
2. Then using **my_date.toString()** method we can display the current date along with the time.
3. Then using the functions like getDate(), getMonth(), getFullYear() we are displaying the date, month and year separately.
4. Similarly using the functions getHours(), getMinutes() and getSeconds() we can display the current hour, minute and seconds separately.

1.12.4 Boolean Objects

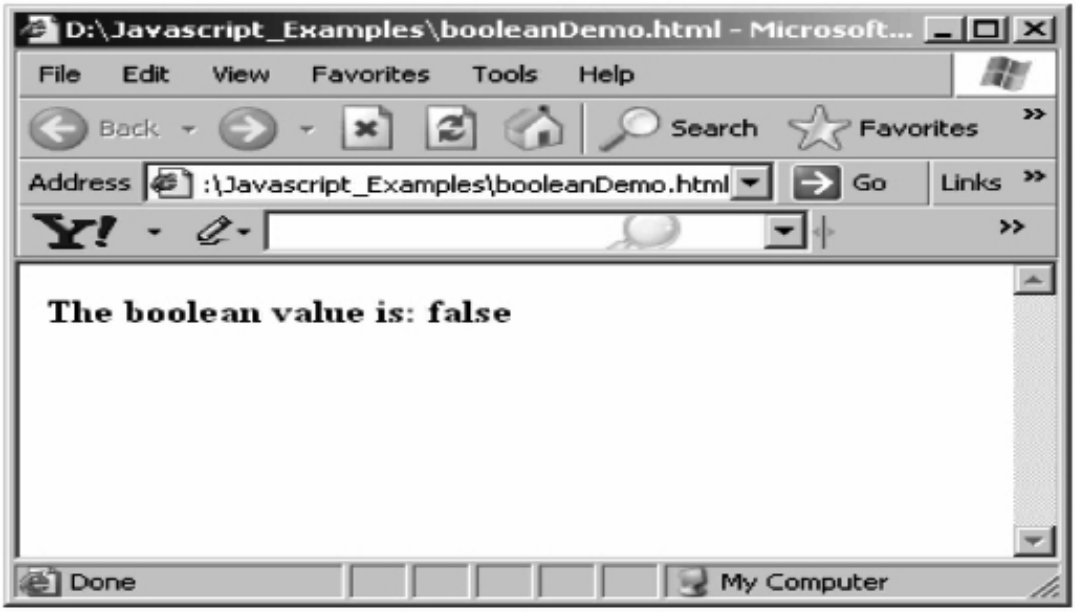
- This object is the simplest kind of object which is used especially when we want to represent **true** and **false** values.

- Here is a simple javascript in which the Boolean type variable is used -

Javascript Program

```
<html>
<body>
<script type="text/javascript">
var temp=new Boolean(false);
document.write("<b>"+"The boolean value is:  ");
document.write(temp.toString());
</script>
</body>
</html>
```

Output



1.12.5 String Objects

- String is a collection of characters.
- In JavaScript using **string** object many useful string related functionalities can be exposed off.
- Some commonly used methods of string object are concatenating two strings, converting the string to upper case or lower case, finding the substring of a given string and so on.
- Here is a listing of some methods of string.

Method	Meaning
concat(str)	This method concatenates the two strings. For example s1.concat(s2) will result in concatenation of string s1 with s2.
charAt(index_val)	This method will return the character specified by value index_val.
substring(begin,end)	This method returns the substring specified by begin and end character.
toLowerCase()	This function is used to convert all the uppercase letters to lower case.
toUpperCase()	This function is used to convert all the lowercase letters to upper case.
valueOf()	This method returns the value of the string.

There is one important property of string object and that is **length**. For example

```
var my_str="Hello";
 var len;
 len=my_str.length;
```

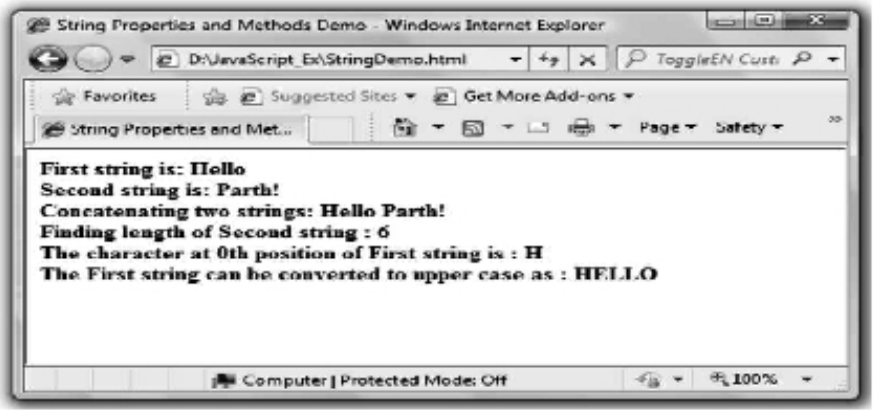
Length of the string "Hello" will be stored in the variable len

Here is sample program in which some methods of string object are used.

JavaScript[StringDemo.html]

```
<!DOCTYPE html>
<html>
<head>
  <title>String Properties and Methods Demo</title>
</head>
<body>
<strong>
   <script type="text/javascript">
    var s1="Hello ";
    var s2="Parth!";
    document.write("First string is: "+s1+"<br>");
    document.write("Second string is: "+s2+"<br>");
    document.write("Concatenating two strings: "+s1.concat(s2)+"<br>");
    document.write("Finding length of Second string : "+s2.length+"<br>");
    document.write("The character at 0th position of First string is : "+s1.charAt(0)+"<br>");
    document.write("The First string can be converted to upper case as : "+s1.toUpperCase());
   </script>
 </strong>
</body>
</html>
```

Output



Example 1.12.1 *Write a script that inputs a line of text, tokenizes it with String method split and displays the tokens in reverse order.*

Solution :

```
<html>
<body>
<script type="text/javascript">
var str=new String("I like writing in javascript");
var a=new Array();
a=str.split(' ');
document.write("<strong>The original string is</strong>"+"<br/>");
for(i=0;i<a.length;i++)
document.write(a[i]+" ");
document.write("<br/>");
document.write("<strong>The reversed string is</strong>"+"<br/>");
for(i=a.length-1;i>=0;i--)
document.write(a[i]+" ");
</script>
</body>
</html>
```

1.12.6 Object Creation and Modification

- The web designer can create an object and can set its properties as per his requirements.
- The object can be created using new expression.
- Initially the object with no properties can be set using following statements

```
Myobj=new Object();
```

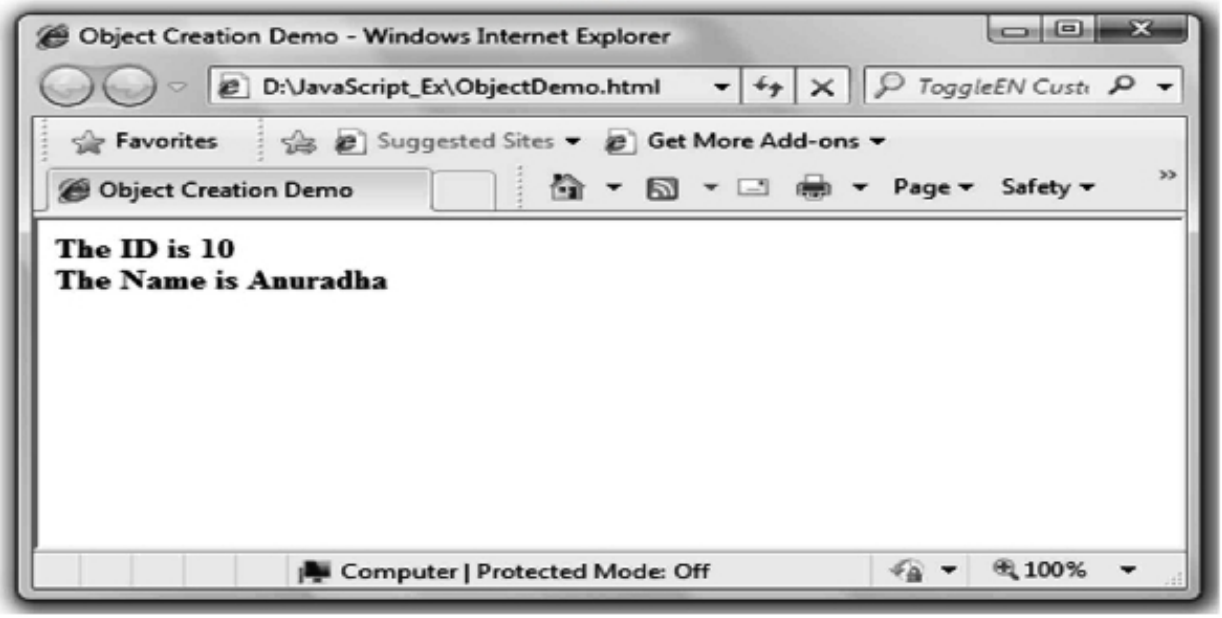
- Then using dot operator we can set the properties for that object.
- The object can then be modified by assigning the values to this object.

JavaScript[ObjectDemo.html]

```
<!DOCTYPE html>
<html>
<head>
  <title>Object Creation Demo</title>
</head>
<body>
```

```
<strong>
  <script type="text/javascript">
    var Myobj;
    var n,i;
    //creating an object
    Myobj=new Object();
    //setting the propertis for newly created object
    Myobj.id=10;
    Myobj.name="Anuradha";
    document.write("The ID is "+Myobj.id+"<br/>");
    document.write("The Name is "+Myobj.name);
  </script>
 </strong>
</body>
</html>
```

Output



Alternative method of creating an object and modifying the properties is as given below -

```
var Myobj={name:"Anuradha",id:10};
```

Then using **document.write** statement we can display the property values as follows -

```
document.write("The ID is "+Myobj.id+"<br/>");
document.write("The Name is "+Myobj.name);
```

The property of created object can be deleted using an expression **delete**. Normally the property of an object is deleted in order to free the allocated memory so that this memory can be reused by other process.

1.13 DOM

1.13.1 Definition of DOM

- The Document Object Modeling (DOM) is for defining the standard for accessing and manipulating HTML,XML and other scripting languages.
- It is the **W3C recommendation** for handling the structured documents.
- Normally the structured information is provided in XML, HTML and many other documents.
- Hence DOM provides the standard set of programming interfaces for working with XML and XHTML and JavaScript.

What is DOM?

Document Object Model (DOM) is a set of platform independent and language neutral Application Programming Interface (API) which describes how to access and manipulate the information stored in XML,HTML and JavaScript documents.

1.13.2 DOM Tree

- Basically DOM is an **Application Programming Interface (API)** that defines the interface between HTML document and application program. That means, suppose application program is written in Java and this Java program wants to access the elements of HTML web document then it is possible by using a set of Application Programming Interfaces (API) which belongs to the DOM.
- The DOM contains the **set of interfaces** for the document tree node type. These interfaces are similar to the Java or C++ interfaces. They have **objects, properties and methods** which are useful for respected node type of the web document.
- The documents in DOM are represented using a tree like structure in which every element is represented as a node. Hence the tree structure is also referred as DOM tree.
- **For example :** Consider following XHTML document.

```
<html>
    <head>
```

```
    <title>This is My Web Page </title>
  </head>
  <body>
    <h1>Hello Friends </h1>
    <h2>How are you?</h2>
    </h3>See you</h3>
  </body>
  </html>
```

- The DOM tree will be

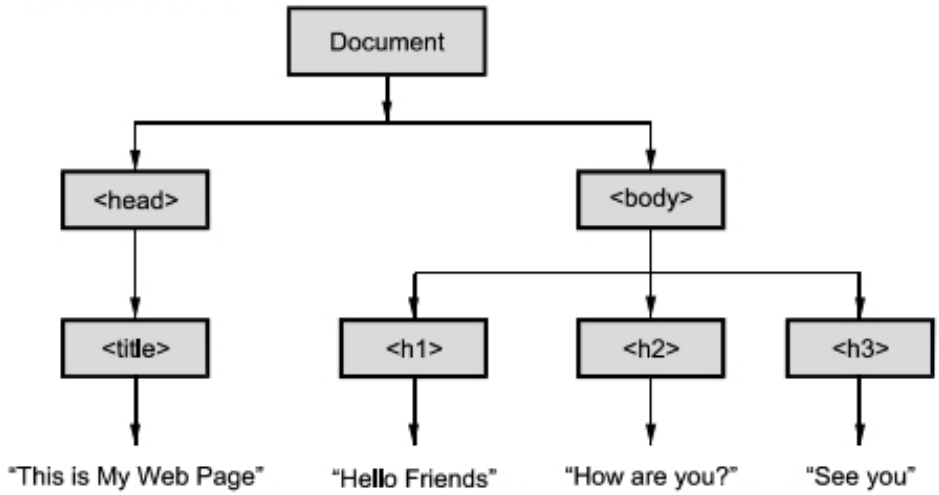


Fig. 1.13.1 DOM structure for simple web document

- We can describe some **basic terminologies** used in DOM tree as follows -
  1. Every element in the DOM tree is called **node**.
  2. The topmost single node in the DOM tree is called the **root**.
  3. Every child node must have a parent **node**.
  4. The bottommost nodes that have no children are called **leaf nodes**.
  5. The nodes that have the common parent are called **siblings**.

1.13.3 Using DOM Methods

We can access or change the contents of document by using various methods. Some commonly used properties and methods of DOM are as follows :

Methods

Method	Meaning
getElementById	This method is used to obtain the specific element which is specified by some id within the script.
createElement	This method is used to create an element node.

createTextNode	Useful for creating a text node.
createAttribute	Useful for creating attribute.
appendChild	For adding a new child to specified node, this method is used.
removeChild	For removing a child node of a specific node, this method is used.
getAttribute	This method is useful for returning the specified attribute value.
setAttribute	This method is useful for setting or changing the specified attribute to the specified value.

Properties

Property	Meaning
attributes	This property is used to get the attribute nodes of the node.
parentNode	This DOM property is useful for obtaining the parent node of the specific node.
childNodes	This DOM property is useful for obtaining the child nodes of the specific node.
innerHTML	It is useful for getting the text value of a node.

1.13.3.1 Accessing Elements using DOM

- There are several ways by which we can access the elements of the web document.
- To understand these methods of accessing we will consider one simple web document as follows.

```
<html >
<head>
    <title>This is My Web Page </title>
</head>
<body>
<form name="form1">
        <input type="text" name="myinput"/>
</form>
</body>
</html>
```

Method 1

- Every XHTML document element is associated with some address. This address is called **DOM address.**
- The document has the collection of **forms** and **elements.** Hence we can refer the text box element as

```
var Dom_Obj=document.forms[0].elements[0];
```

- But this is not the suitable method of addressing the elements. Because if we change the above script as

```
...
<form name="form1">
        <input type="button" name= "mybutton"/>
        <input type="text" name="myinput"/>
</form>
...
```

then index reference gets changed. Hence another approach of accessing the elements is developed.

Method 2

- We can access the desired element from the web document using JavaScript method **getElementById**. The element access can be made as follows -

```
var Dom_Obj=document.getElementById("myinput");
```

- But if the element is in particular group, that means if there are certain elements on the form such as radio buttons or check boxes then they normally appear in the groups. Hence to access these elements we make use of its index. Consider the following code sample

```
<form id="Food">
        <input type="checkbox" name="vegetables" value="Spinach" />Spinach
        <input type="checkbox" name="vegetables" value="FenuGreek" />FenuGreek
        <input type="checkbox" name="vegetables" value="Cabbage" />Cabbage
</form>
```

- For getting the values of these checkboxes we can write following code.

```
var Dom_obj=document.getElementById("Food");
for(i = 0 ; i < Dom_Obj.vegetables.length ; i++)
        document.write(Dom_Obj.vegetables[i]+"<br/>");
```

1.13.3.2 Modifying Elements using DOM

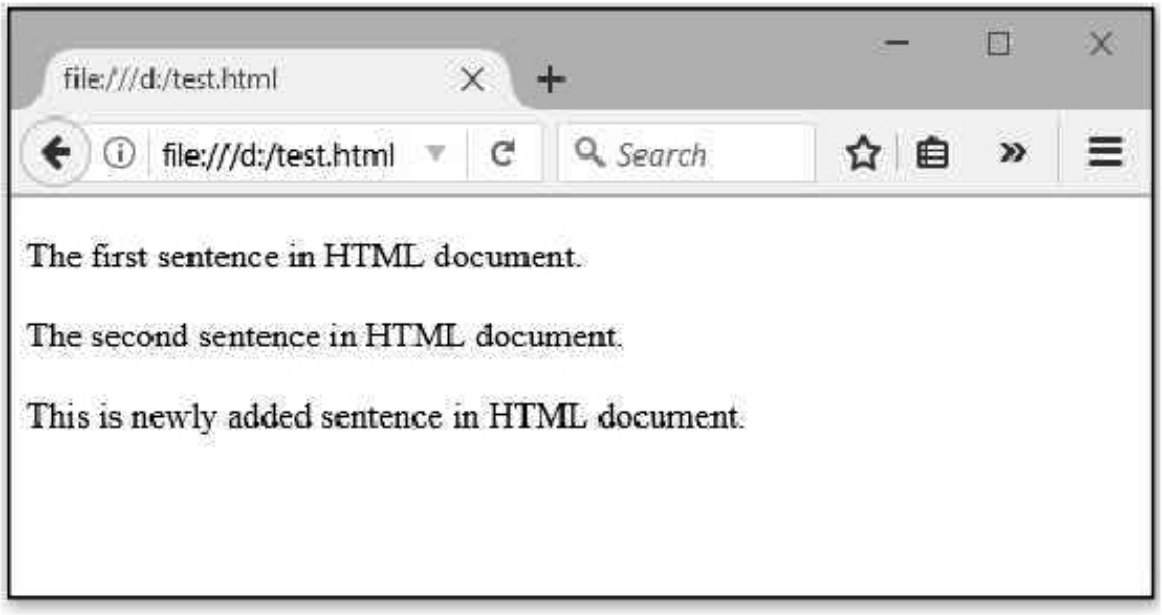
Appending an element Using DOM

We can add new node in the HTML document using the DOM method **appendChild** method

Following example illustrates the idea

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

```
<div id="div1">
<p id="p1">The first sentence in HTML document.</p>
<p id="p2">The second sentence in HTML document.</p>
</div>

<script>
var pnode = document.createElement("p");
var node = document.createTextNode("This is newly added sentence in HTML document.");
pnode.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(pnode);
</script>
</body>
</html>
```



Inserting an element Using DOM

We can insert a node in between the nodes by using **appendChild** and **insertBefore** method. For example

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

<div id="div1">
<p id="p1">First Sentence</p>
<p id="p2">Second Sentence</p>
```

```
</div>

<script>
var pnode = document.createElement("p");
var node = document.createTextNode("Newly Added Sentence!!!");
pnode.appendChild(node);
var element = document.getElementById("div1");
var nextnode = document.getElementById("p2");
element.insertBefore(pnode,nextnode);
</script>

</body>
</html>
```

Output

Removing an element Using DOM

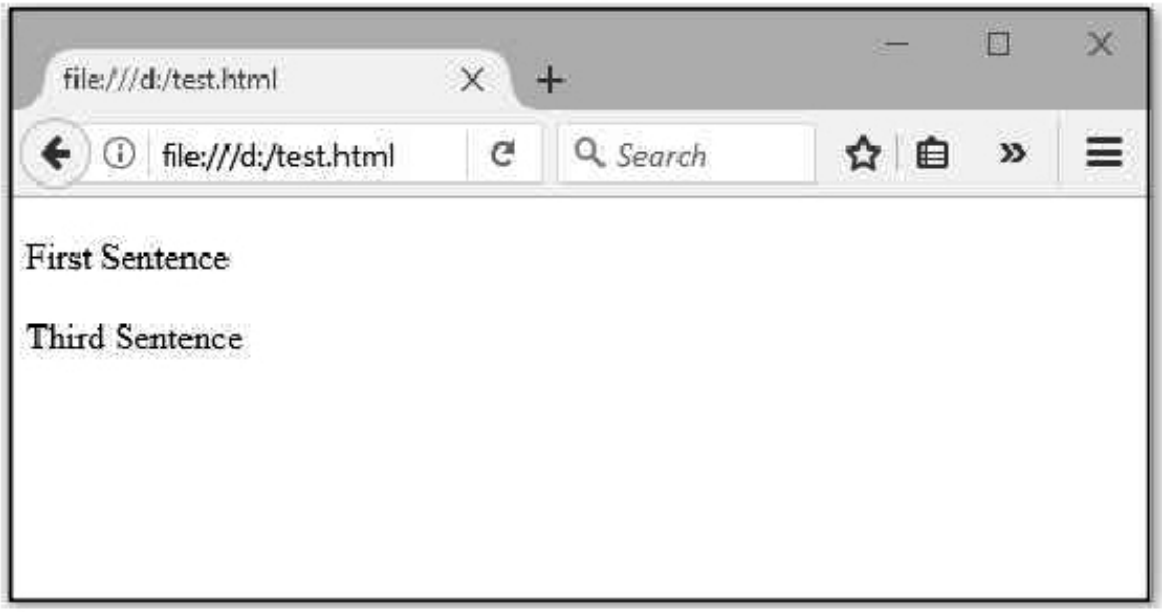
We can remove or delete particular from HTML document using DOM's **removeChild** method.

Here is the illustration

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

<div id="div1">
<p id="p1">First Sentence</p>
<p id="p2">Second Sentence</p>
```

```
<p id="p3">Third Sentence</p>
</div>

<script>
var parentnode = document.getElementById("div1");
var node = document.getElementById("p2");
parentnode.removeChild(node);
</script>
</body>
</html>
```

Output



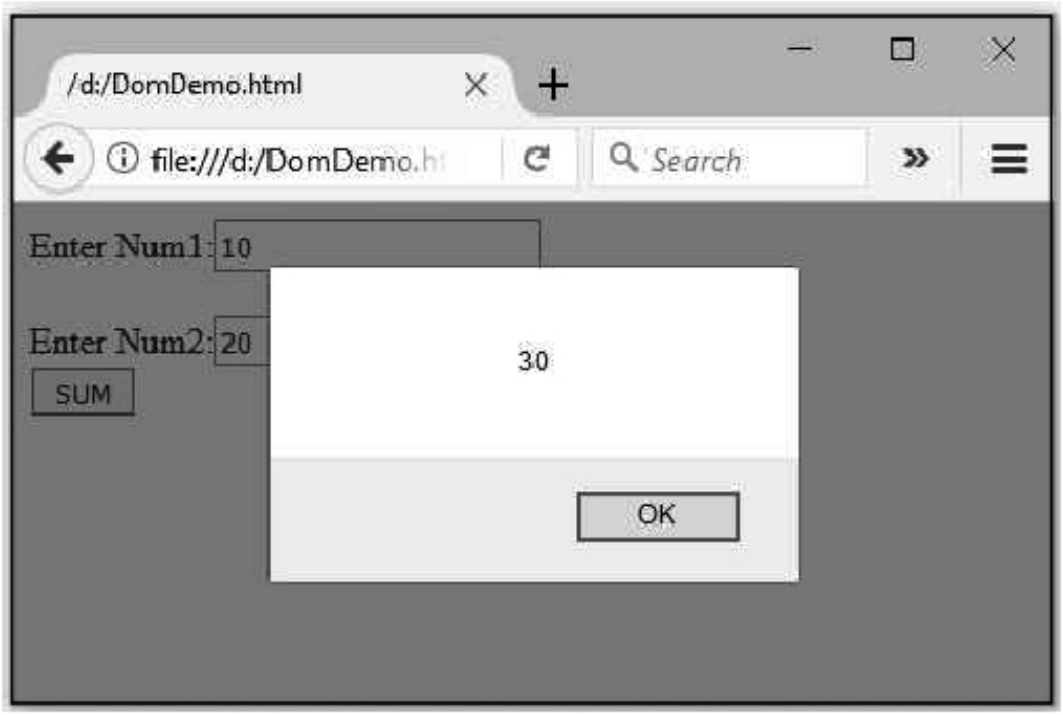
Example 1.13.1 *Write a JavaScript to display sum of two elements. Make use of appropriate DOM method.*

Solution :

```
<!Doctype html>
<html>
<head>
<script type="text/javascript">
function getSum()
{
    var number1=document.getElementById("number1").value;
    var number2=document.getElementById("number2").value;
    var num1=Number(number1);
var num2=Number(number2);
    alert(num1+num2);
```

```
}
</script>
</head>
<body>
<form>
Enter Num1:<input type="text" id="number1" /><br/> <br/>
Enter Num2:<input type="text" id="number2" /><br/>
<input type="button" value="SUM" onclick="getSum()"/>
</form>
</body>
</html>
```

Output



Example 1.13.2 *Write a JavaScript to welcome the user by his/her name when he enters his/her name in textbox. Make use of appropriate DOM method.*

Solution :

```
<!Doctype html>
<html>
<head>
<script type="text/javascript">
function display()
{
    var name=document.getElementsByName("user");
alert("Welcome "+name[0].value);
```

```
}
</script>
</head>
<body>
<form>
Enter Name:<input type="text" name="user" /><br/> <br/>
<input type="button" value="Greet" onclick="display()"/>
</form>
</body>
</html>
```

Output

1.14 Event Handling

- **Definition of Event :** Event is an activity that represents a change in the environment. For example mouse clicks, pressing a particular key of keyboard represent the events. Such events are called **intrinsic events.**
- **Definition of Event Handler :** Event handler is a script that gets executed in response to these events. Thus event handler enables the web document to respond the user activities through the browser window.
- JavaScript support this special type of programming in which events may occur and these events get responded by executing the event handlers. This type of programming is called event-driven programming.

- Events are specified in lowercase letters and these are case-sensitive.
- **Event Registration :** The process of connecting event handler to an event is called event registration. The event handler registration can be done using two methods -
  - Assigning the tag attributes
  - Assigning the handler address to object properties.

Internet Programming

On occurrence of events the tag attribute must be assigned with some user defined functionalities. This will help to execute certain action on occurrence of particular event.

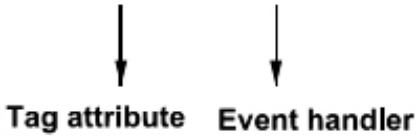
Commonly used events and tag attributes are enlisted in the following table -

Events	Intrinsic event attribute	Meaning	Associated tags
blur	onblur	Losing the focus.	<button> <input> <a> <textarea> <select>
change	onchange	On occurrence of some change.	<input> <textarea> <select>
click	onclick	When user clicks the mouse button.	<a> <input>
dblclick	ondblclick	When user double clicks the mouse button.	<a> <input> <button>
focus	onfocus	When user acquires the input focus.	<a> <input> <select> <textarea>
keyup	onkeyup	When user releases the key from the keyboard.	Form elements such as input,button,text,textarea and so on.
keydown	onkeydown	When user presses the key down	Form elements such as input,button,text,textarea and so on.

keypress	onkeypress	When user presses the key.	Form elements such as input,button,text,textarea and so on.
mousedown	onmousedown	When user clicks the left mouse button.	Form elements such as input,button,text,textarea and so on.
mouseup	onmouseup	When user releases the left mouse button.	Form elements such as input,button,text,textarea and so on.
mousemove	onmousemove	When user moves the mouse.	Form elements such as input,button,text,textarea and so on.
mouseout	onmouseout	When the user moves the mouse away from some element.	form elements such as input,button,text,textarea and so on.
mouseover	onmouseover	When the user moves the mouse away over some element.	Form elements such as input,button,text,textarea and so on.
load	onload	After getting the document loaded.	<body>
reset	onreset	When the reset button is clicked.	<form>
submit	onsubmit	When the submit button is clicked.	<form>
select	onselect	On selection.	<input> <textarea>
unload	onunload	When user exits the document.	<body>

The use of these tag attributes for handling the events is illustrated by following code sample

```
<input type ="button" name="My_button"
onclick="My_fun()"; />
```



That means when the user clicks the **button** then as an event handler a user defined function **My_fun()** gets called. Basically in this function user writes the instructions that need to be executed on the button click event.

Example 1.14.1 *Discuss the properties of mouse events associated with DOM2 with an example.*

Solution : Various types of mouse events associated with DOM2 are click, mousedown, mouseup, mouseout and mouseover.

Events	Intrinsic event attribute	Meaning	Associated tags
mousedown	onmousedown	When user clicks the left mouse button.	Form elements such as input,button,text,textarea and so on.
mouseup	onmouseup	When user releases the left mouse button.	Form elements such as input,button,text,textarea and so on.
mousemove	onmousemove	When user moves the mouse.	Form elements such as input,button,text,textarea and so on.
mouseout	onmouseout	When the user moves the mouse away from some element	form elements such as input,button,text,textarea and so on.
mouseover	onmouseover	When the user moves the mouse away over some element	Form elements such as input,button,text,textarea and so on.
click	onclick	When user clicks the mouse button.	<a> <input>
dblclick	ondblclick	When user double clicks the mouse button.	<a> <input> <button>

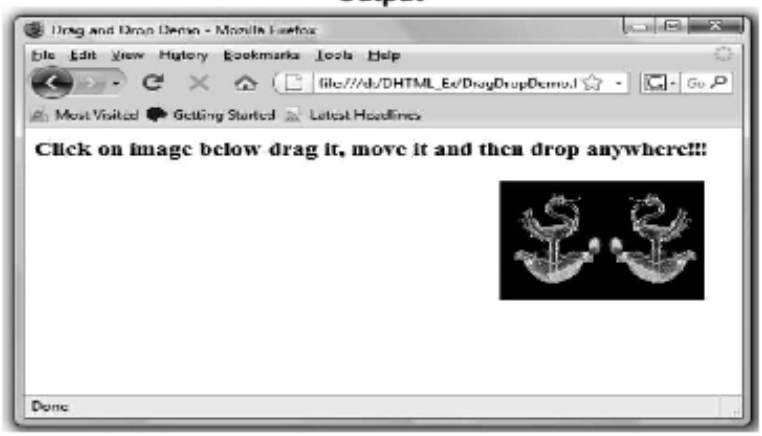
Following is an example in which image can be dragged and dropped anywhere in the HTML document. Accordingly the co-ordinates will be displayed.

```
<html>
<head>
 <title> Drag and Drop Demo</title>
 <script type = "text/javascript">
  var X_offset, Y_offset, Item;
  function Catch_function(e)
```

```
  {
     Item = e.currentTarget;
     var posX = parseInt(Item.style.left);
     var posY = parseInt(Item.style.top);
     X_offset = e.clientX - posX;
     Y_offset = e.clientY - posY;
     document.addEventListener("mousemove", Move_function, true);
     document.addEventListener("mouseup", Drop_function, true);
     e.stopPropagation();
     e.preventDefault();

  }
  function Move_function(e)
  {
     Item.style.left = (e.clientX - X_offset) + "px";
     Item.style.top = (e.clientY - Y_offset) + "px";
     e.stopPropagation();
  }
  function Drop_function(e)
  {
     document.removeEventListener("mouseup", Drop_function, true);
     document.removeEventListener("mousemove", Move_function, true);
     e.stopPropagation();
  }
 </script>
</head>
<body>
 <h3>
  Click on image below-drag it, move it and then drop anywhere!!!
   <br /><br />
   <img src="lamp1.jpg"
   style = "position: absolute;
   top: 100px;
   left: 100px;
   border:2px solid;"
   onmousedown = "Catch_function(event);" />
 </h3>
</body>
</html>
```

Registering the mouse event with the help of **addEventListener** function.

Output



1.14.1 Handling Events from the Body Elements

To understand how events works in JavaScript let us put some Form components on the JavaScript. The **onload** event gets activated as soon as the web page gets loaded in the browser's window. Following script along with its output illustrate the **onload** tag attribute.

JavaScript[OnloadDemo.html]

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Demo of onload Tag Attibute</title>
 <script type="text/javascript">
  function my_fun()
  {
  //This message will be displayed on page loading



 alert("Welcome");
  }
 </script>
</head>
<body onload="my_fun()">
</body>
</html>
```

When web document gets loaded on the browser window then **my_fun()** will be called

Output

1.15 Error Handling

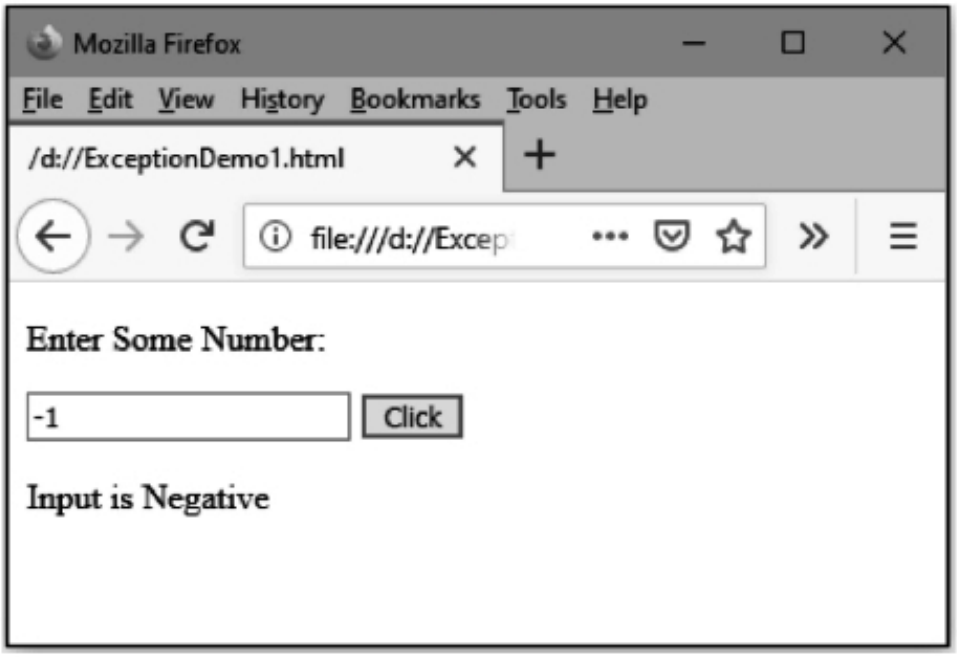
- **Definition :** Exception handling is a mechanism that handles the runtime errors so that the normal flow of the application can be maintained.
- Basically an exception is an event that disrupts the normal flow of the program. It is **an object** which is **thrown at runtime**.
- The exception handling in JavaScript can be performed with the help of following types of statements –
  1. The **try** statement lets you test a block of code for errors.
  2. The **catch** statement lets you handle the error.
  3. The **throw** statement lets you create custom errors.
  4. The **finally** statement lets you execute code, after try and catch, regardless of the result.
- Basic syntax of try-catch block is

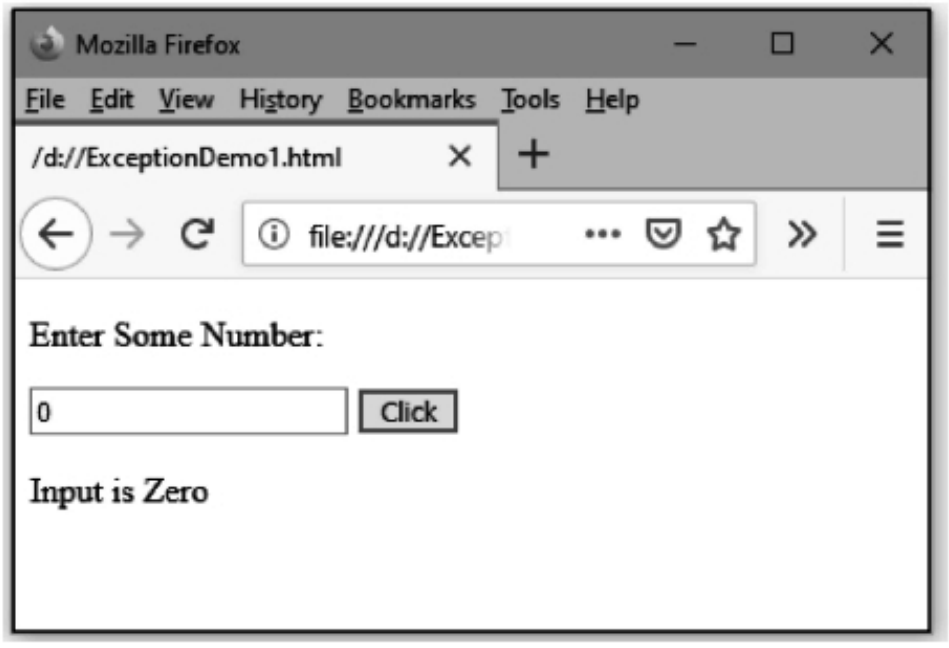
```
try {
    Block of code to try
}
catch(err) {
Block of code to handle errors
}
```

- When error occurs, JavaScript will normally throw an exception so that appropriate error message can be displayed and it will avoid a crash of a program.
- Following example illustrates the exception handling mechanism in JavaScript

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

<p>Enter Some Number:</p>

<input id="demo" type="text">
<button type="button" onclick="GetPositiveNumber()">Click</button>
<p id="myID"></p>

<script>
    function GetPositiveNumber() {
    var message, num;
    message = document.getElementById("myID");
    message.innerHTML = "";
    num = document.getElementById("demo").value;
    try {
            if(num == "")
                throw "empty";
            if(isNaN(num))
                throw "not a number";
            num = Number(num);
            if(num < 0)
                throw "Negative";
            if(num == 0)
                throw "Zero";
    }
    catch(err) {
            message.innerHTML = "Input is " + err;
    }
}
</script>
</body>
</html>
```

Output



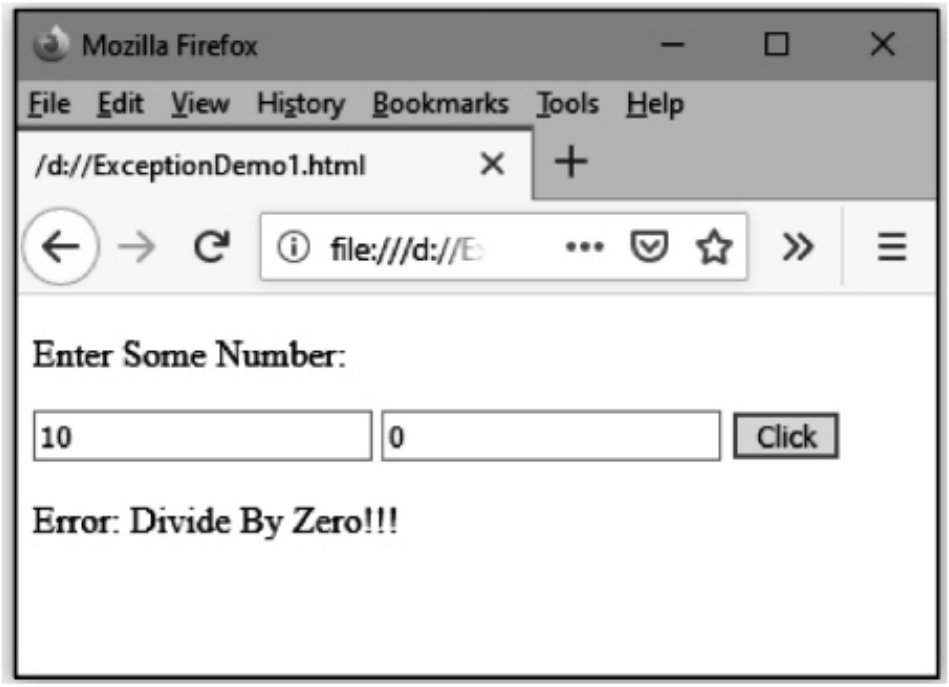


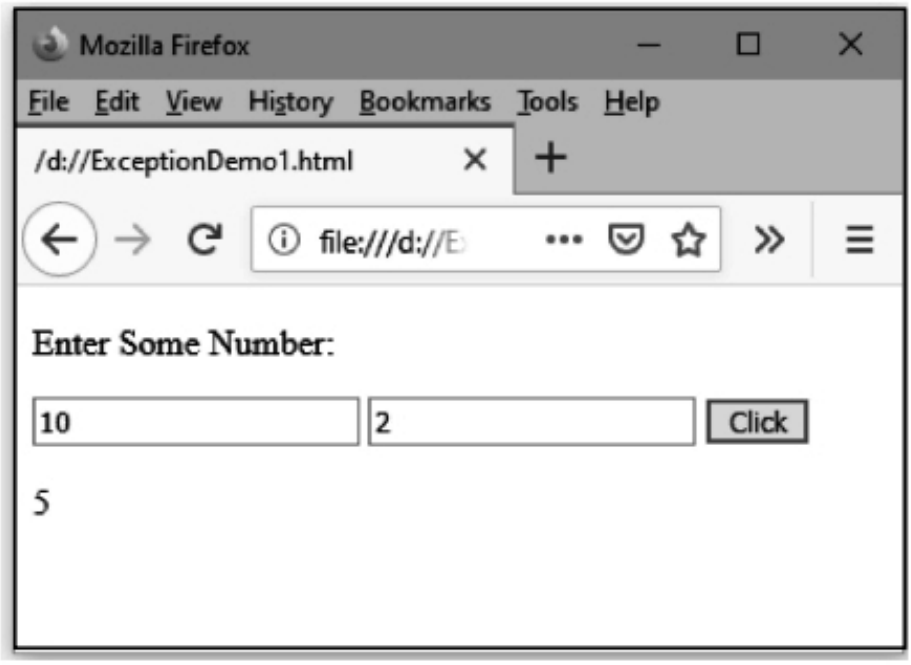
Example 1.15.1 *Write a JavaScript to handle divide by zero error using exception handling mechanism.*

Solution :

```
<!DOCTYPE html>
<html>
<body>
    <p>Enter Some Number:</p>
    <input id="numerator" type="text">
    <input id="denominator" type="text">
    <button type="button" onclick="divide()">Click</button>
    <p id="myID"></p>

<script>
    function divide() {
        var n,result, message, num;
        message = document.getElementById("myID");
        message.innerHTML = "";
        d = document.getElementById("denominator").value;
        try {
            if(d == "")
                throw "empty";
            if(isNaN(d))
                throw "not a number";
            num = Number(d);
            if(num == 0)
                throw "Divide By Zero!!!";
            else
            {
                n = document.getElementById("numerator").value;//obtaining numerator
                n=Number(n);//converting text string to number
                ans=n/num;//performing division
                message.innerHTML = "="+ans;
            }
        }
        catch(err) {
          message.innerHTML = "Error: " + err;
    }
}
</script>
</body>
</html>
```

Output





1.16 Validators

- Various control objects are placed on the form. These control objects are called **widgets.**

- These widgets used in JavaScript are – **Textbox, Push button, Radio button, Checkbox** and so on.
- In JavaScript the **validation of these widgets** is an important task.

Let us understand the validation of form elements with the help of examples –

Example 1.16.1 *Write a JavaScript for password verification.*

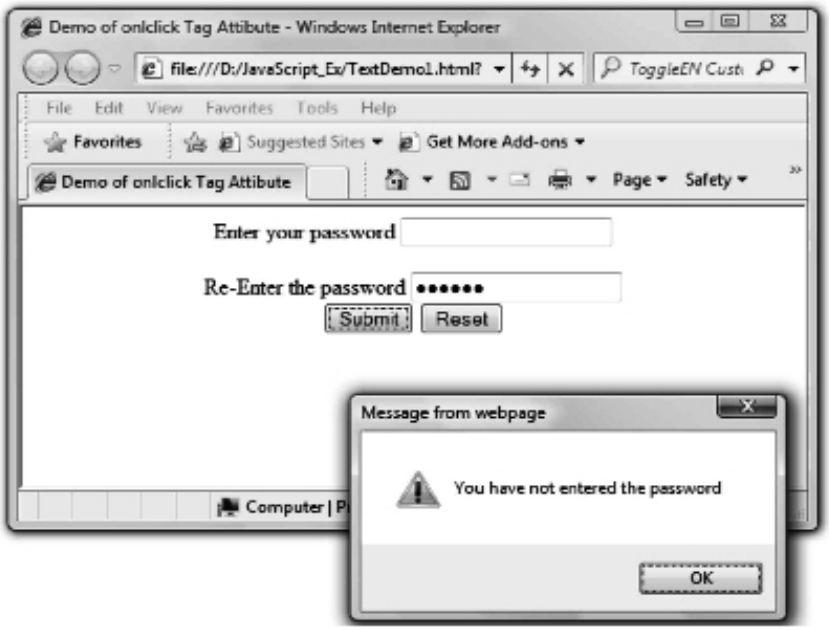
Solution :

```
<!DOCTYPE html>
<html>
<head>
 <title>Demo of onlclick Tag Attibute</title>
<script type="text/javascript">
 function my_fun()
 {
  var mypwd=document.getElementById("pwd");
  var my_re_pwd=document.getElementById("re_pwd");
  if(mypwd.value=="")
  {
    alert("You have not entered the password");
    mypwd.focus();
    return false;
  }
  if(mypwd.value!=my_re_pwd.value)
  {
    alert("Password is not verified, Re-enter both the passwords");
    mypwd.focus();
    mypwd.select();
    return false;
  }
  else
  {
    alert("Congratulations!!!");
    return true;
  }
 }
</script>
</head>
<body>
 <center>
 <form id="form1">
```

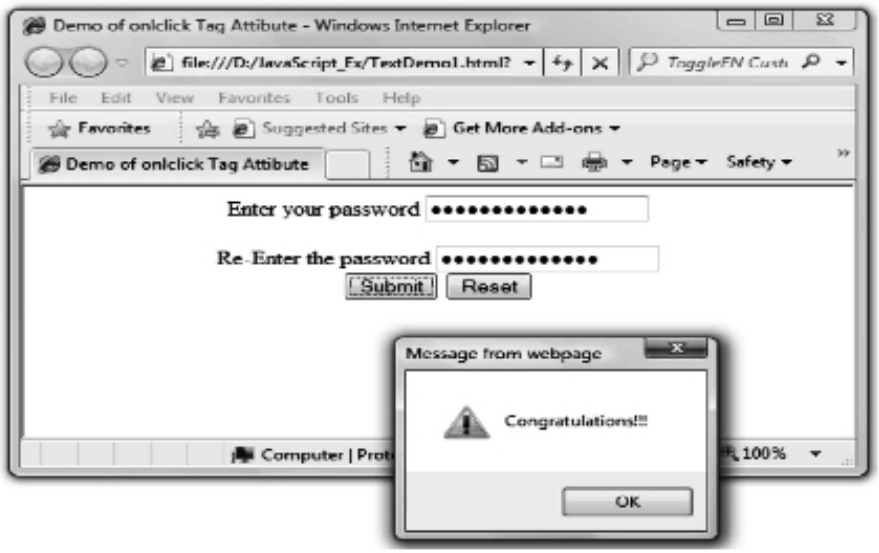
```
  <label> Enter your password
  <input type="password" value="" id="pwd" />
  </label>
  <br/><br/>
  <label> Re-Enter the password
  <input type="password" value="" id="re_pwd" onblur="my_fun();"/>
  </label><br/>
  <input type="submit" value="Submit" name="submit" onsubmit="my_fun();"/>
   <input type="reset" value="Reset" name="reset"/><br/>
 </form>
</center>
</body>
</html>
```

Output(Run1)

Output(Run2)



Output(Run3)



Example 1.16.2 *Write JavaScript to validate a for consisting of Name, Age, Address, EmailID, hobby (check box), Gender(radio box), country(Drop down menu).*

Solution : ApplicationForm.html

```
<html>
<head>
<title>The Student Registration Form</title>
<script type=text/javascript>
```

```
function validate()
{


        var i;
        var name_str=document.my_form.name;
        var phoneID=document.my_form.ph_txt;
        var ph_str=document.my_form.ph_txt.value;
        var str=document.my_form.Email_txt.value;
        if((name_str.value==null)||(name_str.value==""))
        {
        alert("Enter some name")
        return false
        }
       if(document.my_form.Age_txt.value=="")// validating age
        {
        alert("Enter Some Age")
        return false
        }
              if((document.my_form.Age_txt.value<"5")&&(document.my_form.Age_txt.value>"21"))
        {
        alert("Invalid Age")
        return false
        }

        if(ph_str.length<1 ||ph_str.length>11)
        {
        alert("Invalid length of Phone Number")
        return false
        }
        for (i = 0; i < ph_str.length; i++)
        {
           var ch = ph_str.charAt(i);
           if (((ch < "0") || (ch > "9"))){
        alert("Invalid Phone Number")

        phoneID.focus()
         return false
```

Validating Name

Validating Phone Number

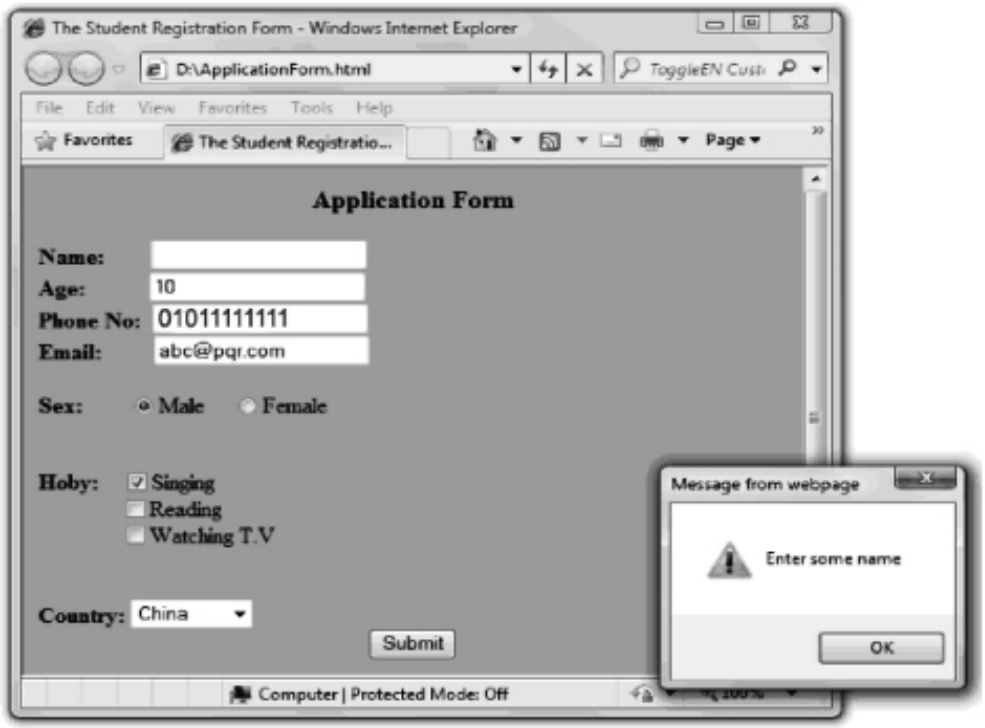
```
        }
}
        var index_at=str.indexOf("@")
        var len=str.length
        var index_dot=str.indexOf(".")
        var emailID=document.my_form.Email_txt
        if ((emailID.value==null)||(emailID.value==""))
        {
        alert("Please Enter your Email ID")
        emailID.focus()
        return false
        }
        if (str.indexOf("@")==-1)
        {
          alert("Invalid E-mail ID")
          return false
        }
       if (str.indexOf(".")==-1 || str.indexOf(".")==0
                || str.indexOf(".")==index_at)
        {
           alert("Invalid E-mail ID")
           return false
        }

         if (str.indexOf("@",(index_at+1))!=-1)
         {
           alert("Invalid E-mail ID")
           return false
         }
         if (str.indexOf(" ")!=-1)
    {
           alert("Invalid E-mail ID")
           return false
         }
    if (!document.my_form.group1[0].checked && !document.my_form.group1[0].checked)
        {
           alert("Please Select Sex");
           return false;
        }
    if (!document.my_form.group1[0].checked && !document.my_form.group1[0].checked)
        {
           alert("Please Select Sex");
```

Validating Email ID

Validating Email ID

```
            return false;
          }
          return true

}
</script>
</head>
<body bgcolor=aqua>
<center><h3>Application Form</h3></center>
<form name=my_form onsubmit=validate()>
<strong>Name: </strong>
<input type=text name=name><br/>

<strong>Age: </strong>
<input type=text name=Age_txt><br/>
<strong>Phone No:</strong>
<input type=text name=ph_txt><br/>
<strong>Email: </strong>
<input type=text name=Email_txt><br/><br/>
<strong>Sex:  </strong>
<input type="radio" name="group1" value="Male">Male
<input type="radio" name="group1" value="Female">Female<br/><br/><br/>
<strong>Hoby: </strong>
<input type="checkbox" name ="option1" value="Singing">Singing<br/>
<input type="checkbox" name ="option1" value="Reading">Reading<br/>
<input type="checkbox" name ="option1" value="T.V.">Watching T.V<br/>
<br/><br/>
<strong>Country:</strong>
<select name="My_Menu">
<option value="India">India</option>
<option value="China">China</option>
<option value="Shrilanka">Shrilanka</option>
</select>
<center>
<input type=submit value=Submit></br>
</center>

</body>
</html>
```

Output



Example 1.16.3 *Design a HTML form for validation the users with fields user name and password and ok button which should receive the input from the user and responses as authorized or invalid user name or invalid password.*

Solution :

```
<head>
  <title>HTML FORM</title>
  <script type=text/javascript>
   function validate()
    {
     var user=document.form1.username;
     var pass=document.form1.password;
    if((user.value==null)||(user.value==""))
    {
     alert("Enter some user name");
     }
   if((pass.value==null)||(pass.value==""))
    {
     alert("Enter some password");

     }
     if(user.value=="admin")
```

```
        {
        alert("Valid user name");

        }
        if(pass.value=="password")
        {
        alert("Valid password");
        }
        else
        alert("Invalid username/password");
       }
   </script>
  </head>
  <body>
   <form id="form1" name="form1" onsubmit=validate()>
   <table width="510" border="0" align="center">
   <tr>   <td>Username:</td>
   <td><input type="text" name="username"/></td>
   </tr>
   <tr>    <td>Password</td>
   <td><input type="password" name="password"/></td>
   </tr>
   <tr>
   <td> </td>
   <td><input type="submit" name="button" value="OK" /></td>
   </tr>
   </table>
   </form>
  </body>
</html>
```

Example 1.16.4 *Write a Javascript to validate radio button, operator field and check box.*

Solution :

```
JavaScript Program
<html>
<head>
<script LANGUAGE="JavaScript">

function ValidateForm(form)
{

if ((form.gender[0].checked == false ) && ( form.gender[1].checked == false ))
 { alert ( "Please choose your Gender: Male or Female" );
   return false;
```

Validating Radio Button

```
}
```

Validating Checkbox

```
if ((form.status[0].checked == false ) && ( form.status[1].checked == false ))
{ alert ( "Please choose your choice: AC or non AC" );
  return false;
}
if ( form.course.selectedIndex == 0 )
{ alert ( "Please select Some Course." );
  return false;
}
return true;
}
</script>
</head>
<body>
<br>
<form>
 Your Gender: <input type="radio" name="gender" value="Male"> Male
 <input type="radio" name="gender" value="Female"> Female
 <br/>
 Choice: <input type="checkbox" name="status">AC
 <input type="checkbox" name="status"> Non AC
 <br/>
 Course:
 <select name="course">
   <option value="">Select an Option:</option>
   <option value="Computer">Computer</option>
   <option value="Mechanical">Mechanical</option>
   <option value="E&Tc">E&Tc</option>
 </select>
 <br/><br/>
<input type="button" name="SubmitButton" value="Submit"
 onClick="ValidateForm(this.form)">
 <input type="reset" value="Reset">
</form>
</body>
</html>
```

Validating Option

1.17 Asynchronous Programming

1.17.1 Introduction to AJAX

- AJAX is a Asynchronous **Java Script** and XML
- Here
  - **Asynchronous** means, the execution of script does not disturb the user's work.
  - **JavaScript** because, it makes use of Javascripting to do the actual work.
  - **XML** because along with JavaScript the XML is also supported to perform the given task in AJAX.
- It is not a new programming language but it is a kind of web document which adopts certain standards.
- AJAX allows the developer to exchange the data with the server and updates the part of web document without reloading the web page.

1.17.2 Architecture

- When user makes a request, the browser creates a object for the **HttpRequest** and a request is made to the server over an internet.
- The Server processes this request and sends the required data to the browser.
- At the browser side the returned data is processed using JavaScript and the web document gets updated accordingly by sending the appropriate response.
- Following Fig. 1.17.1 illustrates this working.

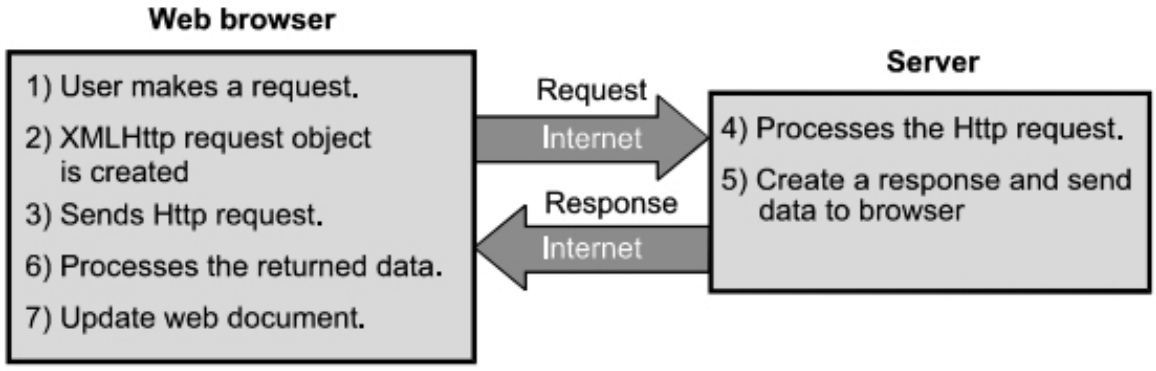


Fig. 1.17.1 Working of AJAX

Similarities and Differences

Sr.No.	Traditional Web Application Architecture	Ajax Based Web Application Architecture
1.	The web applications architectures are based on HTTP request response protocol.	
2.	At the client side the browser client has only one component and that is – user interface.	At the client side the browser client has user interface and AJAX Engine due to which the client gets quick response from the server.
3.	The architecture is based on client server communication.	
4.	It makes use of synchronous communication. The client has to wait to get the response from the server then only client can make the request to the server. It is start-stop-start-stop kind of communication.	By adding a new layer of AJAX Engine at the client side, it eliminates the start-stop-start-stop nature of communication and makes the communication asynchronous.
5.	With traditional web application architecture user cannot get rich user interface experience.	With AJAX architecture user gets rich user interface experience.
6.	It is less responsive.	It is more responsive.
7.	Building web application is simple.	The development time is more while designing the Ajax based web application.
8.	Limited use of JavaScript, CSS and XML technologies. In fact - Most user actions in the interface trigger an HTTP request back to a web server.	Extensive use of JavaScript, CSS and XML - i) standards-based presentation using XHTML and CSS; ii) Dynamic display and interaction using the Document Object Model(DOM); iii) Data interchange and manipulation using XML and XSLT; iv) Asynchronous data retrieval using XMLHttpRequest; v) JavaScript binding everything together

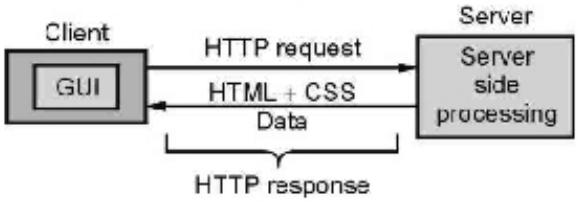


Fig. 1.17.2 Traditional web application architecture

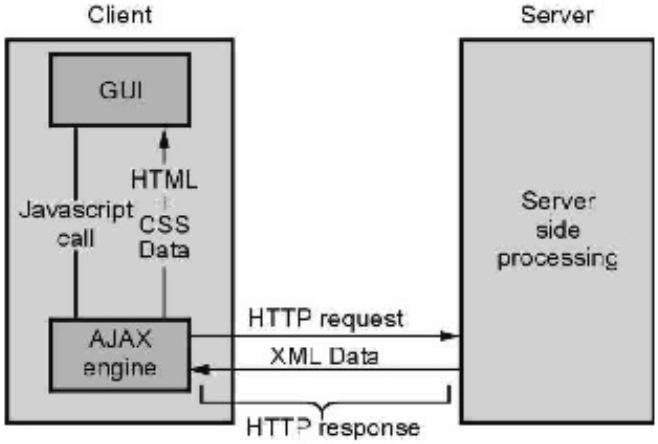


Fig. 1.17.3 AJAX web application architecture

Merits of AJAX

1. Response time is faster, hence increases **performance and speed.**
2. XMLHttpRequest object call as a asynchronous HTTP request to the Server for transferring data both side. It's used for making requests to the non-Ajax pages.
3. It is used in **form validation.**
4. It performs fetching of data from database and storing of data into database **without reloading page.**

Demerits of AJAX

1. It has browsing **compatibility issues.**
2. It is impossible to bookmark AJAX updated page contents.
3. Search engine **would not crawl AJAX generated content.** Hence, Search Engines like Google cannot index AJAX pages.

1.17.3 XMLHttpRequest Object

- XMLHttpRequest object is an important element in AJAX.
- XMLHttpRequest is an API which is used by JavaScript, VBScript and some other scripting languages.

- The methods of **XMLHttpRequest** are used in transferring data between a Web browser and a web server.
- It is because of **XMLHttpRequest** object, to update parts of web page without reloading the whole page.

Syntax

```
Variable = new XMLHttpRequest();
```

Various Methods of XMLHttpRequest object

Method	Purpose
new XMLHttpRequest()	Creates new XMLHttpRequest object.
Open(method, URL, async,user,pwd)	To make request this method is used. method: The request can be GET or POST URL: The location of file async: true(for asynchronous) and false(synchronous) user: user name(optional) pwd:password(optional)
send()	sends the request to the server.
send(string)	sends the request to the server.
setRequestHeader()	Adds the label- value pair to the header.

Various Properties of XMLHttpRequest object

Property	Description
readyState	Specifies the ready state of XMLHttpRequest 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
onreadystatechange	When readystate property changes the onreadystatechange event listener will be automatically invoked
responseText	Returns the response data as a string

status	Returns the status number. Following is the meaning of various numbers returned by the status property 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Returns the status text as "OK", or "Forbidden"

Let understand how AJAX works with the help of programming example

```
<html>
 <head>
  <script type="text/javascript">
   function MyFun()
   {
          if (window.XMLHttpRequest)
     {
       req=new XMLHttpRequest();
      }
      else
       {
         req=new ActiveXObject("Microsoft.req");
       }
      req.onreadystatechange=function()
       {
         if (req.readyState==4 && req.status==200)
          {
               document.getElementById("myID").innerHTML=req.responseText;
          }
       }
       req.open("GET","newdata.txt",true);
       req.send();
    }
  </script>
</head>
<body>
<div id="myID">This text can be changed</div>

<button type="button" onclick="MyFun()">Change</button>
</body>
</html>
```

2 3 4 5 1

Script Explanation (Above numbered steps are explained here)

In above script, we have written some text which can be replaced by some another text on button click.

1. On button click a function **MyFun** is invoked. Thus client triggers the event.
2. **XMLHttpRequest** object is used to exchange data with a server. This object allows the user to change/update the parts of the web page without reloading it fully. The modern web browsers such as IE7+, FireFox,Chrome have built in **XMLHttpRequest** but old web browsers make use of **ActiveXObject**.
3. When a request to a server is sent, then **onreadystatechange** event is triggered.

The **readyState** property holds the status of the XMLHttpRequest. The **readystate=4** means request is finished and response is ready. The **status = 200** means "OK"

The DOM is modified and response is modified using the statement

```
document.getElementById("myID").innerHTML=req.responseText;
```

4. The request can be sent to the server by using two functions **open()** and **send()**.

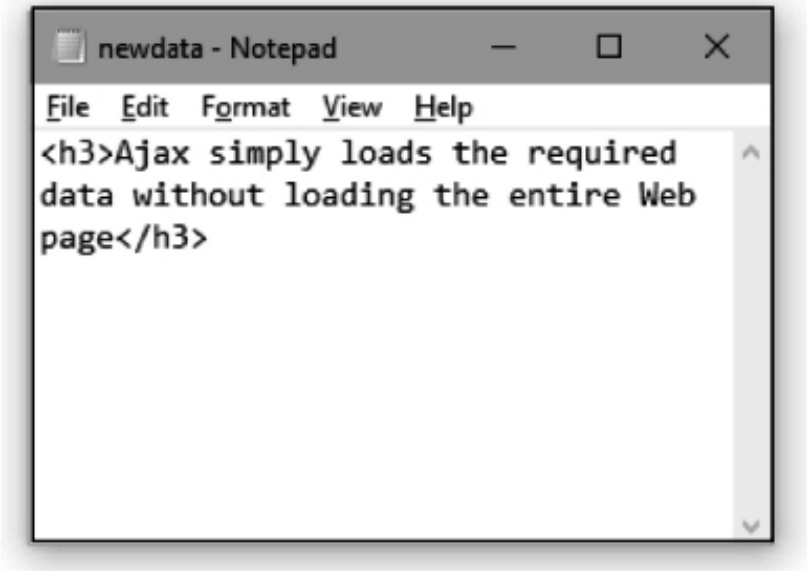
```
req.open("GET","newdata.txt",true);
```

- GET or POST method
- Location of file on server
- true: means asynchronous
  false : means asynchronous

Asynchronous communication allows fast processing of the data.

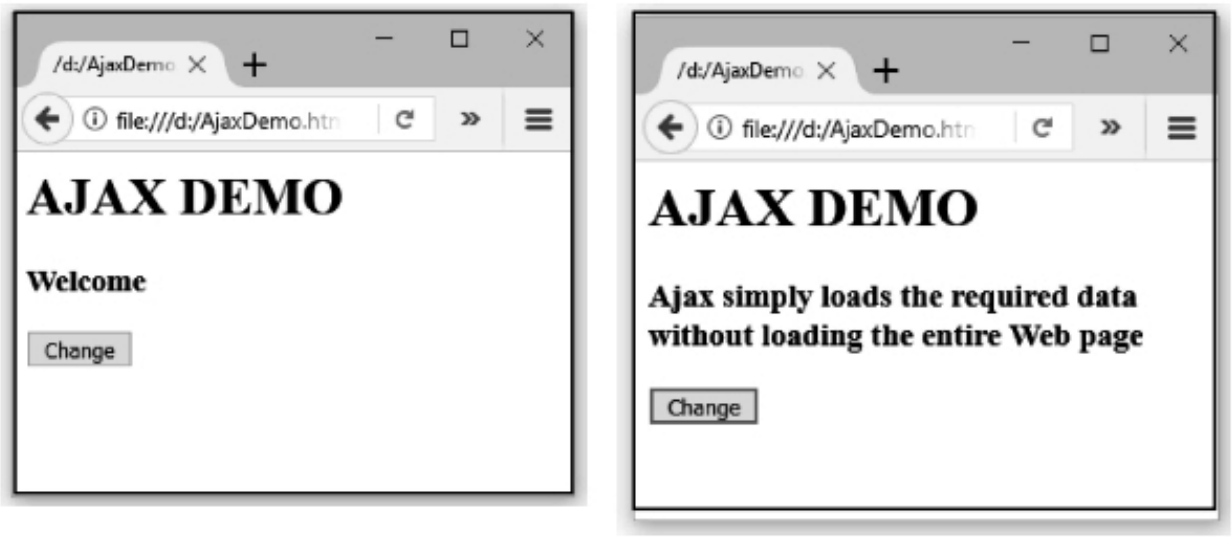
The **newdata.txt** file contains some updating text using which our web page can be updated.

newdata.txt



5. The **send()** method sends the request to the server.

Output



Example 1.17.1 *Write AJAX script to obtain the student information stored in XML document. The information should be displayed on clicking the button. It should be displayed in tabular form.*

Solution :

Step 1 : Create an XML file for storing the student information. The XML file is as follows

Student.xml

```
<Student>
    <student_data>
        <Name>AAA</Name>
        <Marks>45</Marks>
    </student_data>
    <student_data>
        <Name>BBB</Name>
        <Marks>55</Marks>
    </student_data>
    <student_data>
        <Name>CCC</Name>
        <Marks>67</Marks>
    </student_data>
    <student_data>
        <Name>DDD</Name>
        <Marks>84</Marks>
```

```
    </student_data>
</Student>
```

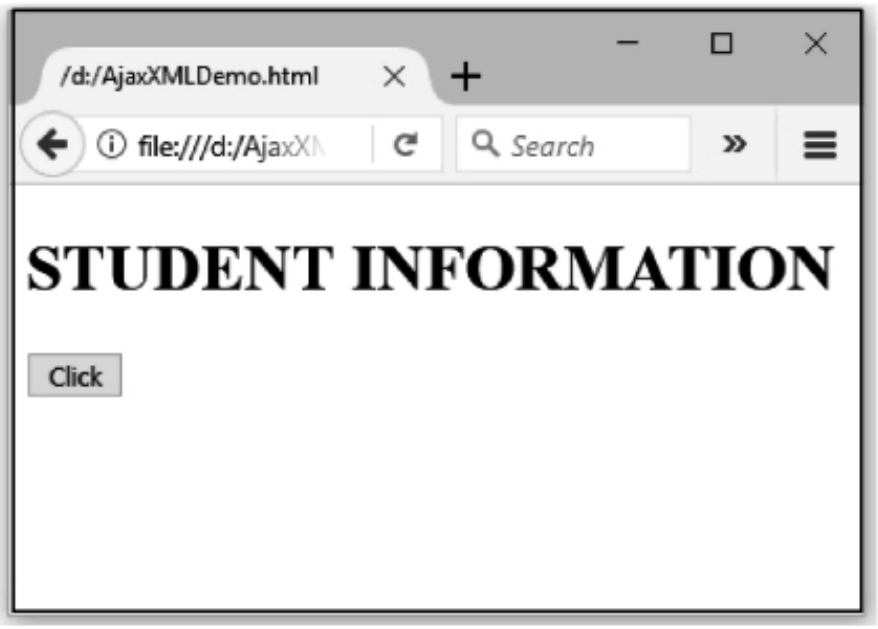
Step 2 : Create a AJAX script as follows –

AjaxXMLDemo.html

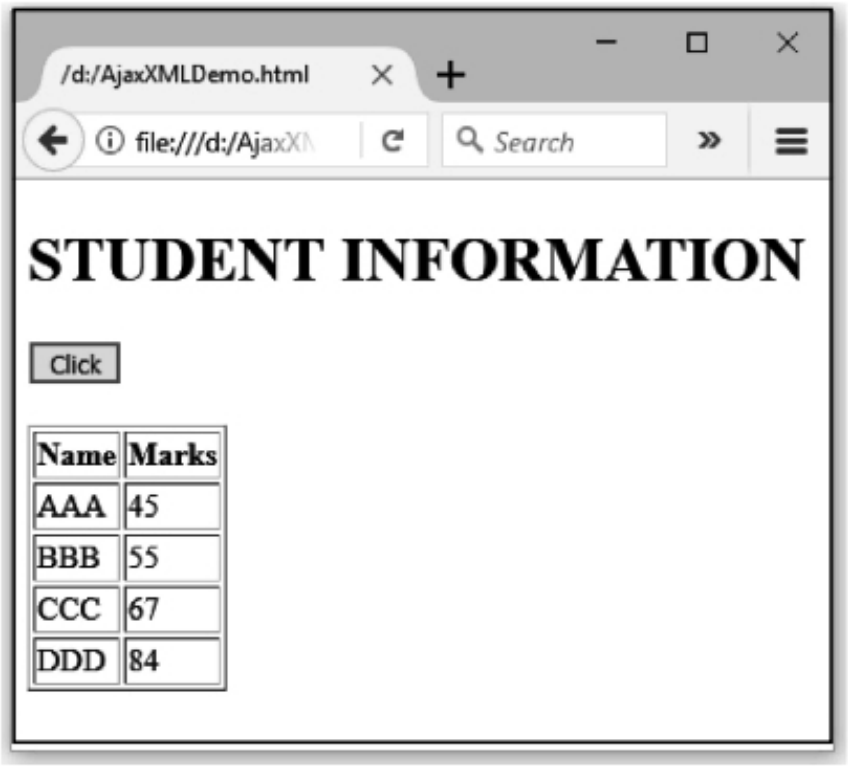
```
<!DOCTYPE html>
<html>
<body>
<h1>STUDENT INFORMATION</h1>
<button type="button" onclick="MyFun()">Click</button>
<br><br>
<table border="1" id="demo"></table>
<script>
function MyFun()
{
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function()
    {
        if (this.readyState == 4 && this.status == 200)
        {
          Load_XML_File(this);
        }
    };
    xhttp.open("GET", "Student.xml", true);
    xhttp.send();
}
function Load_XML_File(xml)
{
    var i;
    var xmlDoc = xml.responseXML;
    var table="<tr><th>Name</th><th>Marks</th></tr>";
    var x = xmlDoc.getElementsByTagName("student_data");
    for (i = 0; i <x.length; i++)
    {
        table += "<tr><td>" +
        x[i].getElementsByTagName("Name")[0].childNodes[0].nodeValue +
        "</td><td>" +
        x[i].getElementsByTagName("Marks")[0].childNodes[0].nodeValue +
        "</td></tr>";
    }
    document.getElementById("demo").innerHTML = table;
}
```

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

Step 3 : Open the web browser and the output will be as follows –



On clicking the button we will get



Review Questions

1. *What is AJAX? Explain the merits and demerits of using AJAX.*
2. *Explain XMLHttpRequest object along with its properties and methods.*
3. *Explain AJAX architecture in detail.*

❑❑❑

2 Introduction to Angular JS

Syllabus

Basics and Syntax of Angular JS, Features, Advantages, Application Structure, Basics of routes and navigation, MVC with Angular JS, Services.

Contents

2.1 Basics of Angular JS

- Angular JS is a powerful JavaScript framework.
- It is an open source front-end enabled web application framework.
- It is licensed under Apache license version 2.0.
- It extends HTML DOM with additional attributes. It is more responsive to user actions.

2.2 Features

There are core features Angular JS that are widely used by web developers. These are -

1. **Data binding :** It allows the automatic synchronization between model and view components.
2. **Scope :** There are some objects from model that can be correlated to controller and view.
3. **Controller :** These are basically JavaScript functions that are bound to particular scope.
4. **Directives :** The directives is designed to give HTML new functionality.
5. **Filters :** This feature allows to select subset from array of items.
6. **Routing :** This feature allows to switch between multiple views.
7. **Services :** There is a support for many built in services for Angular JS.
8. **Dependency injection :** It is a software design pattern that helps the developer to develop and understand the application easily.

Review Question

1. *Enlist the features of angular JS.*

2.3 Advantages and Disadvantages

Advantages

Following are the advantages of Angular JS

1. **Built by Google engineers :** The Angular JS is maintained by dedicated Google Engineers. That means - there is huge community for development and enhancement.
2. **MVC support -** It is based on **model view controller** (MVC) architecture which provides separation of model, view and controller components.

3. **Intuitive -** It is more intuitive as it makes use of HTML as a declarative language. Moreover, it is less brittle for reorganizing.
4. **Comprehensive :** AngularJS is a comprehensive solution for rapid front-end development. It does not need any other plugins or frameworks.
5. **Rich features :** There is a wide range of features supported by Angular JS such as Data binding, dependency injection, enterprise level testing and so on.
6. **Unit testing :** The Angular JS code is unit testable.
7. **Reusable code :** Angular JS support for using the reusable components.
8. **Less code :** It support for less code and more functionality.

Disadvantages

1. As Angular JS is based on JavaScript framework, it is not secure.
2. There are multiple ways to do the same thing with AngularJS. Sometimes, it can be hard for novices to say which way is better for a task.

Review Question

1. *Explain different advantages and disadvantages of Angular JS.*

2.4 Application Structure

- In Angular JS,the capacity of HTML is extended using the **ng-directives.**
- The directives are specified with **ng** prefix.
- The **three commonly used ng-directives** are
  1. **ng-app :** This directive defines an AngularJS application.
  2. **ng-model :** This directive binds the value of HTML controls to application data. These controls can be input, select, textarea and so on.
  3. **ng-bind :** This directive binds application data to HTML view.

Let us now learn and understand how to develop Angular JS application.

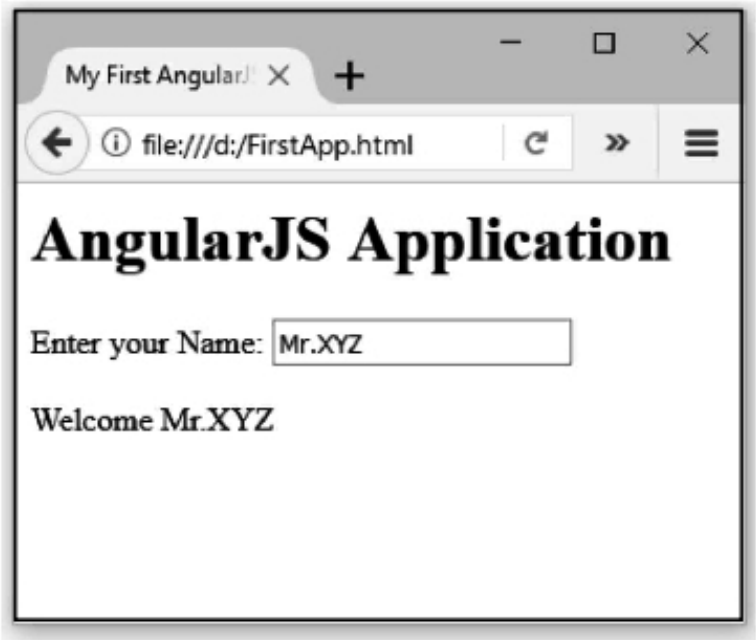
Step 1 : Create an Angular JS code as follows in a Notepad. Save it using the extension .htm or .html

AngularJS Document[FirstApp.html]

```
<html>
    <head>
    <title>My First AngularJS Application</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    </head>
```

```
    <body>
        <h1>AngularJS Application</h1>
        <div ng-app = "">
          <p>Enter your Name: <input type = "text" ng-model = "name"></p>
          <p>Welcome <span ng-bind = "name"></span></p>
        </div>
    </body>
</html>
```

Output



Script Explanation :

1. The AngularJS script is written within <html> tags. The AngualarJS is based on JavaScript framework. The framework for AngularJS is loaded using the <script> tag. It is as follows -

```
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
```

2. **Defining np-app directive :** The np-app directive is defined in above script as follows -

```
<div ng-app = "">
...
...
</div>
```

3. **Defining np-model directive :** The **textbox** control is modeled using np-model directive

```
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
```

4. **Defining np-bind :** We can bind the value entered in the textbox with **name** using np-bind directive.

```
<p>Welcome <span ng-bind = "name"></span></p>
```

Hence, we can see on the output screenshot, that whatever name entered by the user gets bound with the **Welcome** message.

The above output can be viewed by opening any popular web browser and by entering the URL of corresponding AngularJS script.

2.5 MVC with Angular JS

Concept of MVC

The MVC stands for Model, view and controller. It is a pattern for the architectural framework. It consists of three parts -

1. **Model :** This part of the architecture is responsible for managing the application data. This module responds to the request made from **view**. The **model** gives instructions to **controller** to update when the response is made.
2. **View :** This part takes care of the presentation of data. The data is presented in desired format with the help of **view**. This is a script based system using JSP, ASP, PHP and so on.
3. **Controller :** The controller receives input, validates it, and then performs business operations that modify the state of the data model. The **controller** basically responds to user request and performs interaction with **model**. Refer Fig. 2.5.1.

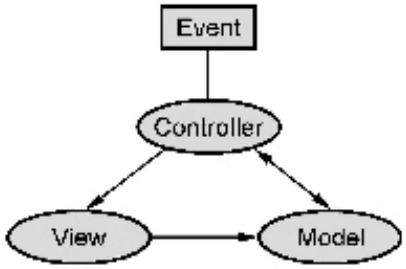


Fig. 2.5.1 MVC architecture

MVC in Angular JS

- Modules in AngularJS serve as containers to help you organize your application into logical units.
- Modules tell AngularJS how an application is configured.
- In angular JS the Module are represented using the directive **<ng-app>** and module name.

- The **scope object** binds the **view** with **controller**.
- In AngularJS it is possible to have multiple views for single page application.
- The **ng-view** and **ng-template** and **ng-route** directives support the concept of views.
- The use of controller in AngularJS is to control the flow of data. The **ng-controller** directive is used for this purpose. Refer Fig. 2.5.2.

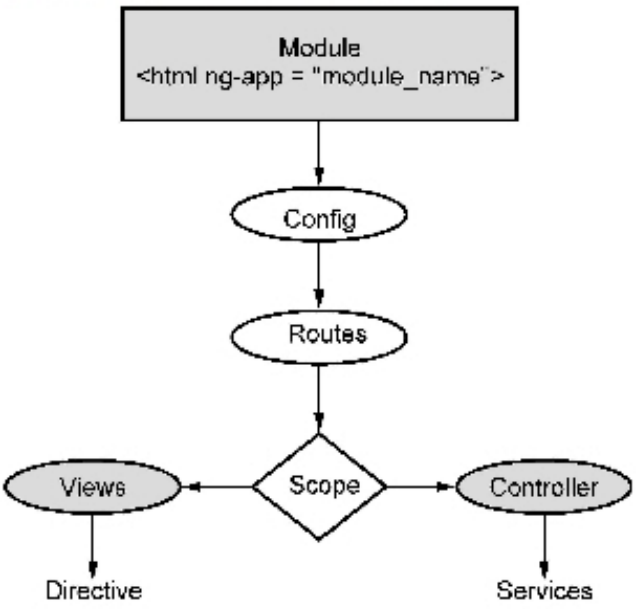


Fig. 2.5.2 MVC with angular JS

Review Question

1. *Explain the concept of MVC. How the MVC architecture is applicable in Angular JS?*

2.6 Basics of Routes and Navigation

- Routing and navigation is one of the core feature in AngularJS.
- The **#route** is used for deep linking of URLs to controllers and views.
- We have to download **angular-route.js** script that contains the ngRoute module from AngularJS official website to use the routing feature. Alternatively we can also use the CDN in our application to include this file. The CDN can be

```
https://ajax.googleapis.com/ajax/libs/angularjs/1.2.28/angular-route.min.js
```

- Then load the **ngRoute** module in the AngularJS application by adding it as a dependent module as shown below.

```
angular.module('appName', ['ngRoute']);
```

- Then using **ngView** directive we can display the contents at specific routes.

We will discuss the implementation of routes and navigation in the next chapter.

2.7 Expression

- In AngularJS the expressions are written within {{ }}. For example {{2+3}} is expression.
- AngularJS binds data to HTML using Expressions.
- AngularJS solves the expression and return the result of evaluation.
- The expressions in AngularJS are just similar to JavaScript expressions. The expression contains literals, variables and operators.

AngularJS Document[ExpressionDemo.html]

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="">
  <h4>Expression Demo</h4>
    <p> Addition of 10 and 20 = {{ 10+20 }}</p>
</div>

</body>
</html>
```

Output

Now from above code if we remove ng-app directive then the expression won't be evaluated. For instance -

instance –

```
<body>
<div>
 <h4>Expression Demo</h4>
   <p> Addition of 10 and 20 = {{ 10+20 }}</p>
</div>
</body>
</html>
```

Expression Demo

Addition of 10 and 20 = {{10+20}}

We can assign the values to variables and then evaluate an expression. This initialization of variable can be using **ng-init**. For example -

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="" ng-init="a=10;b=20">
 <h4>Expression Demo</h4>
   <p> Addition = {{ a+b }}</p>
   <p> Multiplication = {{ a*b }}</p>
</div>
</body>
</html>
```

Output

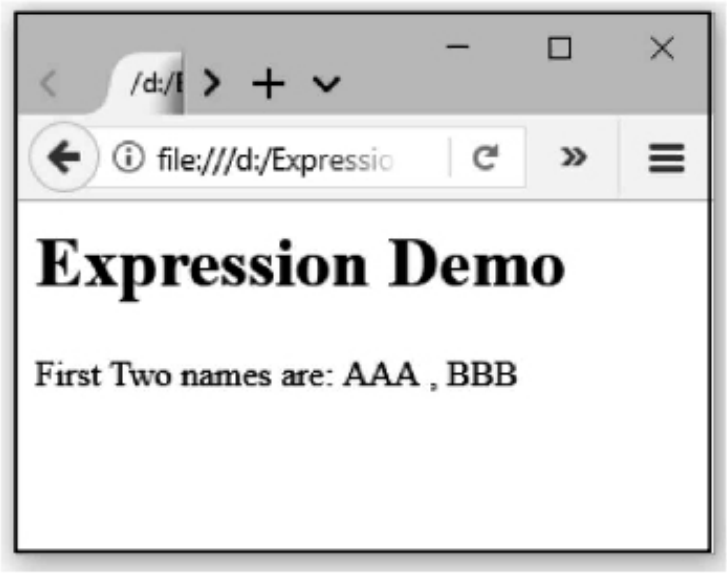
The output will be displayed as 30 and 200

Similarly we can assign strings to variables and can display them. For example

AngularJS Document[ExpressionDemo1.html]

```
<html>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
  <body>
   <h1>Expression Demo</h1>

   <div ng-app = "" ng-init = "names = {first:'AAA',second:'BBB',third:'CCC'}">
     <p>First Two names are:  {{names.first + " , " + names.second}}</p>
   </div>
 </body>
</html>
```

Output



2.8 Controller

- Controllers are basically the JavaScript objects. These objects contain the application logic.
- The controller are normally defined as a part of **angular module**. The angular module is created as follows -

```
var app =    angular.module("Test",[]);
```

Here the parameter **Test** refers to an HTML element in which the application will run.

- We can add the controller to this angular module. This controller can be referred using **ng-controller** directive.

```
<!DOCTYPE html>
<html>
<head>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<script type ="text/javascript">
var app = angular.module("Test",[]);
app.controller("student", function() {
```

The **Test** module is created

Inside the module, the controller is created. The name of the controller is **student**

```
        this.Name = "Chitra";
    });
</script>
</head>

<body ng-app = "Test" ng-controller="student as s">
    Welcome {{s.Name}}
</body>
</html>
```

Only one member of the controller. This member has a name as Name

The controller is referred by **ng-controller**. An instance named **s** is created for referring the members of the controller.

Displaying the value represented(modeled) by the controller on web browser.

Output

Program Explanation : The comments given at the each important coding statement provides the detailed insight of the controller. But you can map the analogy of MVC(Model View Controller) here as follows

Model and Controller : The controller can be considered as a collection of members. It is given as follows -

```
app.controller("student", function() {
    this.Name = "Chitra";
    });
```

Controller

Modelling the member

View: This part shows what the controller has modelled. It is as given below –

```
<body ng-app = "Test" ng-controller="student as s">
    Welcome {{s.Name}}
</body>
```

- We can use controller with properties, methods and external files.

Controller with Properties

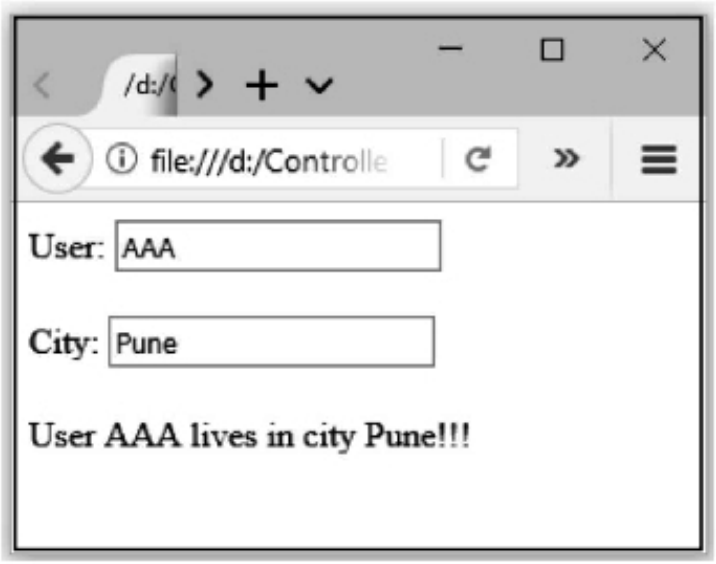
The controller is defined with properties. These properties are then assigned with some values. It is illustrated by following example

AngularJS Document[ControllerDemo.html]

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>

<div ng-app="MyApp" ng-controller="MyController">

User: <input type="text" ng-model="user"><br/><br/>
City: <input type="text" ng-model="city"><br/>
<br/>
User {{user + " "}} lives in city {{ city + "!!!"}}

</div>

<script>
var app = angular.module('MyApp', []);
app.controller('MyController', function($scope)
{
   $scope.user = "AAA";
   $scope.city = "Pune";
});
</script>
  </body>
</html>
```

Properties

Output



Script Explanation :

In above script,

1. we have used two directives **div-app** and **div-controller** which are defined with some suitable attribute name. In above case **div-app** attribute is "MyApp" and **div-controller** attribute is "MyController".
2. The **MyController** function is basically a JavaScript function and AngularJS will invoke the controller using **$scope** object.
3. The controller creates two properties namely - 'user' and 'city'. Using $scope object these properties are assigned with values 'AAA' and 'Pune'
4. The **ng-model** binds these the input fields to the controller properties.

Controller with Method

In above script the controller have two properties. But we can define a method for the controller. Following example illustrates how to use controller method.

AngularJS Document[ControllerMethodDemo.html]

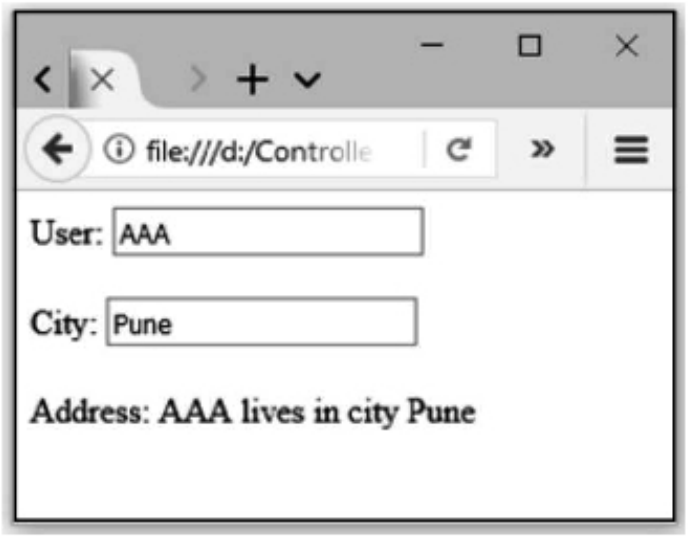
```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="MyApp" ng-controller="MyController">
 User: <input type="text" ng-model="user"><br/><br/>
 City: <input type="text" ng-model="city"><br/>
 <br/>
```

```
 Address: {{address_function()}}
</div>
<script>

var app = angular.module('MyApp', []);
app.controller('MyController', function($scope) {
   $scope.user = "AAA";
   $scope.city = "Pune";
   $scope.address_function=function() {
     return $scope.user +" lives in city "+$scope.city;
  };
});
</script>
</body>
</html>
```

Properties

←——Method definition

Output



Script Explanation :

In above script

1. The two properties **user** and **city** are defined. Along with them, a method named **Address_function()** is invoked.
2. This function returns the string containing values to the properties.
3. This function is then bounded to HTML

Controller with external File

We can write and save the controller in some external file. This file can be then invoked in the HTML document. Here is the illustration.

Step 1 : Create a file in Notepad. Name it with extension .js.

External File[MyController.js]

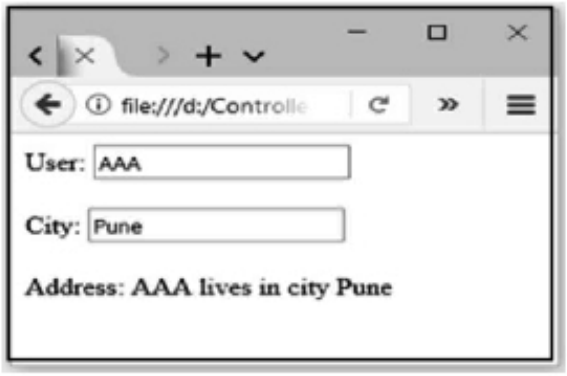
```
angular.module('MyApp', []).controller('MyController', function($scope) {
   $scope.user = "AAA";
   $scope.city = "Pune";
   $scope.address_function=function() {
    return $scope.user+" "+$scope.city;
  }
});
```

Step 2 : Create a HTML document invoking the external js file created in above step.

HTML Document[ControllerFileDemo.html]

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>

<div ng-app="MyApp" ng-controller="MyController">

User: <input type="text" ng-model="user"><br/><br/>
City: <input type="text" ng-model="city"><br/>
<br/>
Address:{{user + " lives in " + city}}
  </div>
<script    src="MyController.js"> </script>
</body>
</html>
```

Output



Review Question

1. *Explain the controller directive in Angular JS with suitable example. Also explain the implementation of MVC analogy by means of controller directive.*

2.9 Scope

The **Scope** is an object that binds the HTML with JavaScript. In other words scope object binds view with controller. The view part is normally written in HTML and controller part is written as JavaScript.

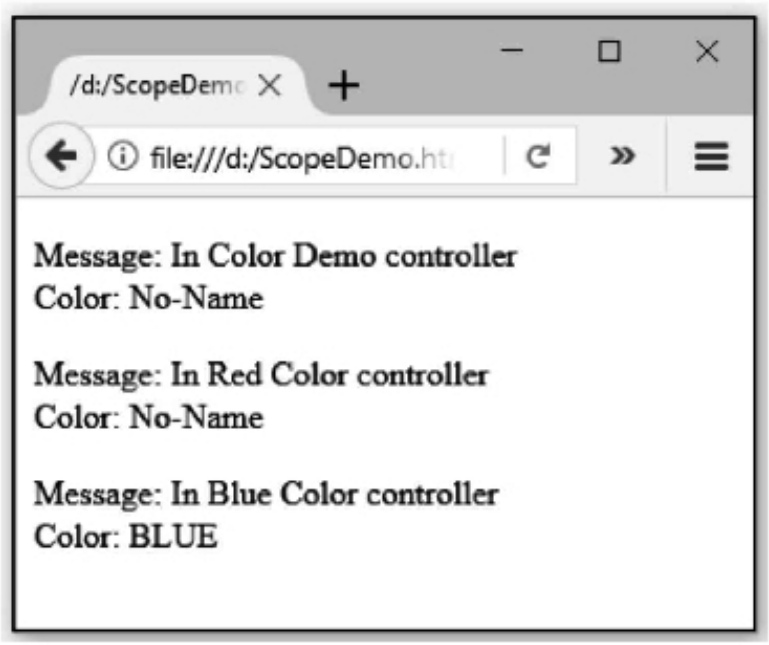
Let us understand the use of scope object with the help of illustrative example

AngularJS Document[ScopeDemo.html]

```
<html>
<body>
    <div ng-app = "MyApp" ng-controller = "MyController">
     <p>Message: {{message}} <br/>Color: {{colorname}} </p>

      <div ng-controller = "RedController">
      <p>Message: {{message}} <br/>Color: {{colorname}} </p>
      </div>

      <div ng-controller = "BlueController">
     <p>Message: {{message}} <br/> Color: {{colorname}} </p>
      </div>

    </div>

    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

    <script>
      var App = angular.module("MyApp", []);

      App.controller("MyController", function($scope) {
        $scope.message = "In Color Demo controller";
        $scope.colorname = "No-Name";
      });

      App.controller("RedController", function($scope) {
        $scope.message = "In Red Color controller";
      });

      App.controller("BlueController", function($scope) {
        $scope.message = "In Blue Color controller";
        $scope.colorname = "BLUE";
      });

    </script>
```

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

Output



Script Explanation :

In above code

1. we have created three controllers namely **MyController, RedController** and **BlueController.**
2. There are two properties namely **message** and **colorname.**
3. Note that **RedController** does not have **colorname** property, hence it inherits this value from **MyController.**
4. The **scope** object is used to define that values for these properties. Hence in HTML(view) part we can simply invoke the name of the property as **message** and **colorname** inside {{ }} and get the values assigned in respective controller.
5. Thus job of **scope** object is to bind the controller with view part of AngularJS document.

2.10 Services

The services are functions or objects that are intended to carry out specific task.

In AngularJS there are 30 built-in services used to carry out variety of tasks.

Built-in Services

Let us understand the use of services with the help of examples -

1. The $http Service

This is a commonly used service in AngularJS. By using this service, the request can be made to the server and the application handles the response.

Step 1 : Create a simple web document as follows -

msg.html

Well Done!!!

Step 2 : Now create a web document that will make use of **$http service** to get the message stored in above html file and to display the contents.

AngularJS Document[ServiceDemo1.html]

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>

<div ng-app="MyApp" ng-controller="MyController">
<p>Message obtained from a file is...</p>
<h1>{{message}}</h1>
</div>

<script>
var App = angular.module('MyApp', []);
App.controller('MyController', function($scope, $http) {
 $http.get("msg.html").then(function (response) {
    $scope.message = response.data;
 });
});
</script>

</body>
</html>
```

Output

Review Question

1. *Explain any three built in services used in Angular JS.*

❑❑❑

3 Angular JS in Details

Syllabus

Modules, Directives, Routes, Angular JS Forms and Validations, Data binding, Creating single page website using Angular JS.

Contents

3.1 Directives

- In Angular JS,the capacity of HTML is extended using the **ng-directives.**
- The directives are specified with **ng** prefix.
- The three commonly used ng-directives are,

1. **ng-app :** This directive defines an AngularJS application.
2. **ng-model :** This directive binds the value of HTML controls to application data. These controls can be input, select, textarea and so on.
3. **ng-bind :** This directive binds application data to HTML view.

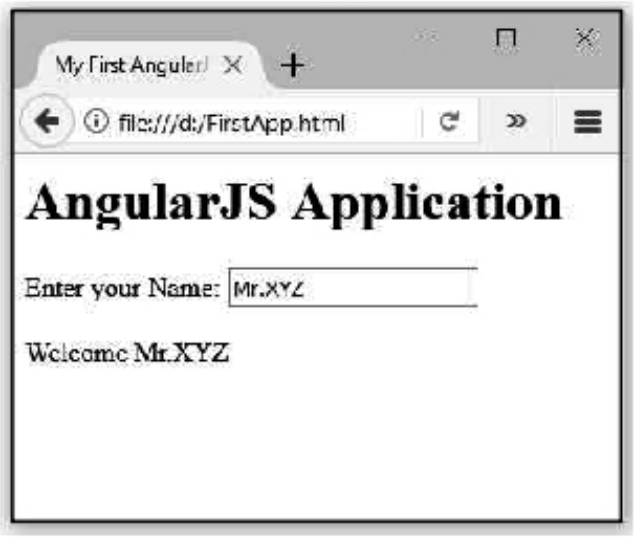
Let us now learn and understand how to develop Angular JS application.

Step 1 : Create an Angular JS code as follows in a Notepad. Save it using the extension .htm or .html

AngularJS Document[FirstApp.html]

```
<html>
    <head>
    <title>My First AngularJS Application</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    </head>
    <body>
       <h1>AngularJS Application</h1>
       <div ng-app = "">
         <p>Enter your Name: <input type = "text" ng-model = "name"></p>
         <p>Welcome <span ng-bind = "name"></span></p>
       </div>
    </body>
</html>
```

Output



Script Explanation :

1. The AngularJS script is written within <html> tags. The AngualarJS is based on JavaScript framework. The framework for AngularJS is loaded using the <script> tag. It is as follows -

```
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
```

2. **Defining np-app directive :** The np-app directive is defined in above script as follows -

```
<div ng-app = "">
...
...
</div>
```

3. **Defining np-model directive :** The **textbox** control is modeled using np-model directive

```
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
```

4. **Defining np-bind :** We can bind the value entered in the textbox with **name** using np-bind directive.

```
<p>Welcome <span ng-bind = "name"></span></p>
```

Hence, we can see on the output screenshot, that whatever name entered by the user gets bound with the **Welcome** message.

The above output can be viewed by opening any popular web browser and by entering the URL of corresponding AngularJS script.

Various Directives supported by AngularJS are -

1. ng-app **2. ng-init** **3. ng-model** **4. ng-repeat**

Let us discuss these directives with suitable scripting examples

1. ng-app Directive :

This directive starts the AngularJS application. It basically defines the **root element**. It initializes the application automatically when the web page containing the AngularJS code is loaded. For example,

```
<div ng-app="">
...
</div>
```

2. ng-init Directive :

The ng-init directive is used for initialization of AngularJS data.

For example

DirectiveDemo1.html

```
<html>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
        <div ng-app="" ng-init="msg='I love my country!!!'">
                <h3>{{msg}}</h3>
        </div>
</body>
</html>
```

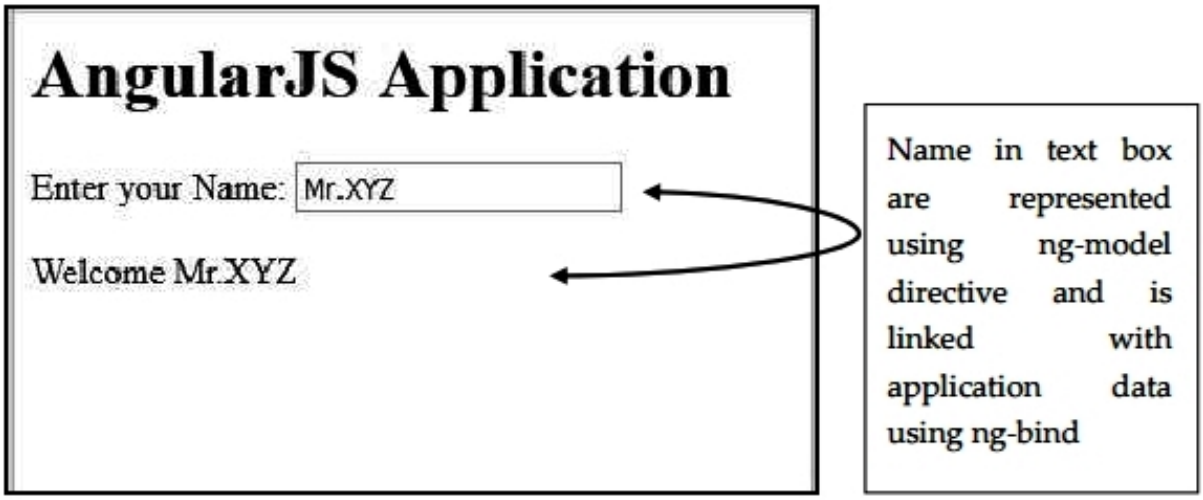
Output

3. **ng-model :** The ng-model directive binds the value of HTML control to application data. These controls can be input, select, textarea and so on. For example,

AngularJS Document[FirstApp.html]

```
<html>
    <head>
    <title>My First AngularJS Application</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    </head>
    <body>
        <h1>AngularJS Application</h1>
        <div ng-app = "">
          <p>Enter your Name: <input type = "text" ng-model = "name"></p>
          <p>Welcome <span ng-bind = "name"></span></p>
        </div>
    </body>
</html>
```

Output

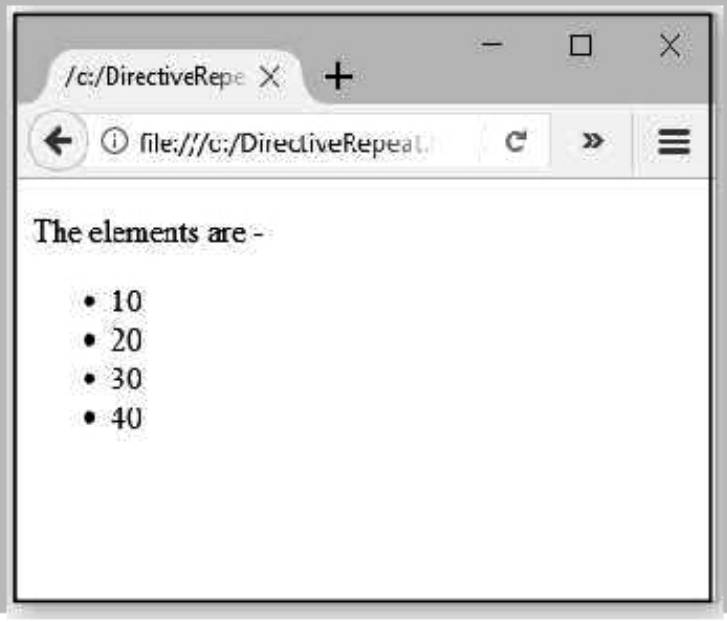


4. ng-repeat : The ng-repeat element directive repeats the HTML element.

AngularJS Document [DirectiveRepeat.html]

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="" ng-init="nums=[10,20,30,40,50]">
 <p>The elements are -</p>
 <ul>
  <li ng-repeat="i in nums">
   {{ i }}
  </li>
 </ul>
</div>
</body>
</html>
```

Output



Review Question

1. *Explain different directives used in Angular JS.*

3.2 Modules

- We can write the script in AngularJs using different modules. Basically the modules help in separation of logic such as controller, service,application etc.
- The modules can be defined in separate **.js** files.
- Most commonly there are two modules - **Application module** and **controller module.**
- Let us understand how to use modules in AngularJs with the help of example
- First Consider the following code without module -

AngularJS Document

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>

<div ng-app="MyApp" ng-controller="MyController">
 User: <input type="text" ng-model="user"><br/><br/>
 City: <input type="text" ng-model="city"><br/>
 <br/>
 Address: {{address_function()}}
</div>
 <script>
var app = angular.module('MyApp', []);
app.controller('MyController', function($scope) {
   $scope.user = "AAA";
   $scope.city = "Pune";
   $scope.address_function=function() {
     return $scope.user +" lives in city "+$scope.city;
  };
});
</script>
</body>
</html>
```

Now we will create three modules from above code -

1. Application Module 2. Controller Module 3. Driver Module or Use Module

Step 1 : Creating Application Module

For application module we will simply declare the application module using **angular.module** function. We've passed an empty array to it. This array generally contains dependent modules.

AngularJS Document[MyAppDemo.js]

```
var App=angular.module('MyApp', []);
```

Step 2 : Creating Controller Module

The controller **MyController** is declared using **App.controller** function.

AngularJS Document[MyControllerDemo.js]

```
App.controller('MyController', function($scope) {
  $scope.user = "AAA";
  $scope.city= "Pune"
  $scope.address_function=function() {
   return $scope.user+" "+$scope.city;
  }

});
```

Step 3 : Creating Driver or use Module

This module will invoke the above modules and will display the result. The two scripts defined in above steps are included in this document using the **<script src>** tag.

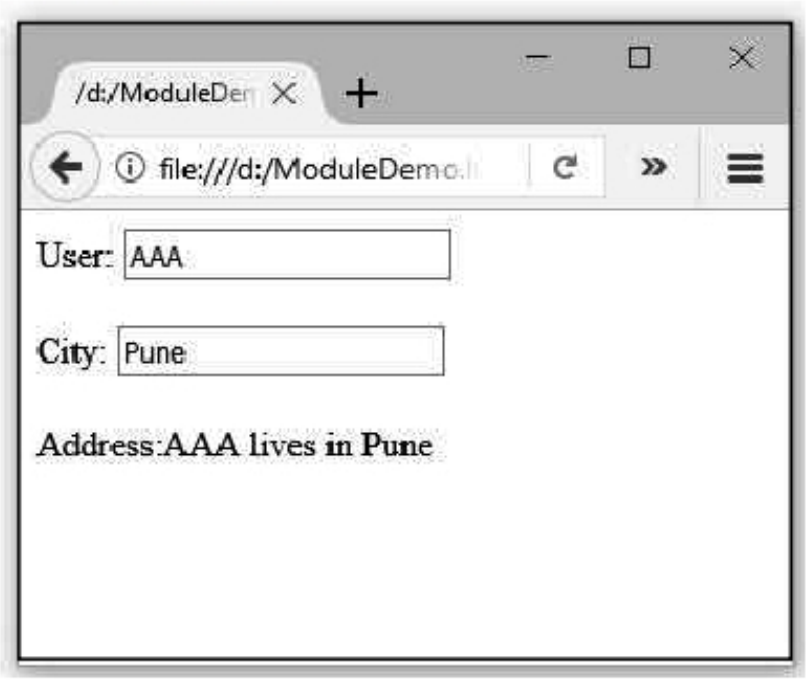
Then using **ng-app** and **ng-controller** directives the **MyAppDemo.js** and **MyController.js** modules are mapped.

AngularJS Document[ModuleDemo.html]

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<script src = "MyAppDemo.js"></script>
<script src = "MyControllerDemo.js"></script>
<body>
<div ng-app="MyApp" ng-controller="MyController">
User: <input type="text" ng-model="user"><br/><br/>
City: <input type="text" ng-model="city"><br/>
<br/>
Address:{{user + " lives in " + city}}
</div>
</body>
</html>
```

Invoking external modules

Step 4 : The output can be viewed as



Review Question

1. *Explain the use of module in Angular Js.*

3.3 Routes

- If you want to navigate to different web pages without reloading the entire application we can use the **ngRoute** module.
- It is used in **angular.module.**

Let us first see the following example, that uses the **ngRoute**. After this example, we will have detail discussion on routes and navigations in Angular JS.

Step 1 : Create main page, that contains the hyperlinks for navigation. It is as follows -

routerDemo.htm

```
<!DOCTYPE html>
<html>
<head>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.js"></script>
<script type ="text/javascript" src="router1.js"></script>

</head>
<body ng-app ="app">
```

```
    <h3>
    <a href='#/red-msg'> What is the color of an Apple? </a>
    <br><br>
    <a href='#/blue-msg'> How is sky? </a>
    <br>
    </h3>
    <div ng-view></div>
</body>
</html>
```

Step 2 : Now create a Javascript page(.js file) that contains configuration for router and controllers.

router1.js

```
var app = angular.module('app', ['ngRoute']);
app.config(['$routeProvider',function($routeProvider){
    $routeProvider
    .when('/red-msg', {
        templateUrl:'red.htm',
        controller: 'message1'
    })
    .when('/blue-msg', {
        templateUrl:'blue.htm',
        controller: 'message2'
    })
    .when('/',{
        template:'<h2>Welcome Page</h2>'
    })
}]);

app.controller('message1',function($scope){
    $scope.colorName = 'Red'
});
app.controller('message2',function($scope){
    $scope.newName = 'Blue'
});
```

Step 3 : Now above mentioned html files are as follows

red.htm

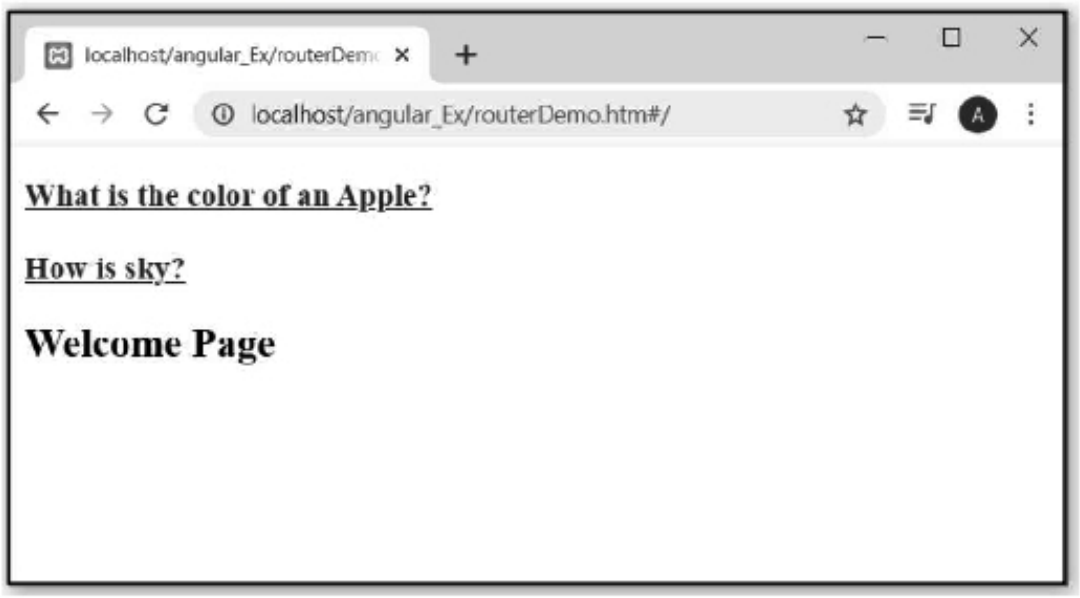
```
<h2>
Apple is {{colorName}}
</h2>
```

blue.htm

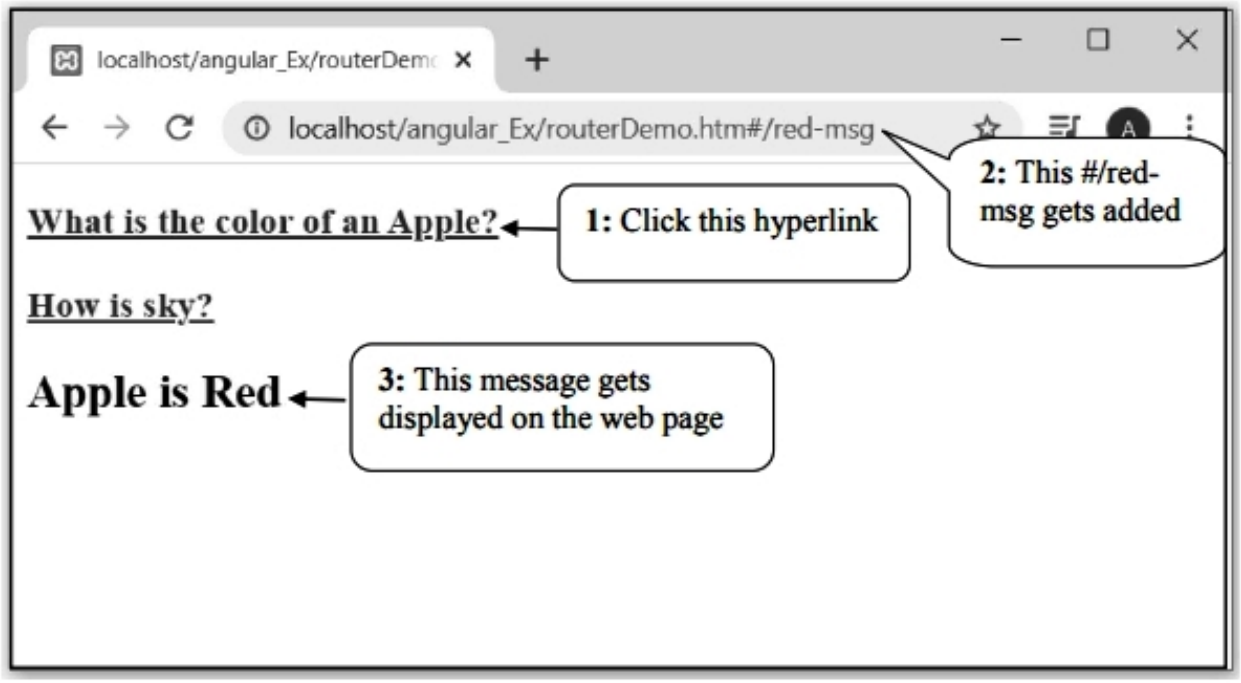
```
<h2>
```

```
Sky is {{newName}}
</h2>
```

Step 4 : The output is as follows – Firstly the default web page is displayed with Welcome message.



Then click on the first hyperlink.



Similarly on clicking another hyperlink, the page gets navigated to another html file(blue.htm) and the contents of that file are displayed on the web page.

Note that, the above output clearly shows that we need not have to reload the entire application for navigating to another web page.

Routing is mainly used in Single Page Applications(SPA) in which new contents are actually injected in the existing web page.

Routing is available both at the server as well as the client side. Note that above example is executed at the server level(XAMPP).

Following are the steps used for using ngRoute are -

Step 1[Include Section] : Include **angular-route.js** after including angular.min.js file. That means, it should be as follows –

```
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.js"></script>
```

Step 2[module dependency]: Use module dependency using **ngRoute** as follows -

```
var app = angular.module('app',[ngRoute']);
```

Step 3[Configuration] : Now the routes need to be configured. The configuration means – which HTML files are loaded on particular routing, what are the view associated with them and what are the controllers that must be associated with these views. For this configuration we have to use **$routeProvider**. With the **$routeProvider** you can define what page to display when a user clicks a link.

The sample code for such configuration is -

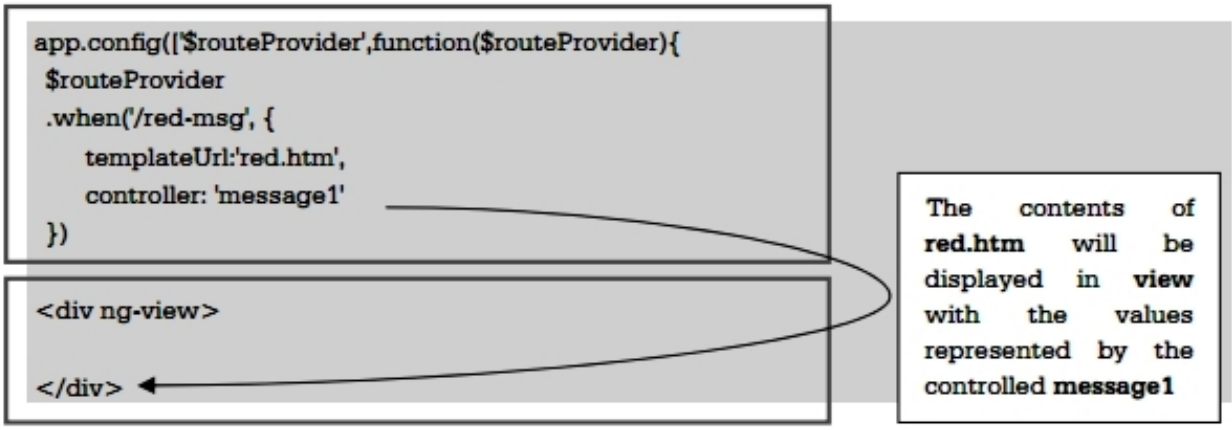
```
app.config(['$routeProvider',function($routeProvider){
```

```
$routeProvider
.when('/red-msg', {
    templateUrl:'red.htm',
    controller: 'message1'
})
```

Step 4[viewing the contents] : In above given example, we are fetching the contents from **red.htm** and associating those contents with the controller. The controller will model those contents. These contents will then be viewed using **ng-view**. The contents can be injected into the view. For that purpose you need to add up following code in your application

```
<div ng-view>
</div>
```

Note that the contents displayed in this view are dynamic. That means, when the **templateUrl** of configuration changes, then accordingly the contents will be displayed or rendered by the view. That means –



Fundamental Definitions of terms used in routing

1) **ngRoute :** The ngRoute module provides routing and deep-linking services and directives for AngularJS applications. We Use module dependency using ngRoute. It is as follows

```
var app = angular.module('app',[ngRoute']);
```

Here, **app** is a name of the module for which we have to use routing.

2) **routeProvider :** The ngProvider is used for configuring routes.

3) **When :** It just states that when the relevant path is chosen, use the route to display the given view to the user. The syntax is as follows -

```
when(path, route)
```

4) **tempalteUrl :** The templateUrl specifies the path of the filename which is to be displayed using routing.
5) **template :** Using template we can specify the text message that is to be displayed on the web page on routing.
6) **Controller :** The name of the controller is specified using **controller** property. The use of controller in Angular JS is to control the flow of data.

3.4 Angular JS Forms and Validations

- Form is a collection of various controls.
- AngularJS support two types of controls - **Data binding** and **validation controls.**
- Various controls supported by AngularJS forms are -
  - Input
  - Select
  - Button
  - textarea.

1. Input Control

User can type some text in the text box and the same text can be displayed on the AngularJS document. For this binding we use inside the <form> tag.

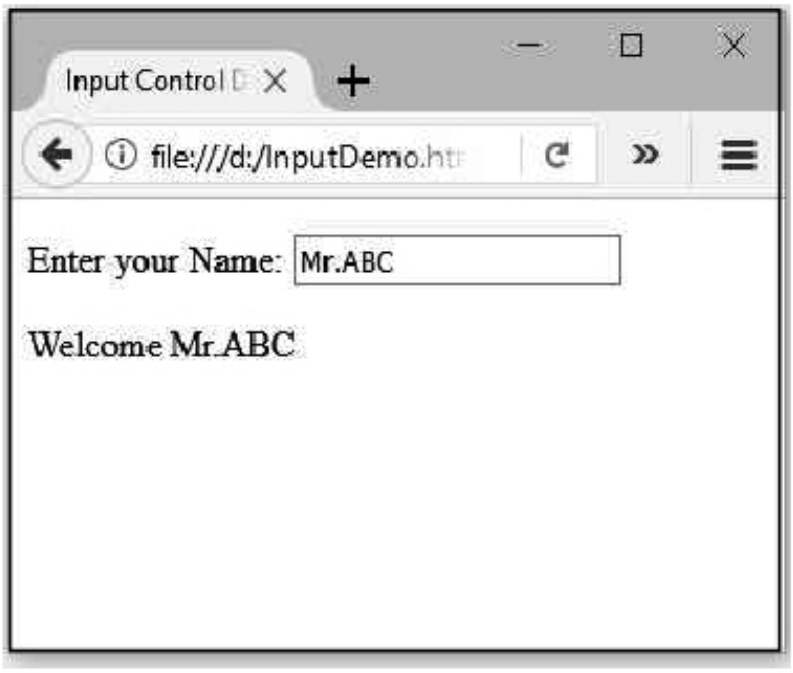
```
<input type="text" ng-model ="name">
```

Following example illustrates the use of input control. When user types the text inside the text box, it will be displayed along with the Welcome message.

AngularJS Document[InputDemo.html]

```
<html>
    <head>
    <title>Input Control Demo</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    </head>
    <body>
        <div ng-app = "">
        <form>
          <p>Enter your Name: <input type = "text" ng-model = "name"></p>
        </form>
          <p>Welcome {{name}}</p>
        </div>
    </body>
</html>
```

Output



2. Checkbox

We can have checkbox components on the Web document using AngularJS. For having checkbox , the input type must be "checkbox"

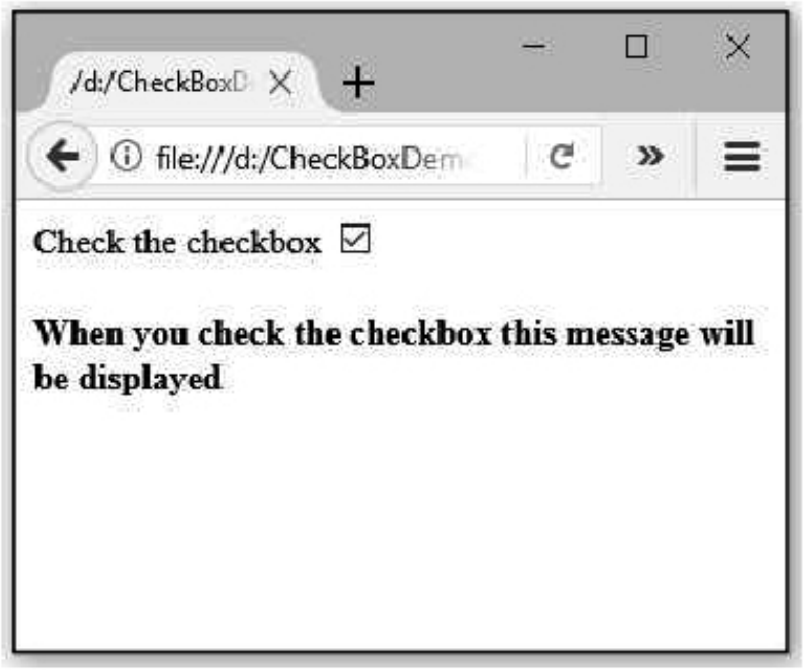
```
<input type="checkbox" ng-model="chked">
```

Here is an illustrative example for checkbox

AngularJS Document[CheckBoxDemo.html]

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="">
 <form>
  Check the checkbox
  <input type="checkbox" ng-model="chked">
 </form>
 <h4 ng-show="chked">When you check the checkbox this message will be displayed</h4>
</div>
</body>
</html>
```

Output



3. Radio Button

The radio button is basically a group of buttons which allows you to choose one value at a time.

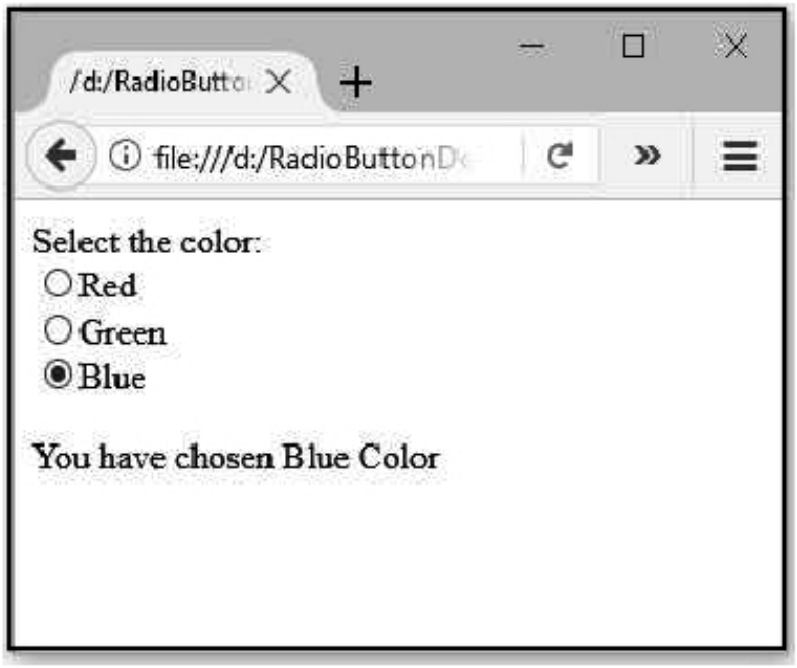
Let us see an illustrative script that uses Radio button control

AngularJS Document[RadioButtonDemo.html]

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body ng-app="">
    <form>
        Select the color:<br/>
        <input type="radio" ng-model="color" value="r">Red <br/>
        <input type="radio" ng-model="color" value="g">Green <br/>
        <input type="radio" ng-model="color" value="b">Blue  <br/>
    </form>
    <div ng-switch="color">
    <div ng-switch-when="r">
        <p> You have chosen Red Color </p>
    </div>
    <div ng-switch-when="g">
        <p> You have chosen Green Color </p>
    </div>
    <div ng-switch-when="b">
```

```
        <p> You have chosen Blue Color </p>
    </div>
    </div>
</body>
</html>
```

Output



Script Explanation

1. We have used <form> </form> tag inside which the <input type="radio"> is taken for display of radio button.
2. The **ng-model** directive is used to bind radio buttons with HTML document, the ng-model is created for a group "color". the values are set as "r", "g" or "b"
3. In this script we have taken **ng-switch directive** to hide and show HTML sections depending on the value of the radio buttons. When the value of particular radio button is matched, then corresponding message is displayed.

4. The selectbox

The dropdown list can be created using <select> <option> tags. For example

AngularJS Document[SelectBoxDemo.html]

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body ng-app="">
        <form>
```

```
            Select the color:<br/>
            <select ng-model="color">
             <option value="r">Red <br/>
             <option value="g">Green <br/>
             <option value="b">Blue  <br/>
            </select>
        </form>
        <div ng-switch="color">
        <div ng-switch-when="r">
            <p> You have chosen Red Color </p>
        </div>
        <div ng-switch-when="g">
            <p> You have chosen Green Color </p>
        </div>
        <div ng-switch-when="b">
            <p> You have chosen Blue Color </p>
        </div>
        </div>
</body>
</html>
```

Output

5. The button

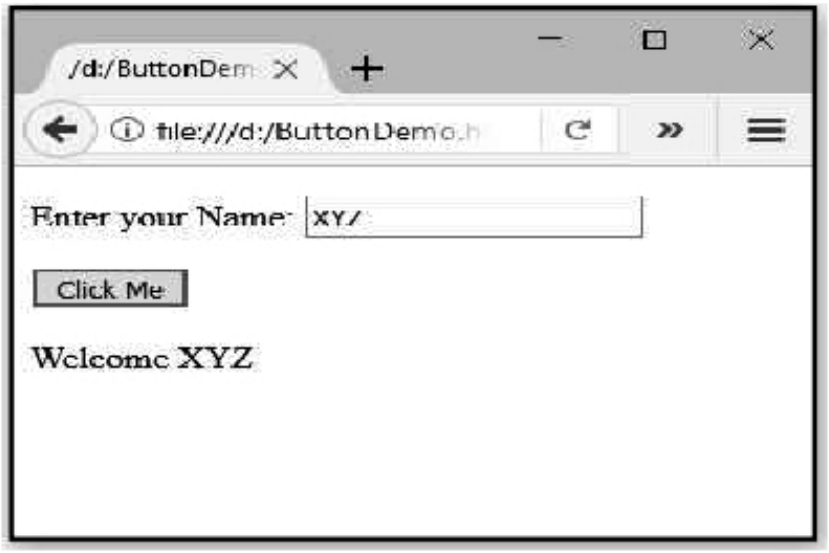
We can create a button control and invoke some functionality at its click event. For example

AngularJS Document[ButtonDemo.html]

```
<html>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
```

```
<body>
<div ng-app = "MyApp" ng-controller="MyController">
<form>
        <p>Enter your Name: <input type = "text" ng-model = "name"></p>
        <input type = "button" value = "Click Me" ng-click="display()"></p>
</form>
<p>Welcome {{username}}</p>
</div>
<script>
var app = angular.module('MyApp', []);
app.controller('MyController', function($scope) {
   $scope.name = "AAA";
   $scope.display=function() {
        $scope.username=$scope.name;
  };
});
</script>
        </body>
</html>
```

Output



In above script on **ng-click** event the function **display** is invoked. This function displays the name typed by the user in the input text box.

Review Question

1. *Explain various form elements with example in angular js.*

3.5 Data Binding

Concept of Data Binding : Data binding is a technique of establishing a connection between view and controller and exchange of synchronized data performed by angular JS in automatic manner.

There are **two** ways to perform data binding,

(1) One way data binding **(2) Two way data binding**

1) One way data binding : This is a kind of data binding in which data is sent from controller to view.

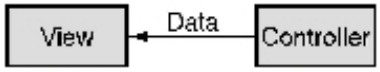


Fig. 3.5.1 One way data binding

Example Program : Following is a simple application created in Angular JS that demonstrates the one way data binding. In this application, we bind the data such as roll number and name of the student from controller to view.

Step 1 : Create a **Controller** in a JavaScript(js) file as follows -

oneway.js

```
var app = angular.module("app",[]);
app.controller('Student',['$scope',function($scope){
    $scope.rollNo = 101;
    $scope.studname = "Chitra";
}])
```

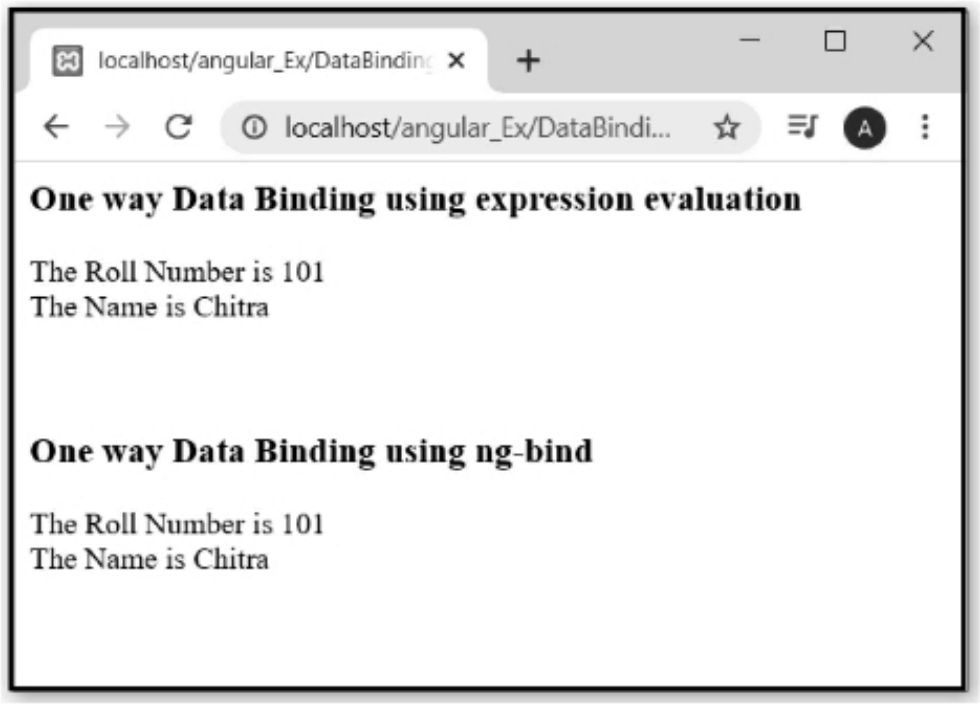
Step 2 : Create a **view** in HTML file as follows -

oneway.html

```
<html>
<head>
    <title></title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>
    <script type = "text/javascript" src = "oneway.js"></script>
</head>
<body ng-app = "app">
    <div ng-controller ="Student">
        <h3> One way Data Binding using expression evaluation </h3>
        <div>
                The Roll Number is {{rollNo}} <br>
```

```
                The Name is {{studname}} <br>
        </div>
        <br><br>
          <h3> One way Data Binding using ng-bind </h3>
        <div>
          The Roll Number is <span ng-bind = "rollNo"></span> <br>
          The Name is <span ng-bind = "studname"></span> <br>
        </div>

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

Step 3 : Open some suitable web browser and load the above html file.



Script Explanation : In above created application,

- Data flows from controller(present in **.js** file mentioned in step 1) to view or other words we can say data is propagated from scope to view.
- **View** data(present in **htm** file given in step 2) is not propagated back to the controller.
- For propagating the data from controller to view we use either expression evaluation in {{ }} or can use **ng-bind** directive.
- Note that in above given output the values of rollnumer and student name are displayed on view. That means these values are passed from controller to view.

2) Two Way Data Binding : It is a kind of data binding in which data is sent from controller to view and from view to controller in a synchronized manner.

Fig. 3.5.2 Two way data binding

Example Program : Following is a simple application created in Angular JS that demonstrates the two way data binding. In this application, we bind the data such as rollnumber and name of the student from controller to view and from view to controller.

Step 1 : Create a **Controller** in a JavaScript(js) file as follows -

twoway.js

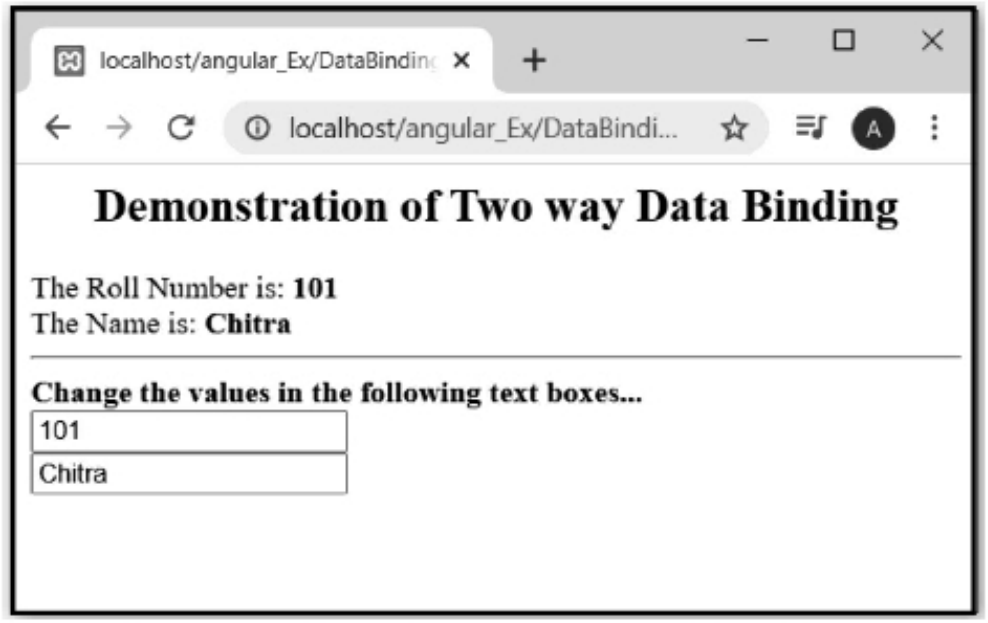
```
var app = angular.module("app",[]);
app.controller('Student',['$scope',function($scope){
    $scope.rollNo = 101;
    $scope.studname = "Chitra";
}])
```

Step 2 : Create a **view** in HTML file as follows -

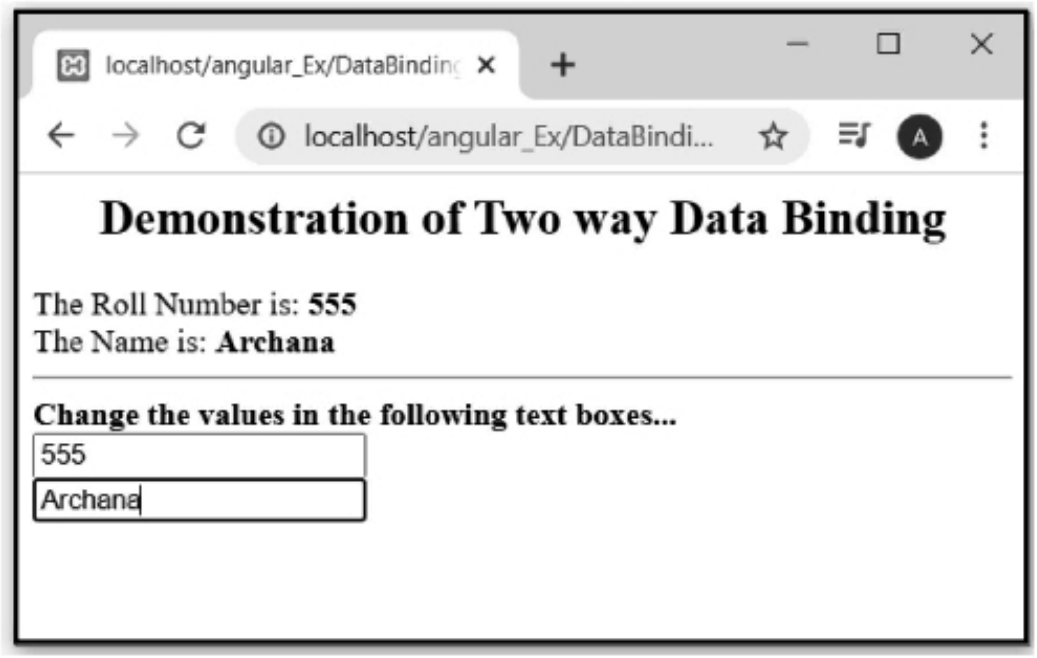
twoway.html

```
<html>
<head>
    <title></title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>
    <script type = "text/javascript" src = "twoway.js"></script>
</head>
<body ng-app = "app">
    <div ng-controller ="Student">
        <center><h2> Demonstration of Two way Data Binding </h2></center>
        <div>
          The Roll Number is: <strong><span ng-bind = "rollNo"></span></strong> <br>
          The Name is: <strong><span ng-bind = "studname"></span></strong> <br>
        </div>
        <hr>
        <strong> Change the values in the following text boxes... </strong>
        <br>
        <input type = text ng-model ="rollNo"/><br>
        <input type = text ng-model ="studname"/><br>
    </div>
</body>
</html>
```

Output



Now change the contents of the text boxes and see the effect. The values get changed instantly while changing the contents of the text box.



Script Explanation : In above created application,

- Data flows from **controller**(present in **.js** file mentioned in step 1) to **view** and from **view to controller.**
- **View** data(present in **htm** file given in step 2) is propagated back to the controller.

For writing the data at view level we use the text box.

- Using **ng-model ="rollNo"** we can associate roll number to text box similarly using **ng-model ="studname"** we can associate student name to the text box.
- For propagating the data from controller to view we use **ng-bind** directive.
- Note that in above given output the values of rollnumer and student name are displayed on view. That means these values are passed from controller to view. Similarly on view page if we change the roll number and student name in the text boxes then those modifications are reflected immediately.

Review Questions

1. *What is data binding ?*
2. *Explain one way data binding with suitable angular JS example.*
3. *Explain two way data binding with suitable angular JS example.*

3.6 Creating Single Page Website using Angular JS

- Single page application is a web application that fits on a single page. That means all the code present in .js, .html or .css file is retrieved with single page load.
- Single page applications or (SPAs) are web applications that load a single HTML page and dynamically update the page based on the user interaction with the web application.
- The navigation between various pages is performed without refreshing the whole page.
- The typical layout of any single page web site is as follows -

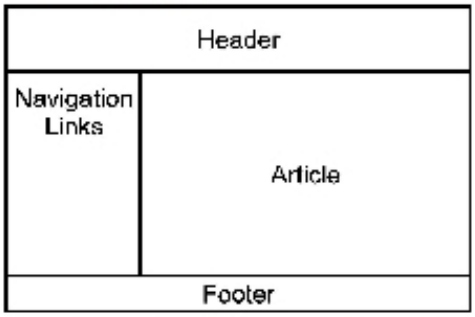


Fig. 3.6.1 Typical layout of SPA

- **Header and Footer :** The header element specifies the header for the document or section A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

- **Navigation Links :** It is used for declaring a navigational section of the HTML document. Websites typically have sections dedicated to navigational links that enable the user to navigate the site.
- **Article :** This part specifies independent, self-contained content. It displays the description part when the specific link is clicked.

Features of SPA

1) **No page refresh :** When you are using SPA, you don't need to refresh the whole page, just load the part of the page which needs to be changed.
2) **Better user experience :** It is fast and responsive.
3) **Ability to work offline :** Even if user loses internet connection, SPA can still work because all the pages are already loaded.
4) **Easy for deployment :** SPA are very easy to deploy in product applications.
5) **Less bandwidth :** SPAs consume less bandwidth since they only load web pages once. Besides that, they can also do well in areas with a slow internet connection.

Example

Step 1 : Create a javascript(.js) file as follows -

```
var app = angular.module('myApp', ['ngRoute']);

app.config(function($routeProvider) {
$routeProvider
.when('/', {
templateUrl : 'home.html',
controller : 'HomeController'
})
.when('/about', {
templateUrl : 'about.html',
controller : 'AboutController'
})
.when('/contact', {
templateUrl : 'contact.html',
controller : 'ContactController'
})
.otherwise({redirectTo: '/'});
});

app.controller('HomeController', function($scope) {
$scope.message = 'Welcome to a home page';
});
```

```
app.controller('AboutController', function($scope) {
$scope.message = 'We are enthusiastic Web Developers';
});

app.controller('ContactController', function($scope) {
$scope.message = 'Our contact information is present on this page';
});
```

Step 2 : Create a view in HTML file as follows

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-route.min.js"></script>
</head>
<body ng-app="myApp">
    <div class = "row">
        <div class = "col-sm-11 well">
        <center><h1>OUR WEB SITE</h1></center>
        </div>
    </div>
    <div class = "row">
        <div class ="col-sm-2 well" style = "background-color:pink;height:100%" >
          <ul>
            <li><a href="#/">Home</a></li>
            <li><a href="#/about">About</a></li>
            <li><a href="#/contact">Contact</a></li>
          </ul>
          <br><br>
        </div>
        <div class ="col-sm-9 well" style="background-color:aqua;height:100%" >
          <div ng-view></div>
          <br><br><br>
        </div>
    </div>
<script src="appDemo.js"></script>
</body>
</html>
```

Step 3 : The required html files(mentioned in the script of step 1) that display the contents on the web page are as follows -

home.html

```
<h3>{{message}}</h3>
```

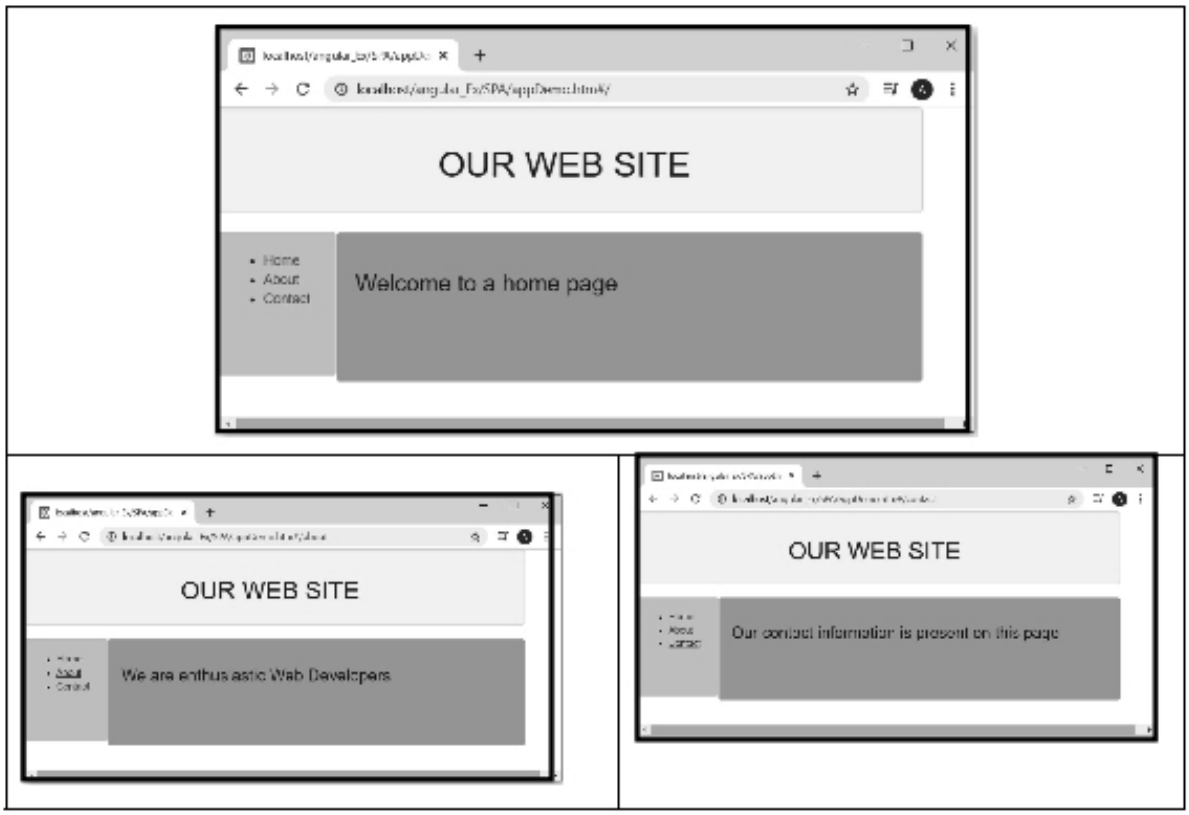
about.html

```
<h3>{{message}}</h3>
```

contact.html

```
<h3>{{message}}</h3>
```

Step 4 : The output can be displayed as follows -



4 Introduction to Node JS

Syllabus

Setup Node JS Environment, Package Manager, Features, Console Object, Concept of Callbacks.

Contents

4.1 Setup Node JS Environment

- NodeJS is an open source technology for server.
- Using Node.js we can run JavaScript on server.
- It runs on various platform such as Windows, Linux, Unix, and MacOS.

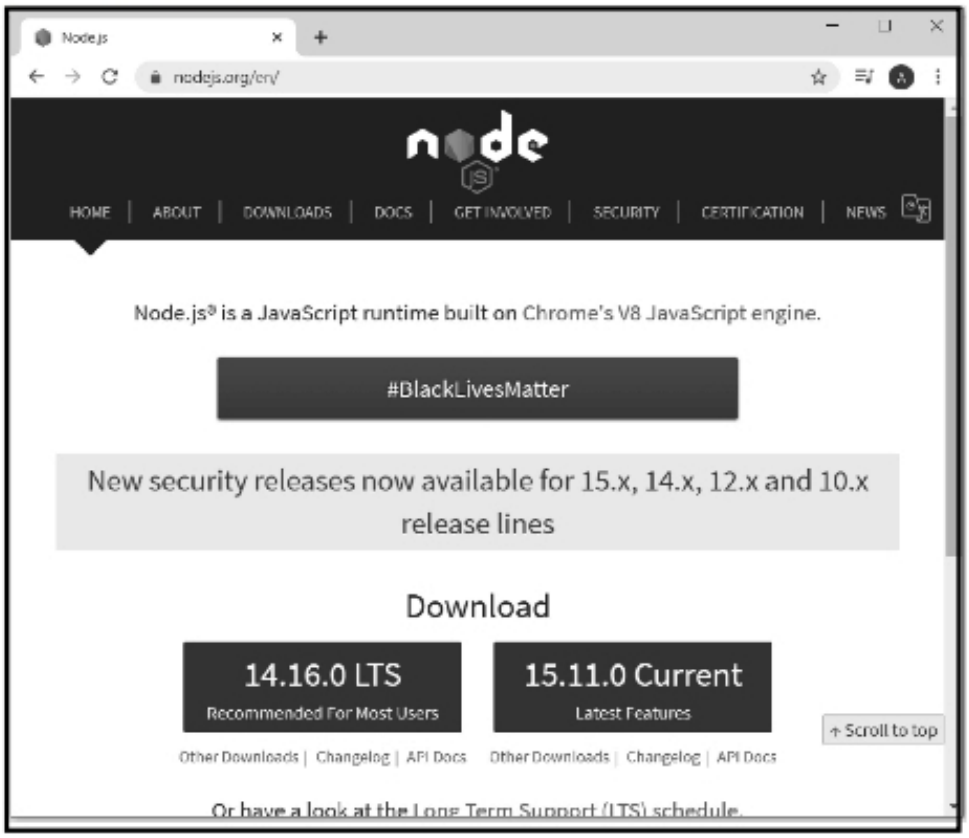
Uses of Node.js

It can perform various tasks such as -

1) It can create, open, read,delete, write and close files on the server.
2) It can collect form data.
3) It can also add, delete, modify data in databases.
4) It generate dynamic web pages.

How to write and Execute Node.js Document ?

- For executing the Node.js scripts we need to install it. We can get it downloaded from https://nodejs.org/en.
- The installation can be done on Linux or Windows OS. It has very simple installation procedure.



- Download node MSI for windows by clicking on 14.16.0 LTS or 15.11.0 Current button.
- After you download the MSI, double-click on it to start the installation as shown below.

- Click **Next** button to read and **accept** the License Agreement and then click Install. It will install Node.js quickly on your computer. Finally, click finish to complete the installation.
- **Verify Installation**

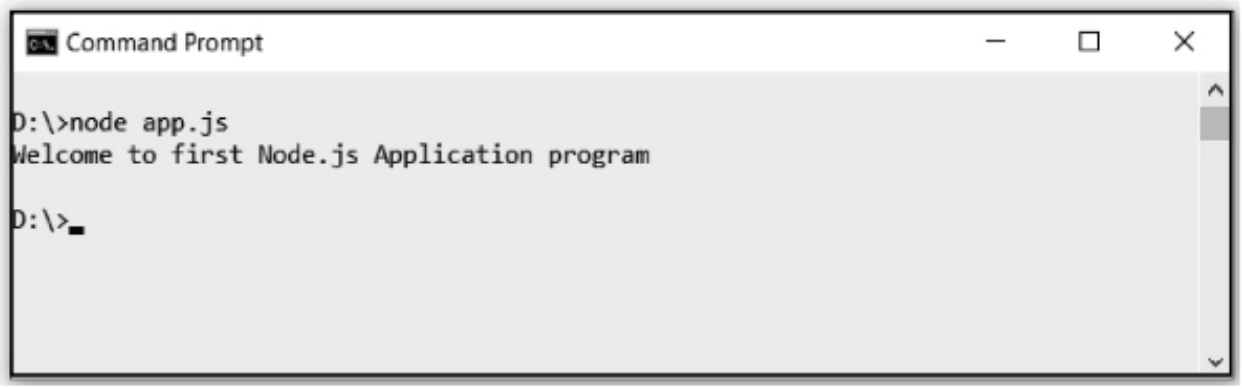
  In order to verify the installation we have to open a command prompt and type the following command –

- If the node.js is successfully installed, then some version number of the node.js which you have installed in your PC will be displayed.
- After successful installation we can now execute the Node.js document. The Node.js file has extension .js.
- Following is a simple node.js program which can be written in notepad.

app.js

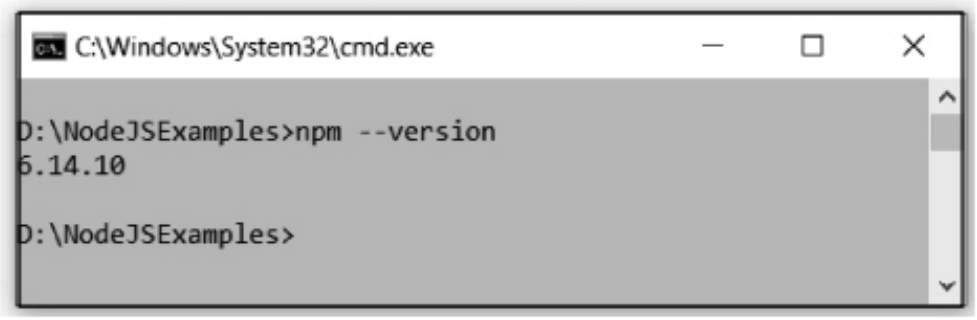
```
console.log("Welcome to first Node.js Application program");
```

Output



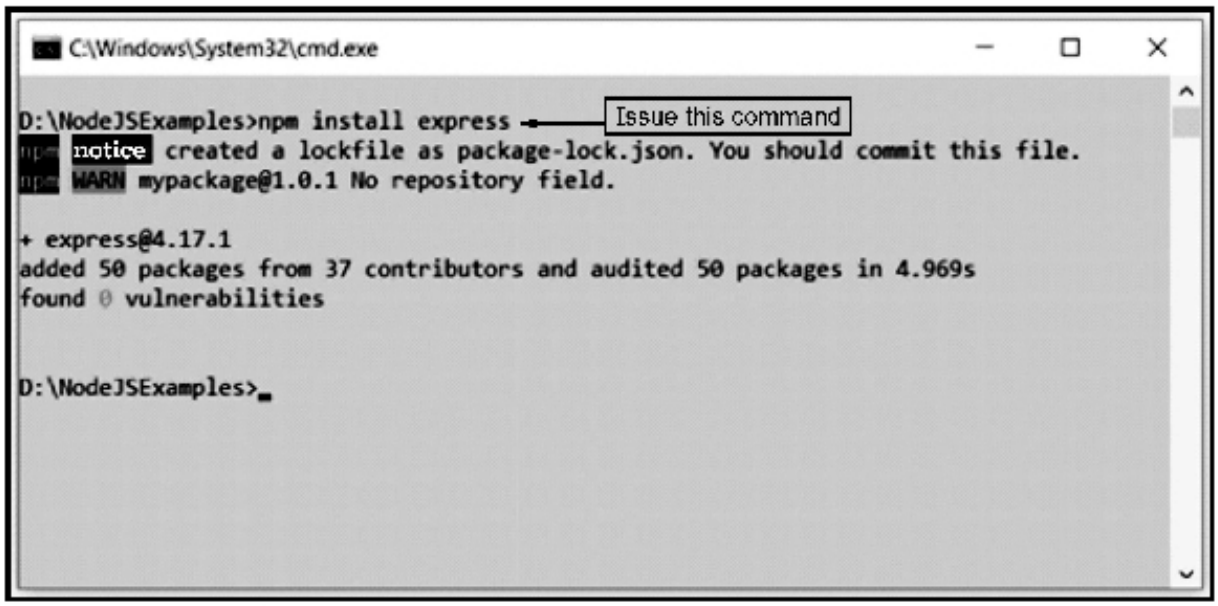
4.2 Package Manager

- A package in Node.js contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.
- The NPM stands for node package manager.
- NPM consists of two main parts:
  1) a **CLI (command-line interface)** tool for publishing and downloading packages, and
  2) an **online repository** that hosts JavaScript packages
- NPM gets installed along with the installation of node.js. To verify the presence of npm package on your machine install following command –



Installing Modules using npm

For installing any node.js module we have to use **install** command. For example –

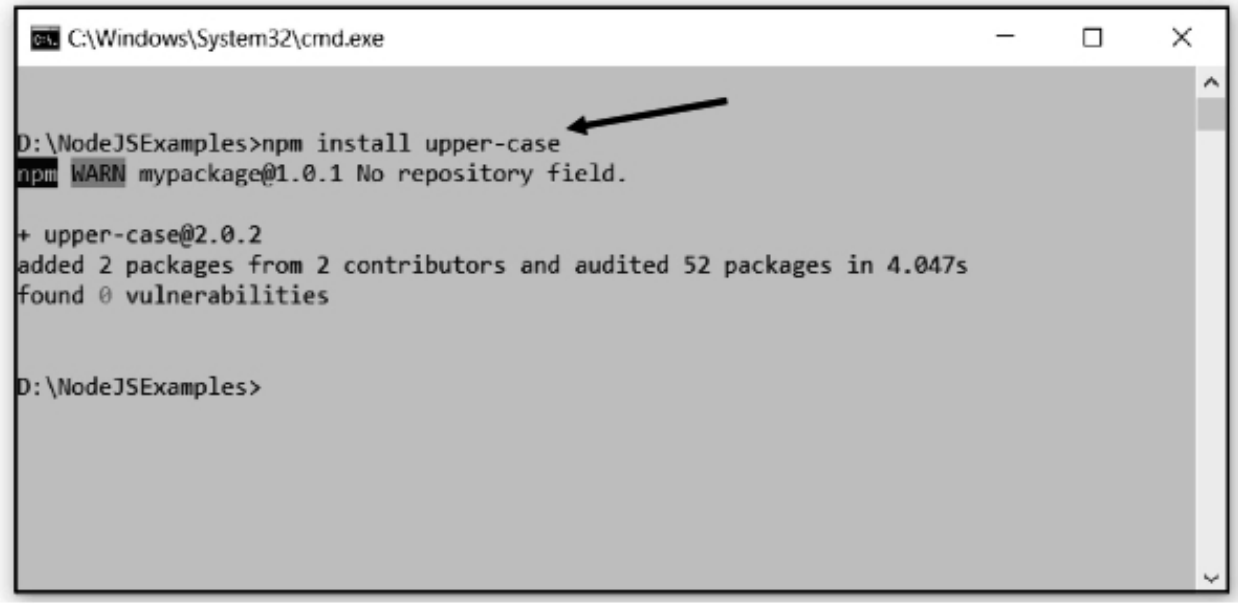


- As soon as we issue this command the package-lock file gets generated in the current folder.
- For using this module in our program we have to use following command –

```
var ex = require('express');
```

Example of using the existing module

Step 1 : We will install the module **upper-case** by issuing the install command



Step 2 : You can locate this module inside **your_folder/node_modules**. Now open any suitable text editor and write node js script that makes use of this module.

moduleDemo.js

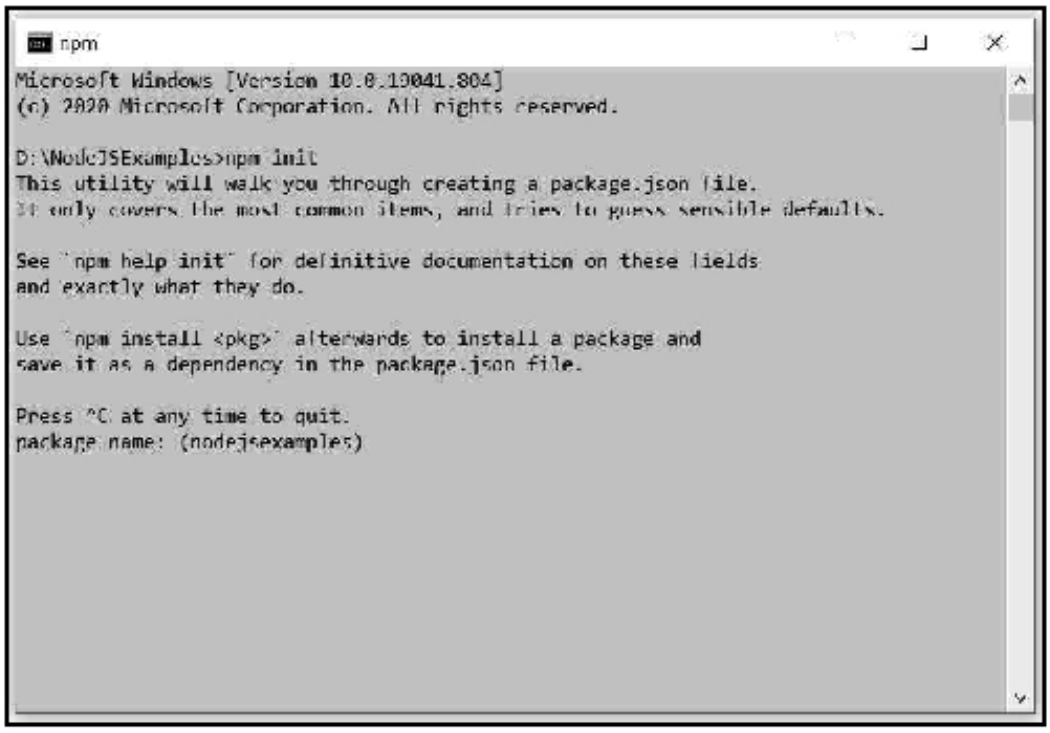
```
var upp_ch = require('upper-case');
console.log(upp_ch.upperCase("i am proud of you"));
```

Output

Use of npm init

- We can create our own package using following steps

Step 1 : Create a sample folder. Open command prompt and go to that directory location. For instance – I have created a folder named **nodejsexamples** . Issue the command **npm init** at the command prompt.



Step 2 : Now fill the information about the package.

Once you execute the above commands, the **package.json** gets created. You can open and test it using simple notepad.

4.3 Features

Following are some remarkable features of node.js –

1) **Non blocking thread execution :** Non blocking means while we are waiting for a response for something during our execution of tasks, we can continue executing the next tasks in the stack.
2) **Efficient :** The node.js is built on V8 JavaScript engine and it is very fast in code execution.
3) **Open source packages :** The Node community is enormous and the number of permissive open source projects available to help in developing your application in much faster manner.

4) **No buffering :** The Node.js applications never buffer the data.

5) **Single threaded and highly scalable :** Node.js uses a single thread model with event looping. Similarly the non blocking response of Node.js makes it highly scalable to sever large number of requests.

Review Question

1. *List and explain the features of node.js*

4.4 Console Object

The console object is a global object which is used to print different levels of messages to stdout and stderr.

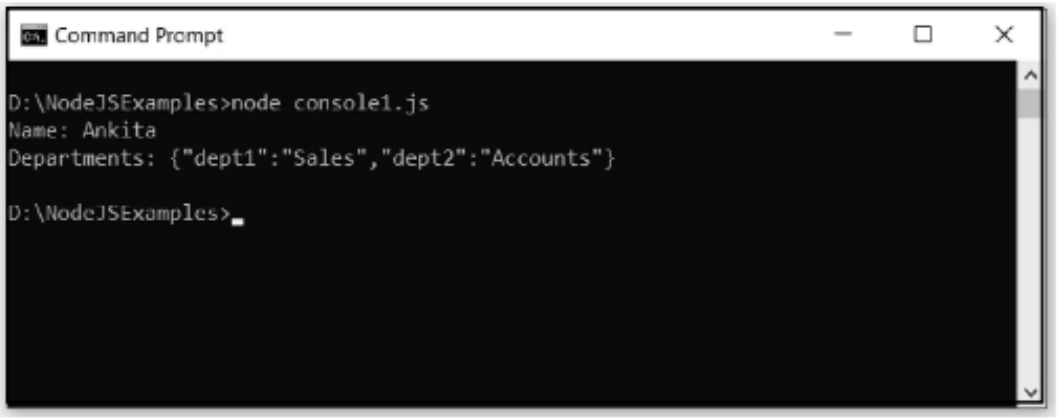
The most commonly used methods console object are console.log console.info, console.warn and console.error. Let us discuss them with illustrative examples

1) **console.log([data][, ...]) :** It is used for printing to stdout with newline. This function can take multiple arguments similar to a printf statement in C.

console1.js

```
var emp_name = 'Ankita',
        emp_depts = {
        dept1: 'Sales', dept2: 'Accounts'
        };
console.log('Name: %s'+'\n'+ 'Departments: %j',emp_name, emp_depts);
```

Output



2) **console.info([data][, ...]) :** This method is similar to console.log.

console2.js

```
var emp_name = 'Ankita',
   emp_depts= {
```

```
    dept1: 'Sales',
      dept2: 'Accounts'};
  console.info('Name: %s'+'\n'+ 'Departments: %j',emp_name, emp_depts);
```

Output

```
Name: Ankita
Departments: {"dept1":"Sales","dept2":"Accounts"}
```

3) **console.error ([data][, ...]) :** This method prints to stderr with newline

console3.js

```
var code = 401
console.error('Error!!!',code)
```

Output

```
Error!!! 401
```

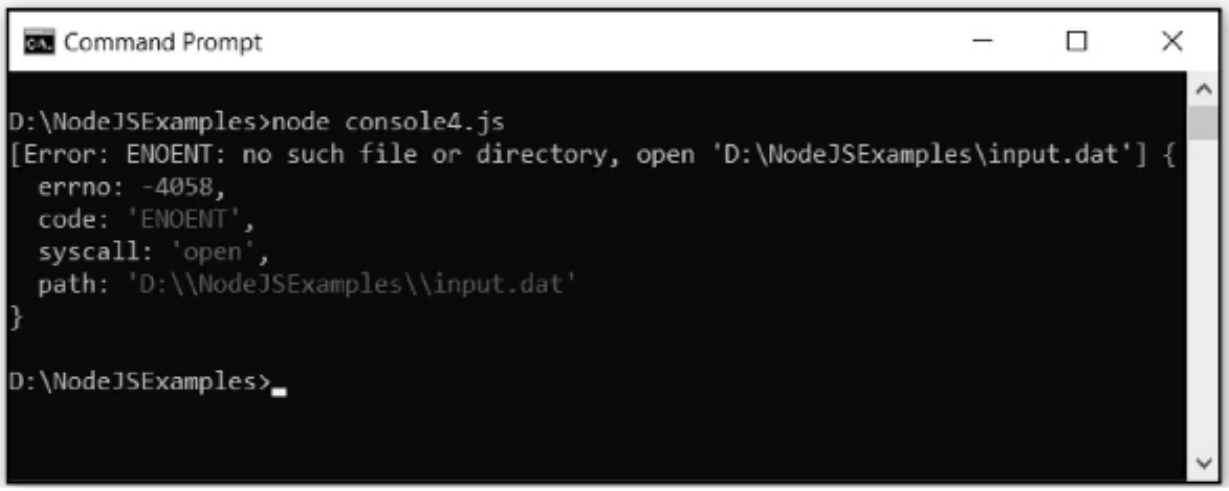
4) **console.warn([data][, ...]) :** This method is similar to console.error() method.

Consider following example – in which using the **fs** module we open a text file. If the file does not open successfully then console.warn() method displays the warning message.

console4.js

```
var fs = require('fs');
fs.open("input.dat","r",function(err,fs){
    if(err)
        console.warn(err);
    else
        console.log("File opening successful!!!");
})
```

Output



Review Question

1. *Explain any four methods of console object in node.js with suitable examples.*

4.5 Concept of Callbacks

What is callbacks ?

Callbacks is a is a function which is usually passed as an argument to another function and it is usually invoked after some kind of event.

How it works ?

The callback function is passed as an argument to another function. When particular event occurs then only it is invoked.

By the time callback function is executing another task gets executed.

Hence execution of callbacks is asynchronous execution.

Example Code

Step 1 : Create a simple text file named "myfile.txt" as follows –

myfile.txt

```
How are you?
Jack and Jill went up the hill,
to fetch a glass of water
```

Step 2 : Create a JavaScript file(.js) that contains the callback function

callbackDemo.js

```
var fs = require("fs");
console.log("Serving User1")

fs.readFile('myfile.txt', function (err, data) {
  if (err) return console.error(err);
  console.log("*** Contents of the File are...***");
  console.log(data.toString());
});
console.log("Serving User2");
console.log("Serving User3");
console.log("Good Bye!!!");
```

Output

```
D:\NodeJSExamples>node callbackDemo.js
Serving User1
Serving User2
```

```
Serving User3
Good Bye!!!
*** Contents of the File are...***
How are you?
Jack and Jill went up the hill,
to fetch a glass of water
```

Program Explanation : In above program,

5) We have to read a file named **myfile.txt**, but while reading a file it should not block the execution of other statements, hence a call back function is called in which the file is read.

6) By the time file gets read, the remaining two console functions indicating "Serving user 1" and "Serving user 2" are not blocked, rather they are executed, once the contents are read in the buffer, then those contents of the file are displayed on the console.

Review Question

1. *Explain the callbacks in node js with suitable example.*

Notes

5 Node JS in Details

Syllabus

Events and Event Loop, timers, Error Handling, Buffers, Streams, Work with File System, Networking with Node (TCP, UDP and HTTP clients and servers), Web Module, Debugging, Node JS REST API, Sessions and Cookies, Design patterns, caching, scalability.

Contents

5.1 Events and Event Loop

5.1.1 Introduction to Events

- Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.

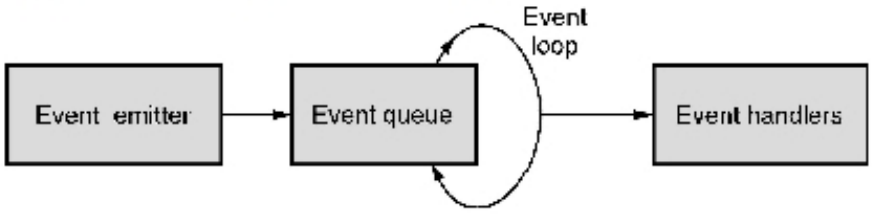


Fig. 5.1.1 Event driven architecture in node.js

What is Event ?

- Every time when user interacts with the webpage, event occurs when user clicks mouse button or presses some key on the keyboard.
- When events are triggered, some functions associated with these events get executed.
- Event driven programming makes use of the following concepts –
  i) When some events occur, an event handler function is called. This is basically a call back function.
  ii) The main program listens every event getting triggered and calls the associated event handler for that event.

Concept of Event Emitter :

- The **Eventmitter** is a module that facilitates communication between objects.
- The emitter objects performs following tasks –
  1) It emits named events.
  2) Registers and unregisters listener functions.
- Basically **EventEmitter** is a class which can be used to raise and handle custom events.

Steps for using event handling in Node.js

Step 1 : import the 'events' module and create an instance of it using following code –

```
var events = require('events');
```

Step 2 : Then create an instance of **EventEmitter()** class. This instance is created because we have to call on() and emit() functions using this instance.

Step 3 : Then define a callback function which should be called on occurrence of event. This function is also called as event listener funciton.

Step 4 : Then register the event using **on()** function. To this function name of the event and callback function is passed as arguments.

Step 5 : Finally raise the event using **emit()** function.

Example Code

eventDemo1.js

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

var myfunction = function() {
    console.log("When event occurs, My Function is called")
}

//Bind FirstEvent with myfunction
em.on('FirstEvent', myfunction);

// Raising FirstEvent
em.emit('FirstEvent');
```

Output

```
D:\NodeJSExamples>node eventDemo1.js
When event occurs, My Function is called.
```

Program Explanation : In above program,

1) We have to import the 'events' module.
2) Then we create an object of EventEmitter class.
3) For specifying the event handler function, we need to use **on()** function.
4) The **on()** function requires two parameters - Name of the event to be handled and the callback function which is called when an event is raised.
5) The **emit()** function raises the specified event.

Example Code for Passing data along with event

We can pass some data inside the event, following example illustrate the same -

eventDemo1.js

```
// get the reference of EventEmitter class of events module
var events = require('events');
//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();
var myfunction = function(data1,data2) {
    console.log("When event occurs, My Function is called");
    console.log("Data passed is: "+data1+"  "+data2);
}
//Bind FirstEvent with myfunction
em.on('FirstEvent', myfunction);
// Raising FirstEvent
em.emit('FirstEvent','data1','data2');
```

Output

```
D:\NodeJSExamples>node eventDemo1.js
When event occurs, My Function is called
Data passed is:  data1   data2
```

5.1.2 Concept of Event Loop

- Node Js is a single threaded system. So it executes the next task only after completion of previous task.
- The event loop allows Node.js to perform non-blocking I/O operations evenif the JavaScript is single threaded.
- Event loop is an endless loop which waits for tasks, executes these tasks and sleeps until it receives next task.
- The event loop allows us to use callback functions.
- The event loop executes task from event queue only when the call stack is empty. Empty call stack means there is no ongoing task.
- Event loop makes execution fast.

Event Loop Execution Model

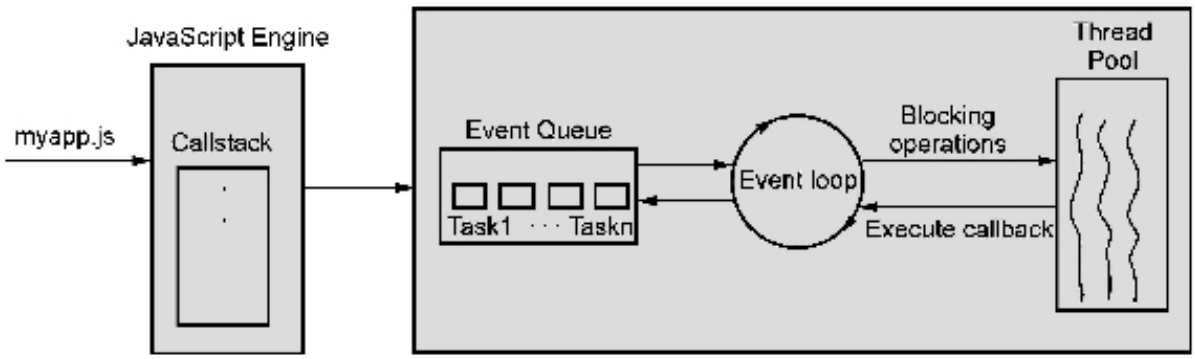


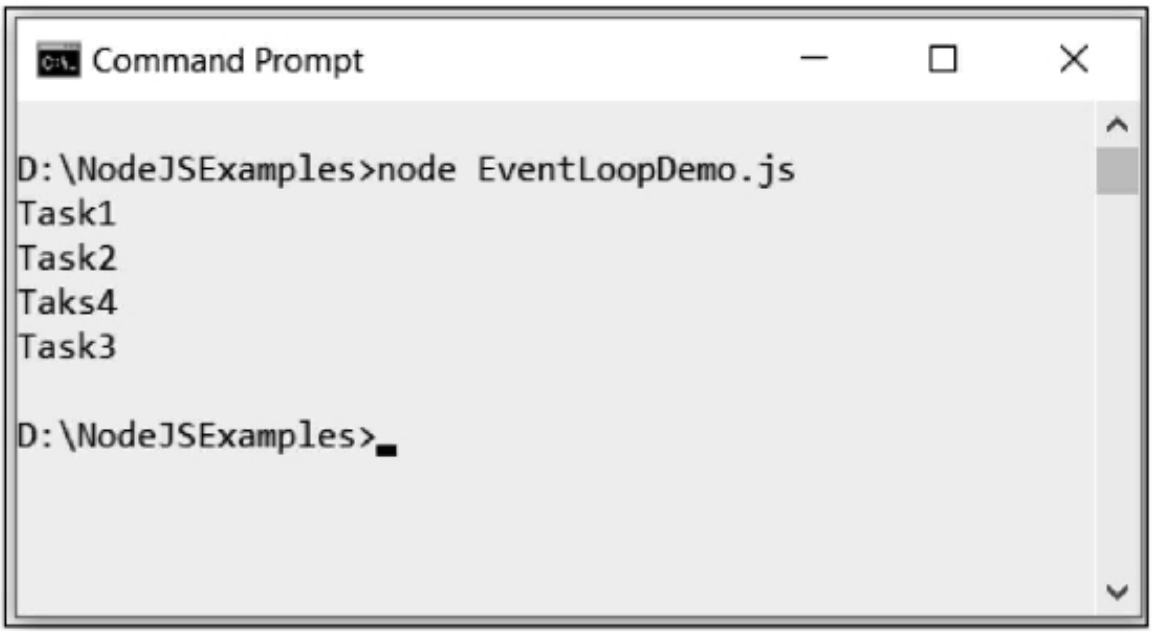
Fig. 5.1.2 Event loop execution

- The Event Loop is a main loop **running continuously** for executing the callback functions. When an async function executes, the callback function is pushed into the queue. The JavaScript engine doesn't start processing the event loop until the code after an async function has executed.

Example Code

```
console.log("Task1");
console.log("Task2");
setTimeout(function() {console.log("Task3") },1000);
console.log("Taks4");
```

Output



Phases of Event Loop

Various phases of event loop are –

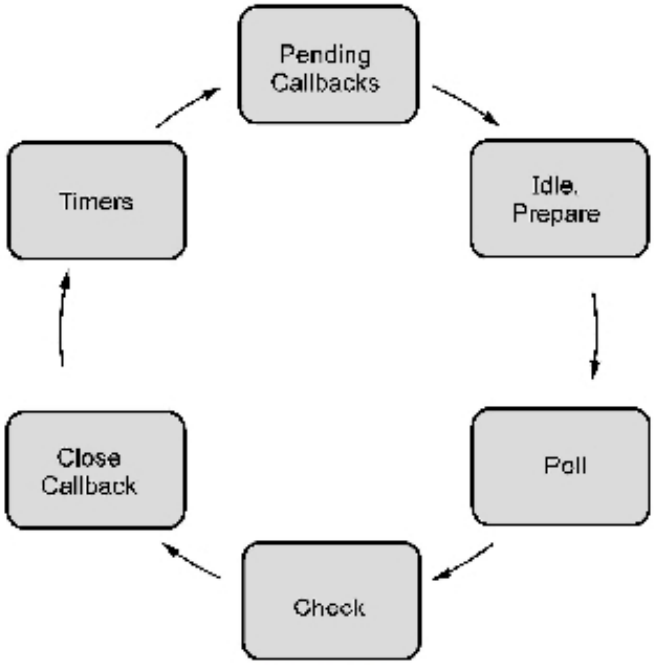


Fig. 5.1.3

1) **Timers :** This is the first phase in the event loop. It finds expired timers in every iteration (also known as Tick) and executes the timers callbacks created by setTimout and setInterval.
2) **Pending callbacks :** It handles I/O callbacks deferred to the next iteration, such as handling TCP socket connection error.
3) **Idle, prepare :** It is used internally.
4) **Poll :** The Poll phase calculates the blocking time in every iteration to handle I/O callbacks.
5) **Check :** This phase handles the callbacks scheduled by setImmediate(), and the callbacks will be executed once the Poll phase becomes idle.
6) **Close callback :** This phase handles callbacks if a socket or handle is closed suddenly and the 'close' event will be emitted.

5.2 Timers

- Timers module in Node js provides a way for scheduling the functions to be executed on some given time.
- For using the timers module in Node.js it is not required to import it via **require()**

function as all the methods are globally available to JavaScript API. Hence we do not have to use **require** in the source code while use timer functions.

- There are **three** set timer functions used in Node.js

1. setTimeout()	2. setImmediate()	3. setInterval()

Let us understand them with suitable examples -

1) setTimeout() : This function is used to set one time callback after delaying by some milliseconds.

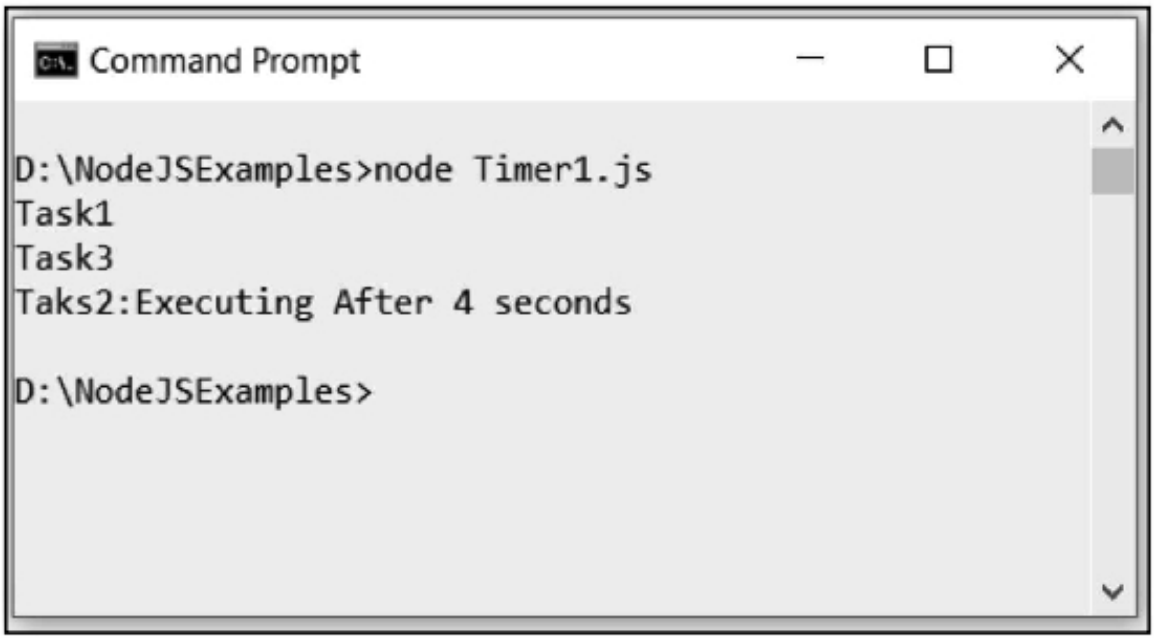
Syntax

```
setTimeout(function,delay_in_milliseconds)
```

Timer1.js

```
console.log("Task1");
setTimeout(function() {console.log("Taks2:Executing After 4 seconds") },4000);
console.log("Task3");
```

Output



Example code for the function having some parameters

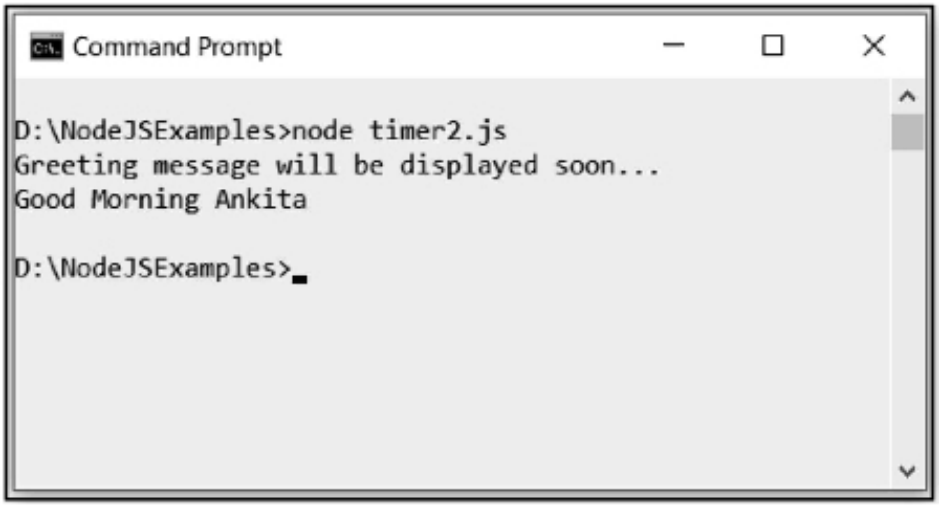
We will now discuss to use **setTimeout** function for accepting a function with parameters. The function that will be passed to setTimeout along with parameters has following syntax

```
setTimeout(function_name, delay_in_milliseconds,parameter1,parameter2, ...);
```

Timer2.js

```
console.log("Greeting message will be displayed soon...");
var myfun = (user) => {
    console.log("Good Morning "+user);
};
setTimeout(myfun,1000,"Ankita");
```

Output



2) setImmediate() : This function schedules the immediate execution of callback function after I/O event.

The syntax is

```
setImmediate(() => {
  //function body
})
```

We don't need to pass any time interval to this method. The first argument is the function to execute. You can also pass any extra arguments to this function .

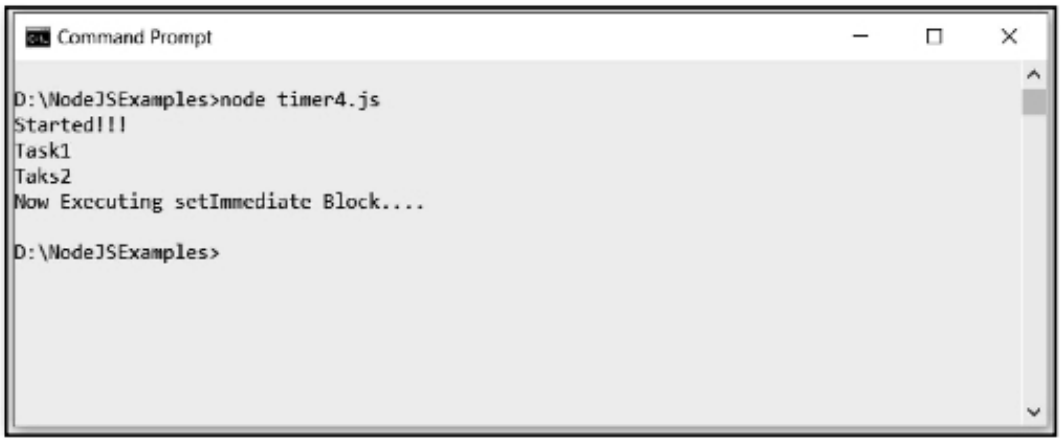
This method will execute a function right after the next code blocks which is present after the **setImmediate** function.

For example -

```
console.log("Started!!!");
setImmediate(() => {
    console.log("Now Executing setImmediate Block....");
});
console.log("Task1");
console.log("Taks2");
```

Note that these **console.log** statements will execute first and then immediately after that the setImmediate function will execute.

Output



3) setInterval() : This function helps to execute a task in a loop after some time interval. This function has the same syntax as that of **setTimeout.**

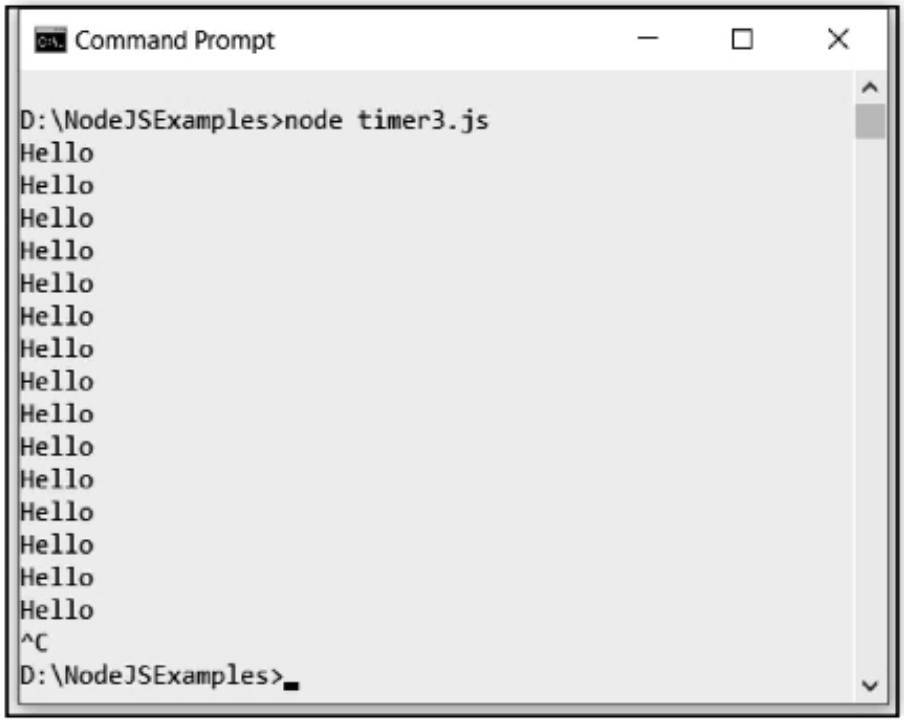
Syntax

```
timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

For example

```
    var myfun = () => {
        console.log("Hello");
};
setInterval(myfun,1000);
```

Output



The above code present in the function **myfun** gets executed after every 1000 milliseconds. To break this program running in a loop we have to press control+c.

5.3 Error Handling

When some error occurs in asynchronous programming, we must use **throw** statement.

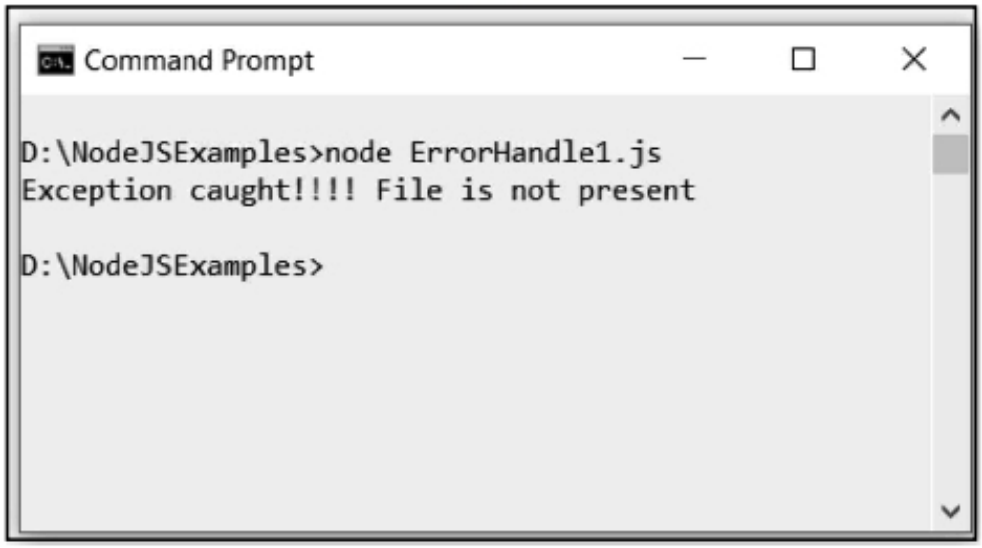
The error is treated as an event and using **on()** method we use the **uncaughtException** to handle error gracefully.

Following example shows how to handle the error in the simplest manner

ErrorHandle1.js

```
var fs = require('fs');
fs.readFile("inp.txt",function(error, data){
    if(error)
        throw error;
    console.log(data);
});
process.on("uncaughtException", function(error){
    console.log("Exception caught!!!! File is not present");
});
```

Output



5.4 Buffers, Streams, Work with File System

5.4.1 What is Buffer ?

- Buffer is class that allows us to store raw data similar to arrays.
- It is basically a class that provides the instances for storing the data.

- It is a global class so we can use it without any need of importing a buffer module.

Creation of Buffer

Creating a buffer is very simple. For example

```
var buffer = new Buffer(10);
```

This will create a buffer of 10 units.

Similarly we can create a buffer as follows –

```
var buffer = new Buffer([1,2,3,4,5],"utf8");
```

The second parameter which we have passed here is encoding scheme. It allows various encodings such as utf8(this is default one), ascii,ucs2,base64, or hex.

Writing to Buffer

The syntax for writing to the buffer is

```
write(string[,offset],[,length][,encoding])
```

where

string : It is the string to be written to the buffer.

offset : It is the index from where we can start writing. The default value is 0

length : It indicates the number of bytes to be written to the buffer.

Encoding : It indicates the desired encoding. The default one is "utf8".

Example Code

Following is a simple node js program that shows how to write data to the buffer.

BufferExample.js

```
var buffer = new Buffer(50);
buffer.write("Good Morning");
console.log(buffer.toString('utf8'));
```

Output

```
Good Morning
```

Program Explanation: In above program,

We have used **write** method for writing to the buffer. For displaying the string written to the buffer we have to use **toString** method along with the encoding scheme.

Reading from Buffer

The syntax for reading from buffer is as follows –

```
toString([encoding][,start][,end])
```

where

encoding : It indicates the encoding scheme. The default one is "utf8".

start : Beginning index is indicated by this value. The default is 0.

end : Ending index is indicated by this value.

Example Code

```
var buffer = new Buffer(50);
buffer.write("Good Morning");
console.log(buffer.toString('utf8'));
```

5.4.2 Concept of Stream

Stream can be defined as objects that reads data from source or writes the data to the destination in continuous fashion.

The stream is an EventEmitter instance that emits several events. Following are the events that occur during stream operations –

1) **data :** This event is fired when we want to read the data from the file.
2) **end :** When there is no more data to read then this event is fired.
3) **error :** If any error occurs during reading or writing data then this event is fired.
4) **finish :** When data is completely flushed out then this event occurs.

There four types of streams,

1) **Readable :** This stream is used for read operation only.
2) **Writable :** This stream is used for write operation only.
3) **Duplex :** This type of stream is used for both read and write operation.
4) **Transform :** In this type of stream the output is computed using input.

Write Operation

The steps that are followed to write data to the stream using node.js are as follows -

Step 1 : Import the file stream module **fs** using **require**.

Step 2 : Create a writable stream using the method **createWriteStream.**

Step 3 : Write data to the stream using **write** method.

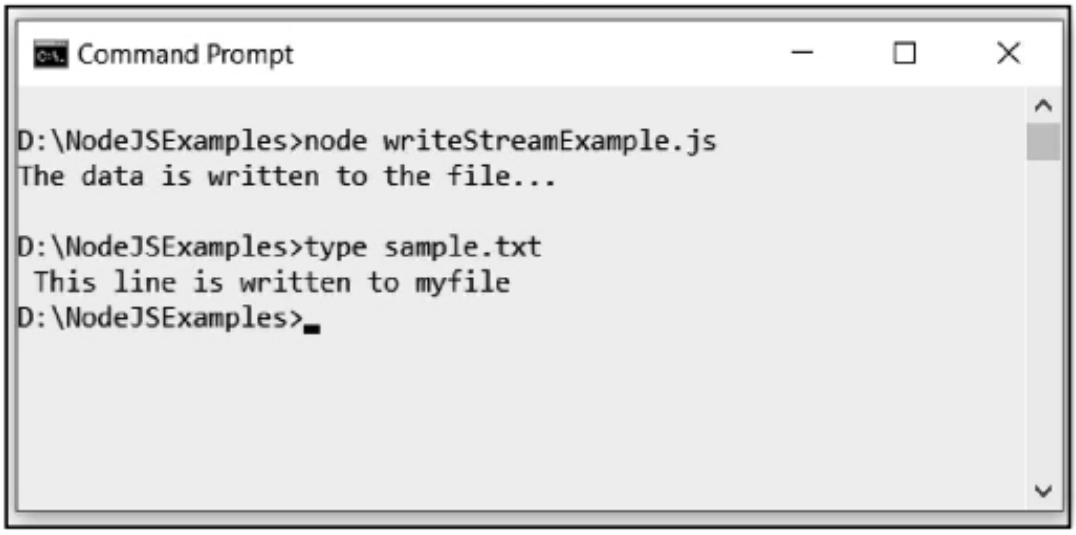
Step 4 : Handle the events such as finish or error(if any)

Following Nodejs script shows how to write data to the string,

writeStreamExample.js

```
var fs = require("fs");
var str = " This line is written to myfile";
var ws = fs.createWriteStream('sample.txt');
ws.write(str,'utf8');
ws.end();
console.log("The data is written to the file...");
```

Output



The above code is simply modified to handle the error event as follows -

```
var fs = require("fs");
var str = " This line is written to myfile";
var ws = fs.createWriteStream('sample.txt');
ws.write(str,'utf8');
ws.end();
console.log("The data is written to the file...");

ws.on('error',function(err) {
        console.log(err.stack);
});
```

Read Operation

For reading from the stream we use createReadStream. Then using the **data** event the data can be read in chunks.

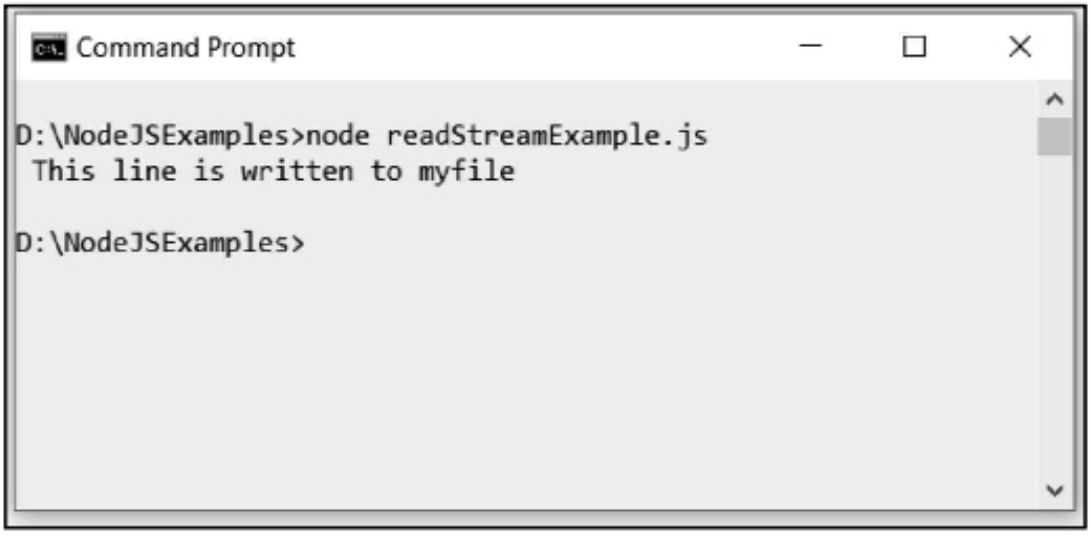
Example Code

```
var fs = require("fs");
var str = "";
var rs = fs.createReadStream('sample.txt');
rs.setEncoding("utf8");

//handling stream event 'data'
rs.on('data',function(chunk){
    str += chunk;
});

//handling stream event 'end'
rs.on('end',function(){
    console.log(str)
});

//handling stream event 'error'
rs.on('error',function(err){
    console.log(err.stack);
});
```

Sample.txt

```
This line is written to myfile
```

Output



Pipe Operation

The pipe operation is a kind of operation in which output of one stream acts as an input of another stream.

There is no limit on pipe operation that means, all the output of one stream can be fed as input to input to another stream.

Following are the steps used for using the pipe operation in your node.js program –

Step 1 : Import the **'fs'** module using **require.**

Step 2 : Create readable stream using **createReadStream** function.

Step 3 : Create writable stream using **createWriteStream** function.

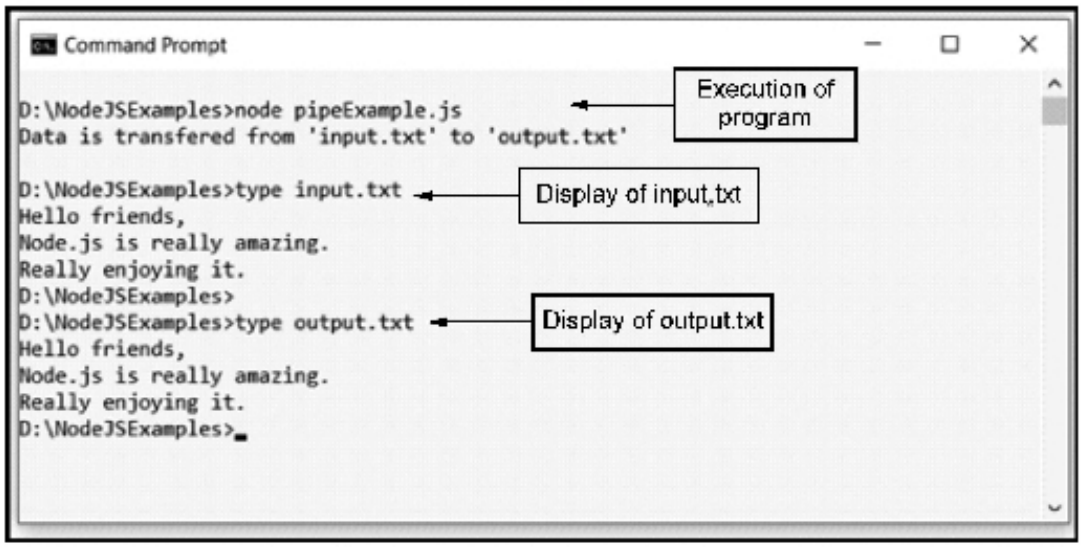
Step 4 : Use the pipe() function for passing the data from readable stream to writable stream.

Following example shows the use of pipe operation -

pipeExample.js

```
var fs = require("fs");
var rs = fs.createReadStream('input.txt');
rs.setEncoding("utf8");
var ws = fs.createWriteStream('output.txt');
rs.setEncoding("utf8");
rs.pipe(ws);
console.log("Data is transfered from 'input.txt' to 'output.txt' ");
```

Output



5.4.3 File System

File system fs is implemented in the node.js using require function.

var fs = require('fs');

The common operations used with the file system module are -

1. Read file 2. Create file 3. Update file 4. Delete file

1. Read Operation

The read file operation can be performed **synchronously** or **asynchronously**.

For reading the file synchronously we use **readFileSync** method. And for reading the file synchronously **readFile**

Normally while reading the file using asynchronous form we use callback function as the last parameter to the **readFile** function. The form of this callback function is,

```
function(err, data)
{
    //function body
}
```

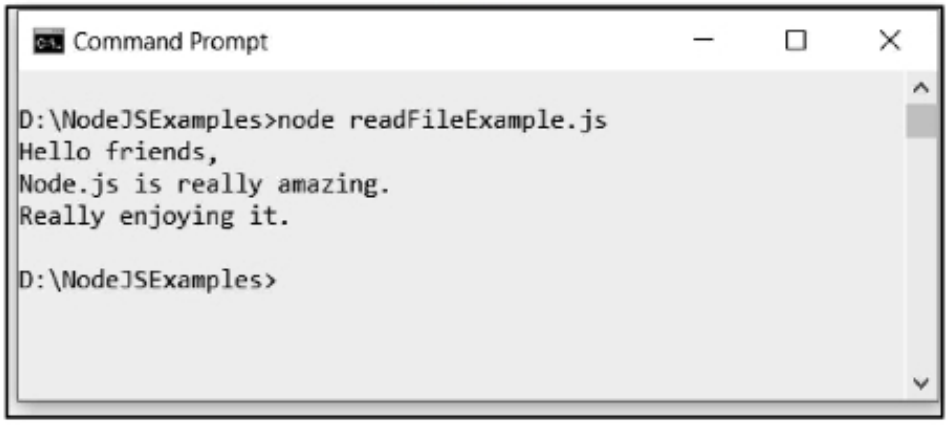
Example code

```
var fs = require("fs");
fs.readFile('input.txt',function(err,data){
    if(err)
        console.log(err);
    console.log(data.toString());
});
```

input.txt

```
Hello friends,
Node.js is really amazing.
Really enjoying it.
```

Output



Similarly we can read the file synchronously using following command -

readFileExample.js

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log(data.toString());
```

2. Create File Operation

We can create an empty file using the command **open()**

Syntax

```
open(path,flags[,mode],callback)
```

where

path : It is the path of the filename.

flag : It indicates, the behaviour on opening a file. That means "r" is open file for reading, "w" is for writing, "rs" is for synchronous mode, "r+" or "w+" means open a file for both read and write purpose and so on.

mode : it indicates readable or writable mode.

Callback : It is basically a function which has two arguments(err,data)

Example Code

```
var fs = require("fs");
fs.open('myfile.txt', 'w',function(err,file){
    if(err)
        console.log(err)
    console.log('File created!!!');
});
```

3. Update/Write File Operation

For writing to the file two operations are used most commonly -

1. appendFile() and 2. writeFile()

The **appendFile()** method is used to write the contents at the end of the specified file.

Example Code

```
var fs = require("fs");
str = "This line is written to the file";
fs.appendFile('myfile.txt', str,function(err){
    if(err)
        console.log(err)
    console.log('File appended!!!');
});
```

Output

The **writeFile()** method is used to write the contents to the file

Example Code

```
var fs = require("fs");
str = "This line is replacing the previous contents";
fs.writeFile('myfile.txt', str,function(err){
    if(err)
        console.log(err)
    console.log('File Writing Done!!!');
});
```

Output

4. Delete File Operation

To delete the file, we use **unlink()** method.

Syntax

```
unlink(path, callback)
```

where

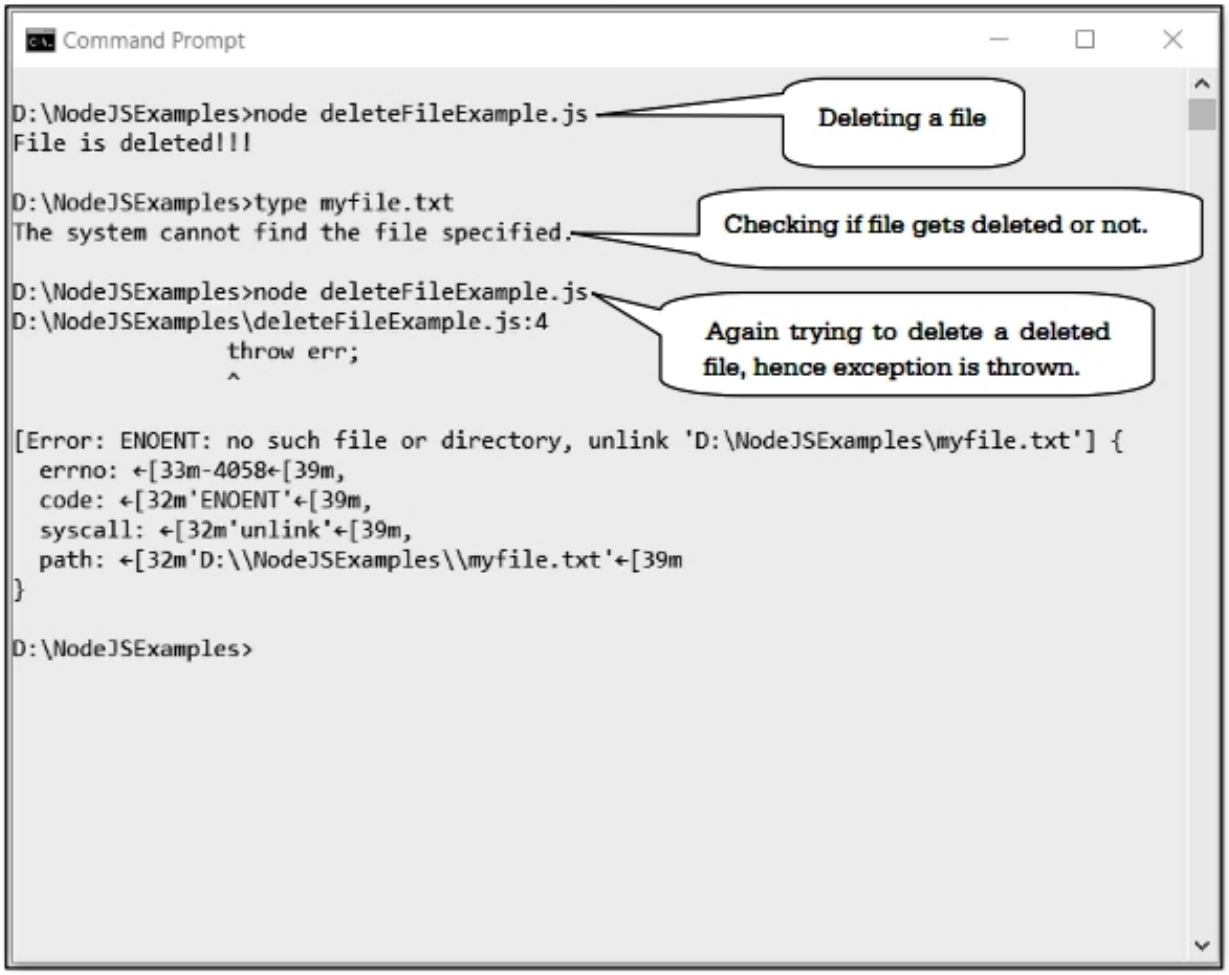
path : Is a path of the file to be deleted.

callback : It is a callback function with the only one argument for 'error'.

Example Code

```
var fs = require("fs");
fs.unlink('myfile.txt',function(err){
    if(err)
        throw err;
    console.log("File is deleted!!!");
});
```

Output



5.5 Networking with Node

Basics of Network Programming

- Socket is the most commonly used term in network programing. Socket provides the way by which the computers can communicate using **connection oriented** or **connection less communication**.
- A socket is basically an **endpoint** of a two-way communication link between two programs running on the network. Typically these programs are Server program and Client program.
- Thus socket is OS-controlled interface into which the applications can send or receive messages to and fro from another application.
- A socket is bound to a port number so that the TCP/UDP from transport layer can identify the corresponding application at destination.

Creating TCP server-client communication Application

- Transmission Control Protocol(TCP) is a connection-oriented, reliable protocol which supports the transfer of data in continuous streams.
- The addressing scheme used in TCP is makes use of **ports**. On separate ports the communication can be established concurrently by the system. During this communication, the server waits for the client to get connected to a specific port for establishing communication. This type of communication is called as **socket programming.**
- There are some ports which are reserved for specific service. These ports are called as reserved ports. Various port numbers specifying their services are given in the following table

Port number	Service
21	FTP
23	Telnet
25	SMTP
80	HTTP
110	POP3

- **User level processes** or services generally use port numbers **>=1024.**
- A socket is bound to a port number so that the TCP/UDP from transport layer can identify the corresponding application at destination.

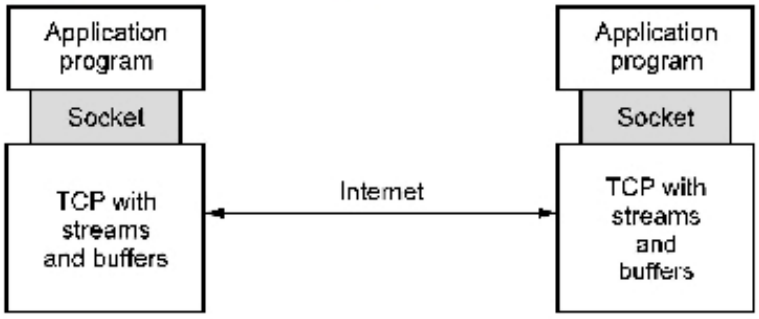


Fig. 5.5.1 Client Server communication using TCP

- **Server** is a device which has resources and from which the services can be obtained. For example : there are various types of servers such as web server which is for storing the web pages, there are print servers for managing the printer services or there are database servers which store the databases.
- **Client** is a device which wants to get service from particular server.
- First of all server starts and gets ready to receive the client connections. The server-client communications occurs in following steps

Sr. No.	Client	Server
1.	Creates TCP socket.	Creates TCP socket.
2.	Client initiates the communication. i) Communicate.	Creates another socket for listening client. i) Accept message from client and repeatedly communicate.
3.	Close the connection.	Close the connection.

We have two build to node.js programs - One for server and other for client.

TCP Server Program

To build the TCP server program we have to follow the steps as given below -

Step 1 : We use "net" module by adding following line at the beginning of both the programs -

```
var net = require("net");
```

Step 2 : Now we use the **createServer()** method for creating the stream based TCP server. For that purpose we have to add following code in the program.

```
var server = net.createServer();
```

Step 3 : We use the event handler **on** for the server when any client gets connected to it. In this event handling code we use the **socket** class. The attributes

remoteAddress and **remotePort** will return the remote address and port number of the client which is connected currently to the server.

Step 4 : Then using socket instance we can handle three events such as **data,close** and **error.**

- For **data** event, the data received from the client is displayed on the console using **cosole.log** method.
- For **close** event, the connection closing message is displayed to the client.
- If any error occurs during establishment of connection, the **error** event is fired, and error message is displayed on the console.

Step 5 : During this entire, client server communication Server must be in listening mode. This can be done using **listen** method. To this method, the port number is passed. The **same port number** must be used in the client program, so that both the server and client can communicate on the same port number.

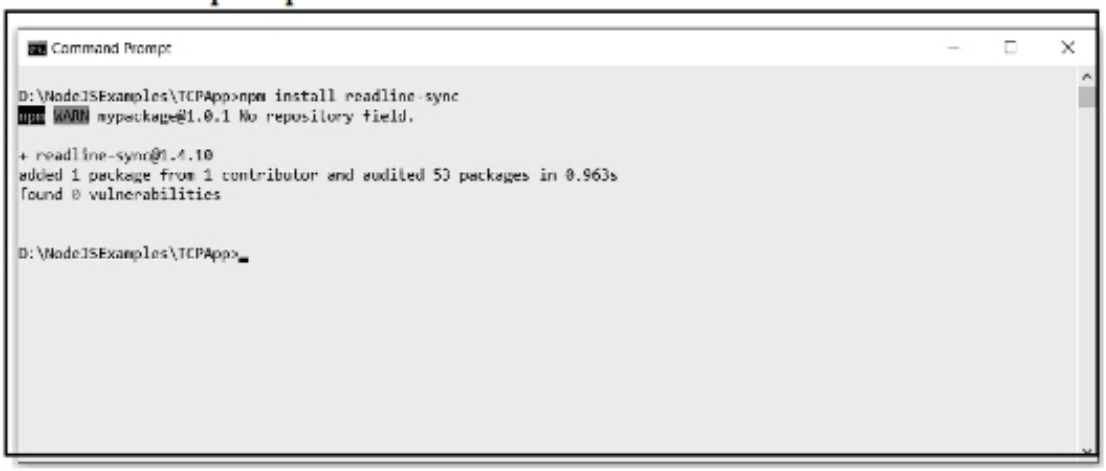
The complete code for server program using TCP is as given below -

server.js

```
var net = require("net");

var server = net.createServer();

server.on("connection",function(socket){
    var clientAddress = socket.remoteAddress+":"+socket.remotePort;
    console.log("Connection with the client %s is established!!!",clientAddress);

    socket.on("data",function(mydata){
        console.log("\n Data received from client is: %s",mydata.toString());
        //sever sending data to the client
        socket.write("I am fine");
    });
socket.once("close",function(){
        console.log("Connection with %s is closed!!!",clientAddress);
});
socket.on("error",function(err){
cosole.log("Connection error %s",err.message);
});
});
server.listen(8082, function(){
    console.log("Server is listening to %j",server.address());
});
```

TCP Client Progtam

Following is a client program, in which we are getting some input from the user through command prompt and sending that data to the server. For reading the input from keyboard we need to install **readline-sync** module. For installation of this module, open the command prompt. The installation of this module is as follows -



To build the TCP client program we have to follow the steps as given below -

Step 1 : We use "net" module by adding following line at the beginning of the client program.

```
var net = require("net");
```

Step 2 : As our server is running on port number 8082 and the **localhost** as the host, the client must run on the same for establishing the communication between server and the client. The instance of socket class is created and it is named as var **client.**

```
const client = new net.Socket();
```

Step 3 : Now it is possible to establish a connection with the server using **connect** method. For establishing the connection we should pass the same hostname and port number as parameter.

Step 4 : Client can send the data to the server using **write** method. Here we are getting the data when user enters some data through keyboard. For getting the data from the console, we use the **readline-sync** module's **question** method. The required code for this operation is -

```
var read_line = require("readline-sync");

var mydata = read_line.question("Enter some data for sending it to server: ")
    client.write(mydata);
```

Step 5 : Two more events can be handled by the client program those are – **data** and **end.**

- On the **data** event, the data received from the server is displayed on the console window, using **console.log**

```
client.on("data",function(d){
    //body
});
```

- On the **end** event, the we can acknowledge the server than now client is going to end.

```
client.on("end",function(){
    //body
});
```

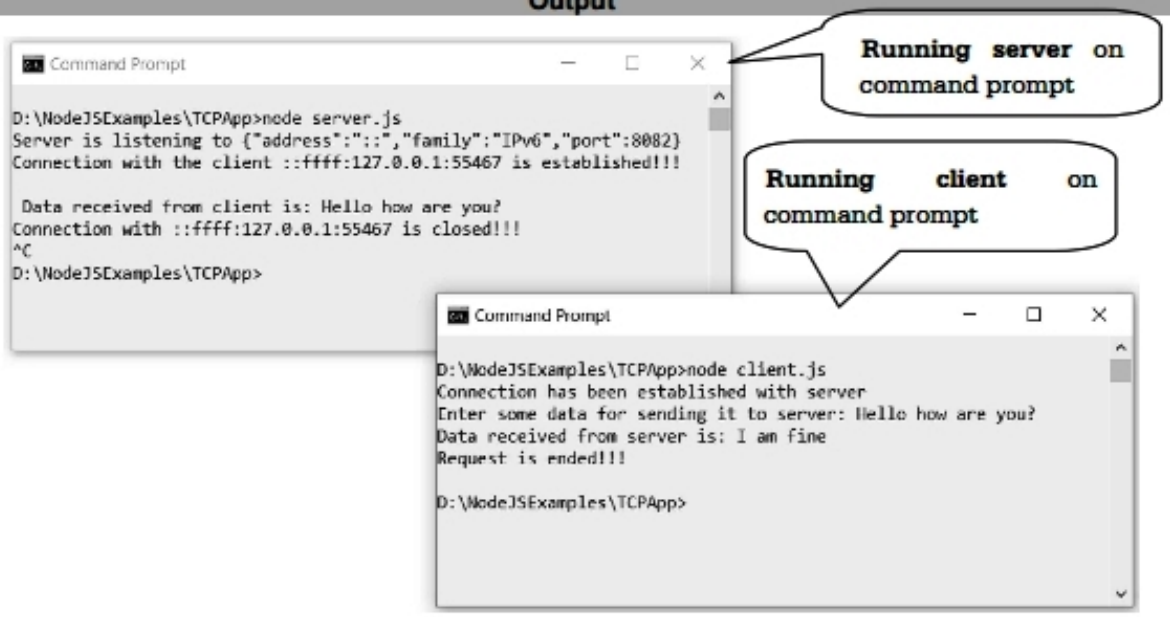
The complete code for client program using TCP is as given below -

client.js

```
var net = require('net');
var read_line = require("readline-sync");

const PORT = 8082;
const HOST = 'localhost';
// create a TCP client
const client = new net.Socket();

client.connect(PORT,HOST,function(){
    console.log("Connection has been established with server");
    //client sending data to server
    var mydata = read_line.question("Enter some data for sending it to server: ")
    client.write(mydata);

});
client.on("data",function(d){
    console.log("Data received from server is: %s",d);
    client.end();
});
client.on("end",function(){
    console.log("Request is ended!!!");
});
```

For getting the output, open the two command prompt windows - One for executing the server and other for executing the client program. Execute the server program first and then client. The following output illustrates this application.

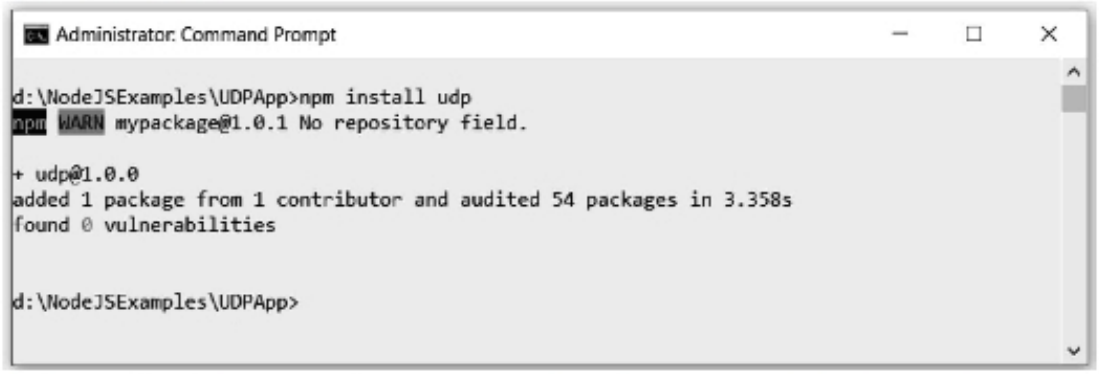
Output



Creating UDP server-client Communication Application

- A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time and content are not guaranteed.
- It is a basic transfer unit associated with a packet-switched network. The applications that communicate via datagrams send and receive completely independent packets of information.
- For using UDP client server communication, the first and foremost thing which is required is installation of UDP module.
- Following screenshot shows how to install **udp** module in the current working directory.

Installing UDP Module



We have two build to node.js programs - One for server and other for client.

UDP Server Program

To build the UDP server program we have to follow the steps as given below -

Step 1 : We use "udp" module by adding following line at the beginning of both the programs -

```
var udp = require("udp");
```

Step 2 : Now we use the **createSocket** method for creating the UDP sockets. For that purpose we have to add following code in the program

```
var server = udp.createSocket("udp4");
var client = udp.createSocket("udp4");
```

we create a UDP socket which will be udp4 for IPV4.

Step 3 : We use the event handler **on** for the server when any client gets connected to it.we can handle three events such as **message ,listening, close** and **error.**

- For **message** event, the data received from the client is displayed on the console using **cosole.log** method. Similarly we can send the data to the server using **send** method.
- For the **listening** event, the port number and ip address of the server on which it is running is displayed.
- For **close** event, the connection closing message is displayed to the client.
- If any error occurs during establishment of connection, the **error** event is fired, and error message is displayed on the console.

Step 4 : During this entire, client server communication Server must bind to one particular **port number**. This can be done using **bind** method. To this method, the port number is passed. The same port number must be used in the client program, so that both the server and client can communicate on the same port number.

The complete node.js code for UDP server is as given below -

server.js

```
var udp = require("udp");

var server = udp.createSocket("udp4");
var client = udp.createSocket("udp4");

server.on("error",function(err){
    console.log("Error!!!!"+err.message);
    server.close();
});
```

```
server.on("message",function(msg,info){
    console.log("Data received from client is: "+msg.toString());
    var msg="Hello friend";
    //sending data to the server
    server.send(msg,info.port,"localhost", function(err){
        if(err)
          client.close();
        else
          console.log("Data sent!!!");
    });

});

server.on("listening",function(){
    var address = server.address();
    var port = address.port;
    var ipaddr = address.address;
    console.log("Server is listening to port: "+port);
    console.log("Server's IP Address: "+ipaddr);
});
server.on("close",function(){
    console.log("Closing the server socket now!!!!");
});
server.bind(8082);
```

To build the UDP client program we have to follow the steps as given below -

Step 1 : We use "udp" module by adding following line at the beginning of the client program.

```
var udp = require("udp");
```

Step 2 : Now we use the **createSocket** method for creating the UDP sockets. For that purpose we have to add following code in the program

```
var client = udp.createSocket("udp4");
```

Step 3 : We use the event handler **on** for the client when any server gets connected to it. For that purpose we the **message** event handler.For **message** event, the data received from the server is displayed on the console using **cosole.log** method.

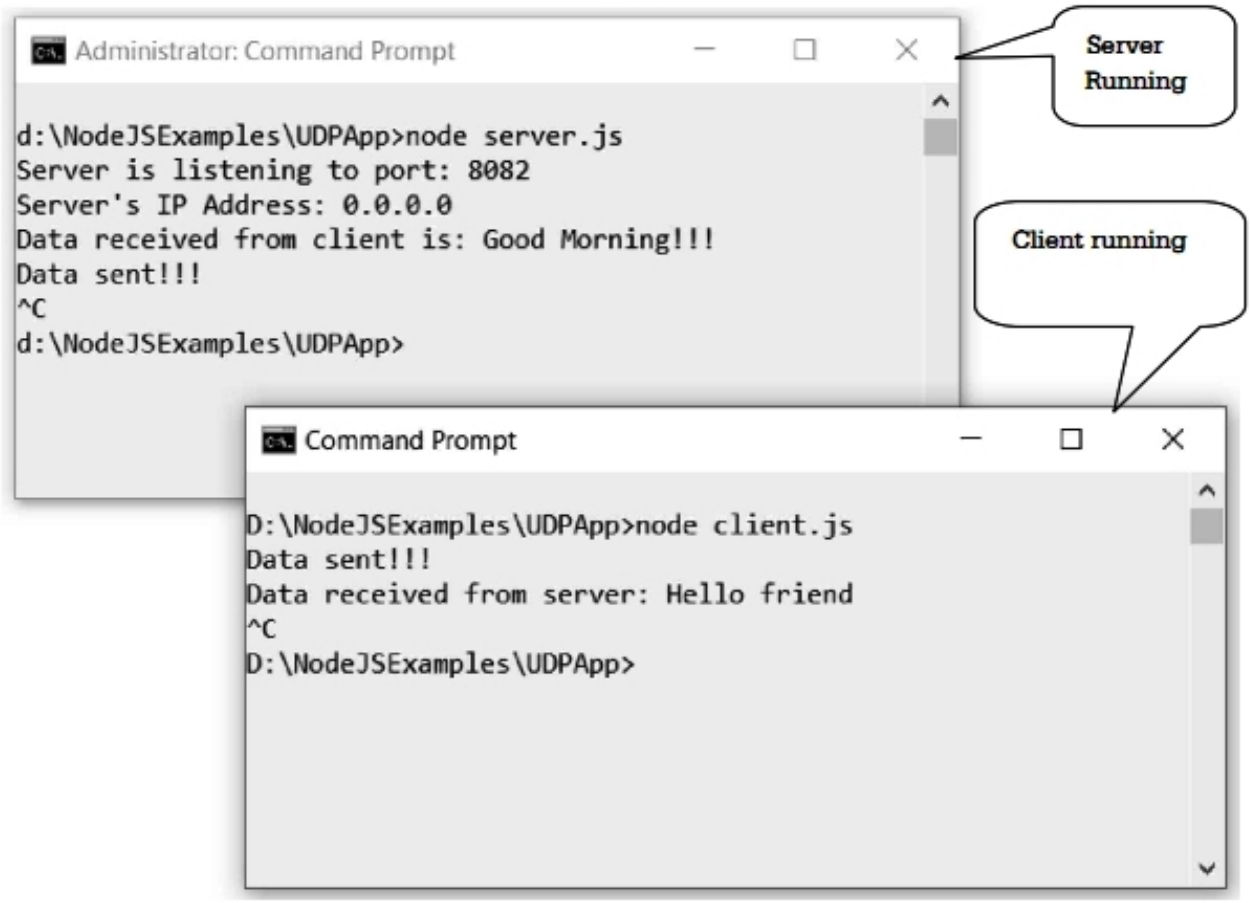
Step 4 : Similarly we can send the data to the server using **client.send** method. Note that during this entire communication the same port number and host name is used which was used for server program.

The complete code for client program using TCP is as given below -

client.js

```
var udp = require("udp");
var client = udp.createSocket("udp4");

var mydata = "Good Morning!!!";

client.on("message", function(msg,info){
    console.log("Data received from server: "+msg.toString());
});
client.send(mydata, 8082,"localhost", function(err){
    if(err)
        client.close()
    else
        console.log("Data sent!!!");
});
```

Output



5.6 Web Module

The Web module is mainly controlled by a web server that handles the requests and provides the responses.

What is Web server ?

- Web server is a special type of server to which the web browser submits the request of web page which is desired by the client.
- There are some popularly used web servers such as **Apache** and **IIS** from Microsoft.

Functions of web server

Various functions of web server are -

1. The web servers **accepts the requests** from the web browsers.
2. The user **request is processed** by the web server.
3. The web servers respond to the users by **providing the services** which they demand for over the web browsers.
4. The web servers **serve the web based applications.**
5. The DNS translate the **domain names into the IP addresses.**
6. The servers **verify** given address exists, **find necessary files** ,run appropriate scripts exchange cookies if necessary and returns back to the browser.
7. Some servers actively participate in **session handling techniques.**

Working Principle of Web Server

- When user submits a request for a web page, he/she is actually demanding for a page present on the web server.
- When web browser submits the request for a web page, the web server responds this request by sending back the requested page to the web browser of the client's machine.

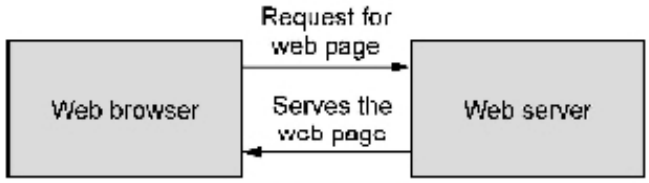


Fig. 5.6.1

Step 1 : Web client requests for the desired web page by providing the IP address of the website

Step 2 : The web server locates the desired web page on the website and responds by sending back the requested page. If the page doesn't exist, it will send back the appropriate error page.

Step 3 : The web browser receives the page and renders it as required.

Example Code

Node.js has a built in module named **http** which allows us to transfer data over the Hyper Text Transfer Protocol(HTTP)

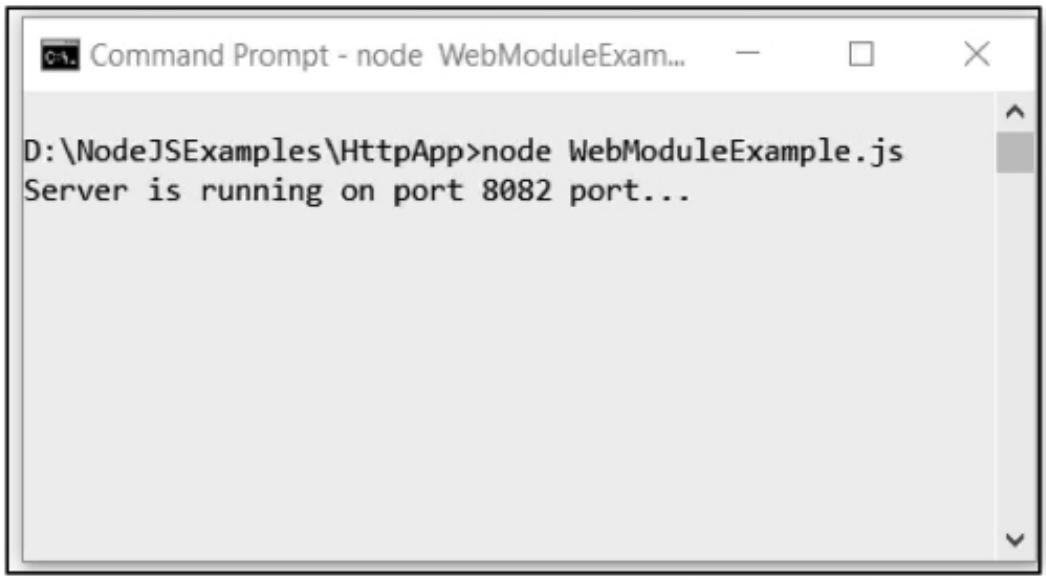
In this section we will build a web server using **http** module.

WebModuleExample.js

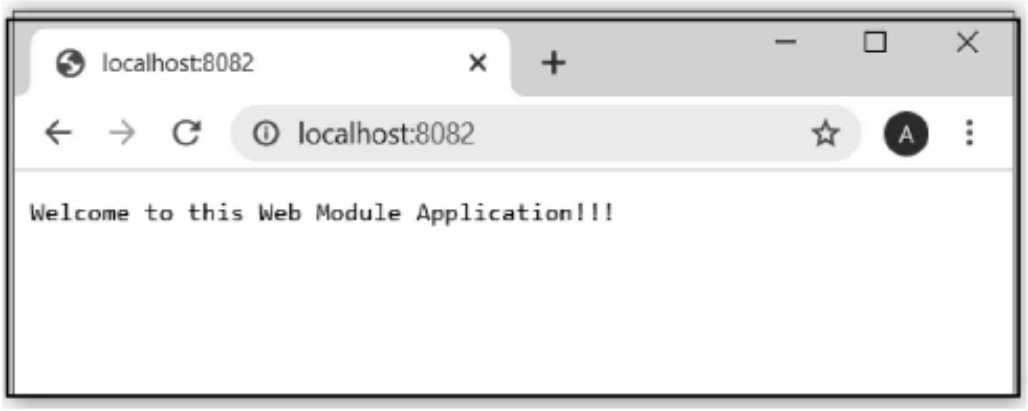
```
var http = require("http");
var server = http.createServer(function(request,response) {
    response.writeHead(200,{"Content-Type": "text/plain"});
    response.end("Welcome to this Web Module Application!!!");
});
server.listen(8082);
console.log("Server is running on port 8082 port....");
```

Output

Step 1 : First of all open command prompt and type following command



Step 2 : Now open some suitable web browser such as Mozilla Firefox or Google Chrome and type the **locahost:8082** as URL. It is illustrated by following screenshot



Script Explanation : In above given Node.js script,

1) We have imported **http** module.
2) The **http.createServer()** method includes **request** and **response** parameters which is supplied by Node.js.
3) To send a response, first it sets the response header using **writeHead()** method and then writes a string as a response body using **write()** method.
4) Finally, Node.js web server sends the response using **end()** method. But we can directly write the response body using **end** method.
5) Using **server.listen** method we can specify the port number on which our web server is running.
6) Just refer the illustrative output given above.

Handling various Http Requests

The **Http.request** object is used to get information about the current Http request. The response object is used to send the response for current HTTP request.

Following Node.js script shows how different HTTP requests are made and how to respond them using response object of HTTP module.

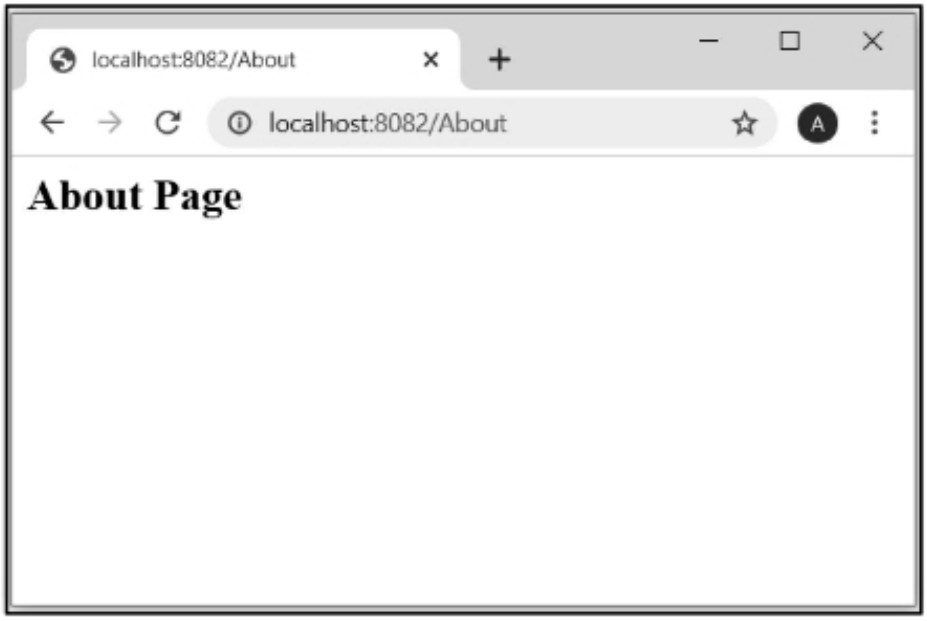
WebModuleExample.js

```
var http = require("http");
var server = http.createServer(function(request,response) {
    if(request.url == "/") {
        response.writeHead(200,{"Content-Type": "text/html"});
```

```
        response.write("<html><body><h1>Welcome to this Web Module
        Application!!!</h1></body></html>");
        response.end();
    }
    else if(request.url == "/About"){
        response.writeHead(200,{"Content-Type": "text/html"});
        response.write("<html><body><h2>About Page</h2></body></html>");
        response.end();
    }
    else if(request.url == "/Contact"){
        response.writeHead(200,{"Content-Type": "text/html"});
        response.write("<html><body><h2>Contact Information</h2></body></html>");
        response.end();
    }
    else
        response.end("This is invalid request");

});
server.listen(8082);
console.log("Server is running on port 8082 port...");
```

Output

We can open a web browser and get the about page by make the request using the URL as "localhost:8082/About". The illustrative output is as given below -



5.7 Debugging

Node.js provides a built in debugging tool. This tool is command driven and non-graphic tool.

Following table shows various commands used in debugging -

Command	Description
next(n)	It moves on to next line for debugging.
cont(c)	Continues the execution and stops at 'debugger' statement if any.
step	Steps in the function
out	Steps out of the function
watch	Add the expression or a variable for inspecting its value.
watchers	This allows us to see the value of the expression or variable that are passed to **watch** command.
.exit	It exits the debugging mode and returns to the command prompt.

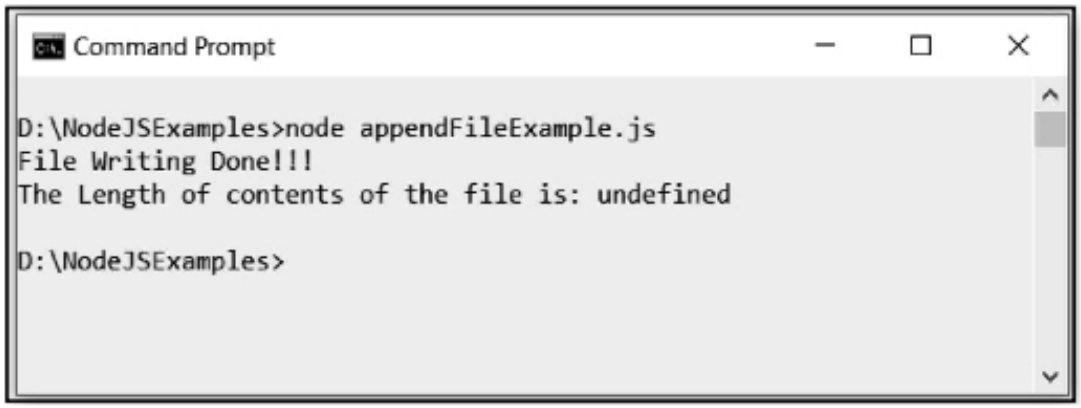
Now let us see how to debug a sample Node.js program with the help of following example -

appendFileExample.js

```
var fs = require("fs");
str = "This line is replacing the previous contents";
var strlen = str.lengthh;
fs.writeFile('myfile.txt', str,function(err){
    if(err)
        console.log(err)
    console.log('File Writing Done!!!');
    console.log("The Length of contents of the file is: "+strlen);
});
```

Purposely made this spelling mistake(**length**), to get to know about this error during debugging process.

Output



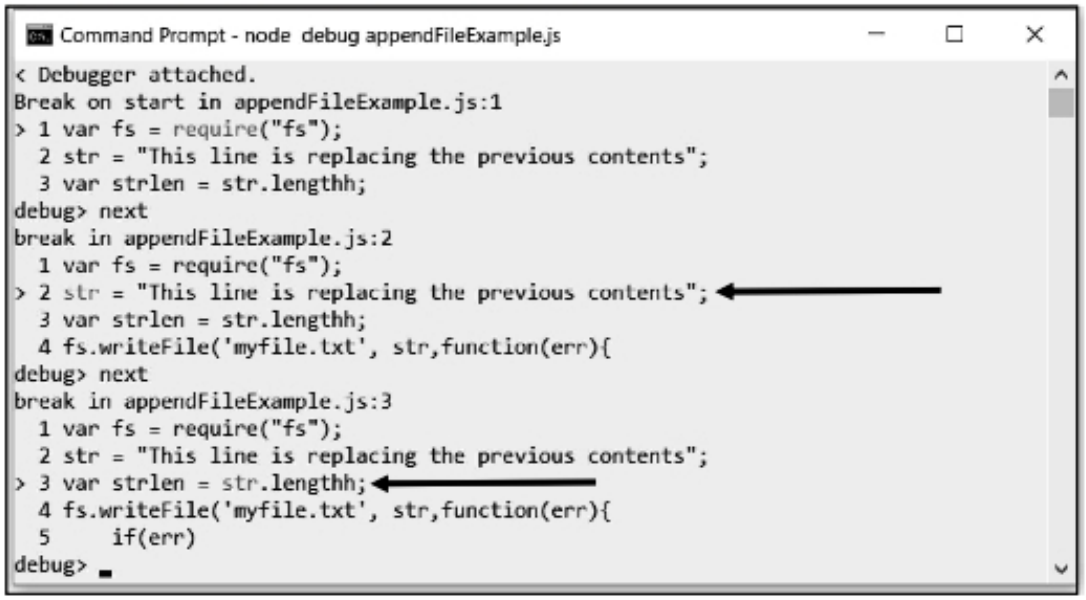
Note that the length of the contents is undefined because of the wrong spelling. But this program does not raise any syntactical error. It simply returns 'undefined'. So what went wrong? Yes, in order to find it out we need to debug our code. To debug the above application, run the following command

Prompt:\>node debug appendFileExample.js

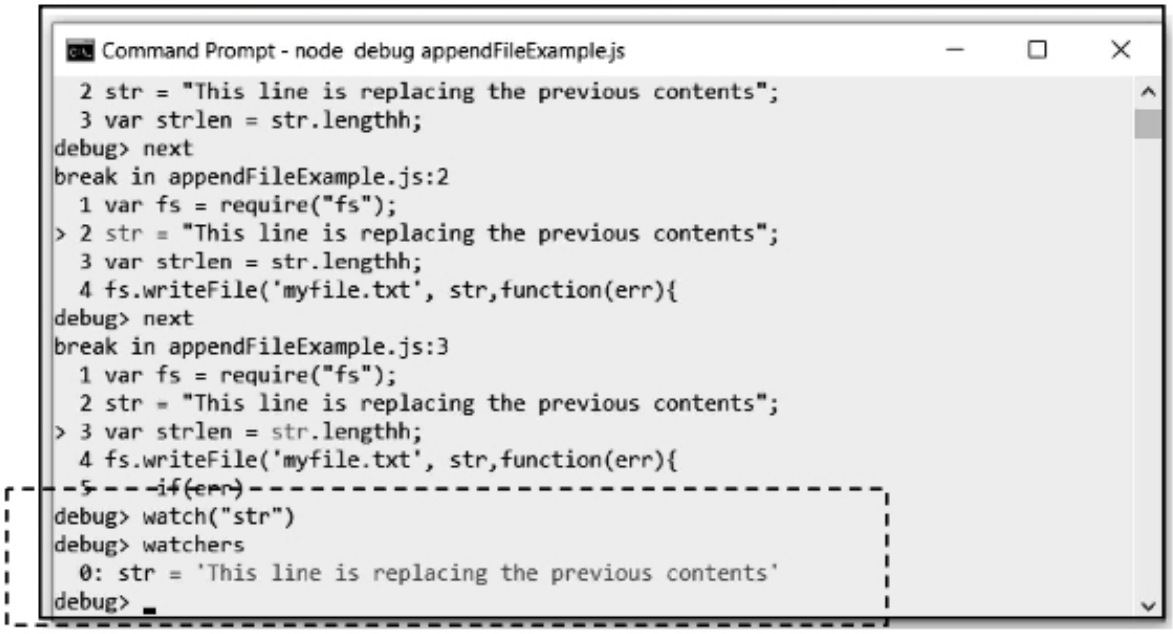
By above command, the debugger starts and waits at some line after executing few lines -

As from above screen shot, we can notice > symbol which indicates the current debugging statement.

Now we can use **next** to move on the next statement

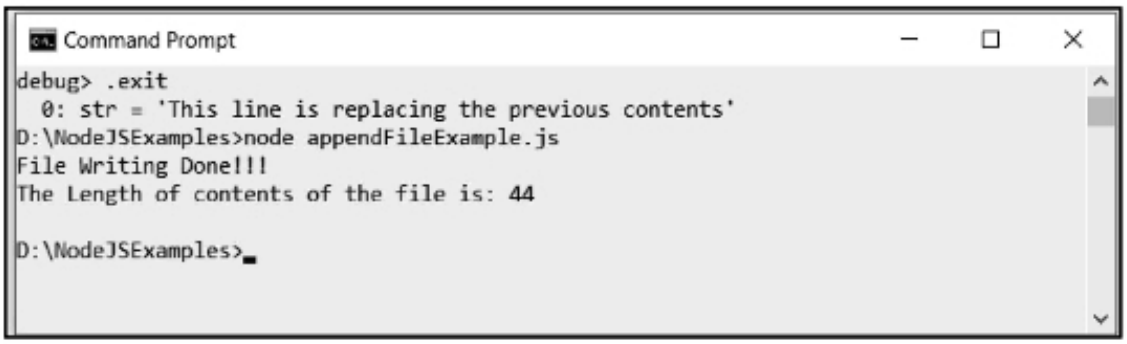


We can inspect the values of the variable using **watch** and **watchers** command.



Then continue to debugging by using next command. Then again by using **watch** command we can inspect the value of **strlen** variable

And here we get to know, that something went wrong in finding the string length. And thus the spelling mistake of **length** attribute of **str** variable can be corrected. Then save your node.js file with the appropriate corrections, stop the debugger and re-run the above application program. The illustrative output is



5.8 Express in NodeJS

Express is web application framework used in node.js. It has rich set of features that help in building flexible web and mobile applications.

Features of Express

1) **Middleware:** Middleware is a part of program,that accesses to the database and respond to client requests.
2) **Routing:** Express JS provides a routing mechanism so that it is possible to reach to different web pages using the URLs.
3) **Faster Server Side Development:** It is very convenient to develop server side development using Express JS
4) **Debugging:** ExpressJs makes debugging easier by providing the debugging mechanism.

Installing Express

Express is installed using Node Package Manager(npm). The command issued for installing the express is,

Prompt:>npm install express

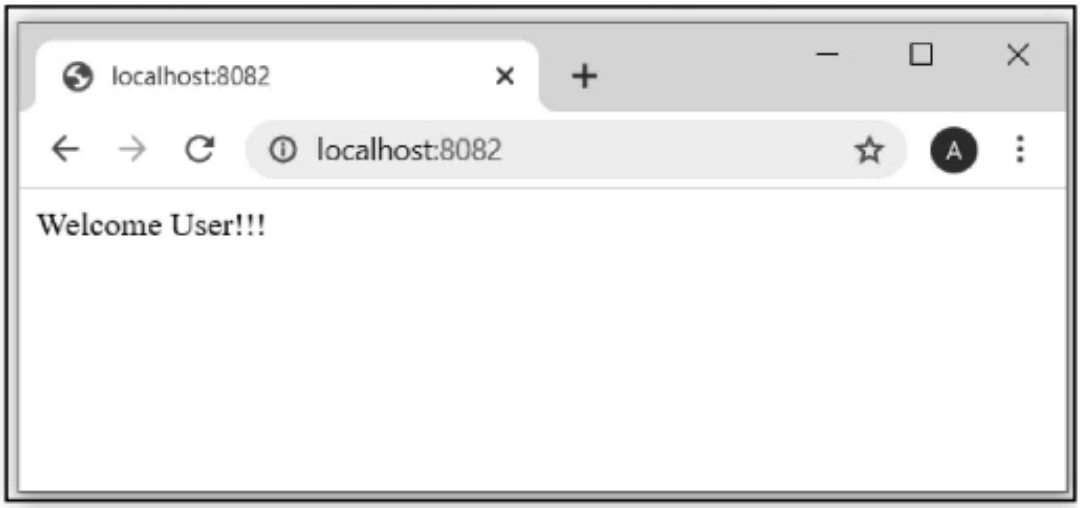
Following is a simple application program that uses Express JS for displaying "welcome message" on the web page.

app.js

```
var express = require('express');
var app = express();
app.get('/', function(req,res) {
    res.send("Welcome User!!!");
});

var server = app.listen(8082, function(){
    console.log("Server started!");
});
```

Output

Open some web browser and issue the URL "localhost:8082"



Routing

Routing is a manner by which an application responds to a client's request based on particular endpoint. The sample endpoints are,

localhost:8082/aboutus

localhost:8082/contact

localhost:8082/home

There are various types of requests such as **GET, POST, PUT or DELETE**. While handling these requests in your express application, the request and response parameters are passed. These parameters can be used along with the **send** method.

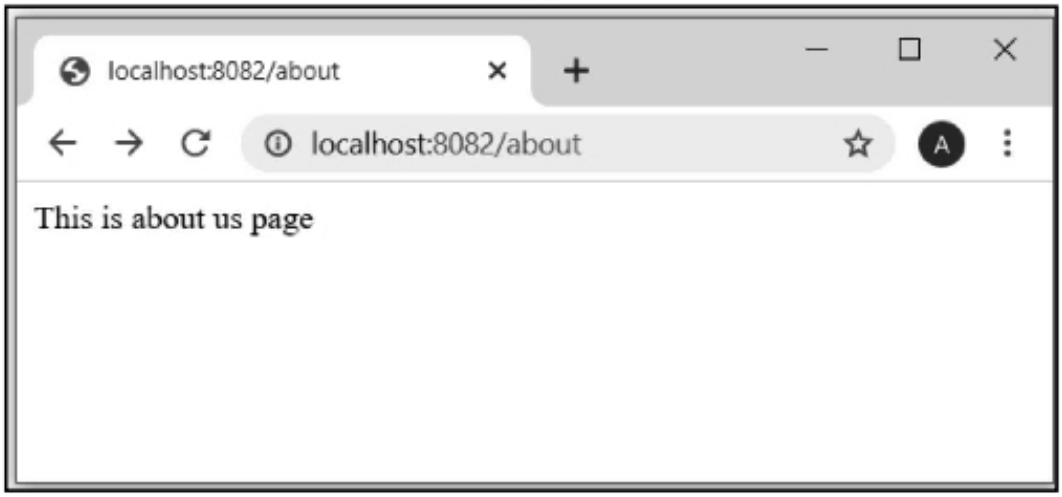
Following application represents how to performing routing using express

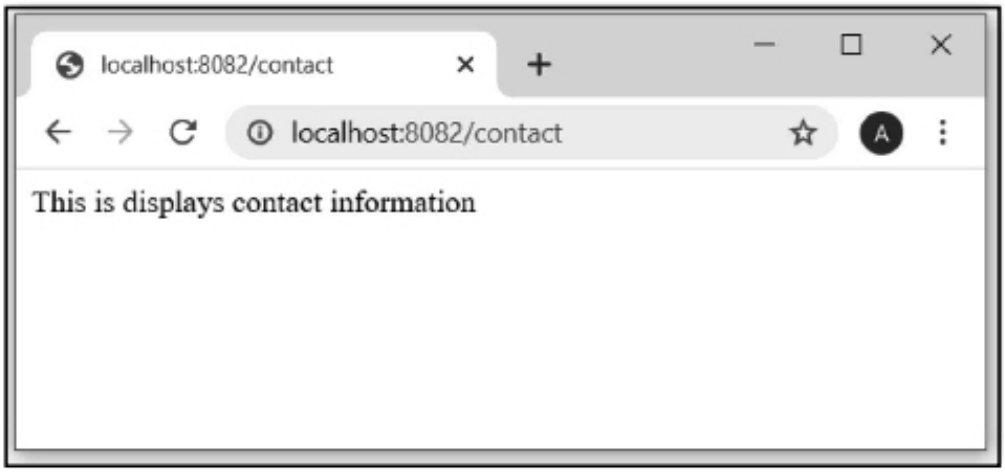
app.js

```
var express = require('express');
var app = express();

app.get('/', function(req,res) {
    res.send("Welcome User!!!");
});
//routing to 'about us' page
app.get('/about', function(req,res) {
    res.send("This is about us page");
});
//routing to 'contact' page
app.get('/contact', function(req,res) {
    res.send("This is displays contact information");
});
```

```
var server = app.listen(8082, function(){
    console.log("Server started!");
});
```

Output





5.9 Node JS REST API

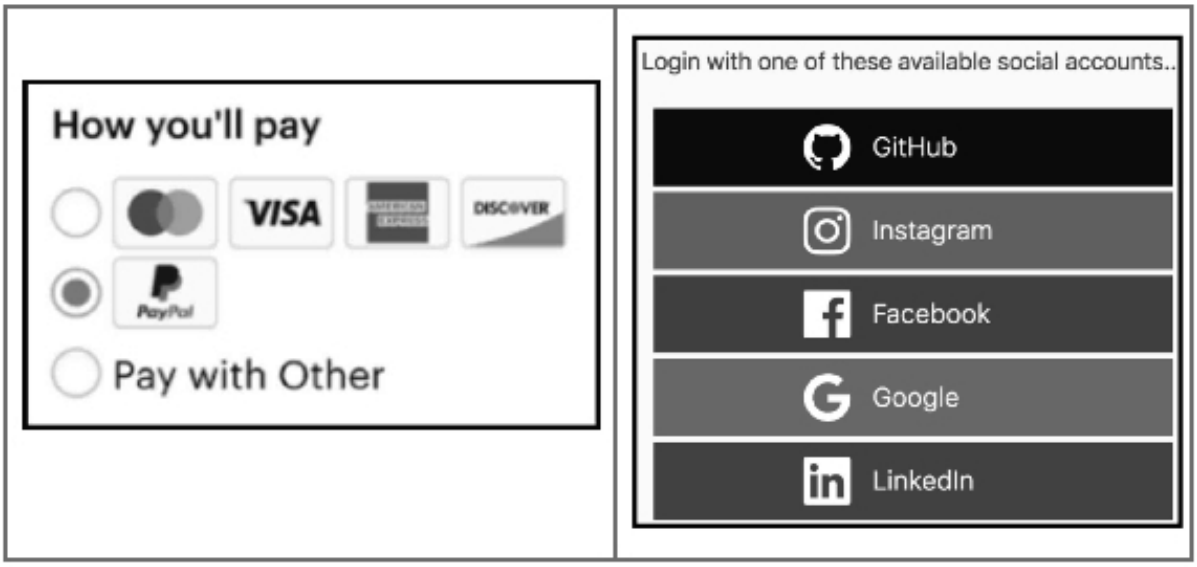
What is API ?

API stands for **Application Programming Interface**. APIs are a set of functions and procedures that allow for the creation of applications that access data and features of other applications, services, or operating systems.

In other words, if you want to interact with a computer or system to retrieve information or perform a function, an API helps you communicate what you want to that system so it can understand and fulfil the request.

It is basically a software intermediary that allows two applications to talk to each other. For example - When we use facebook, send a message or check the weather on our mobile phone, we use API.

For example - If we want to buy a movie ticket, we go to the online ticket booking site, enter movie name, use credit card information, get the ticket, print it. The software that interacts with us to perform all these task is nothing but the API. Following screenshots some well known examples of API.



What is REST API ?

REST stands for Representational State Transfer. It is a set of rules that developers follow while creating their API.

Each URL made by the client is a **request** and data sent back to the client is treated as **response**.

The request consists of,

1. End Point	2. Method	3. Headers	4. Data

(1) Endpoint

The endpoint (or route) is the URL you request. For example -

http://localhost:8082/

(2) Methods

There are various types of methods. But the most commonly used one are as follows -

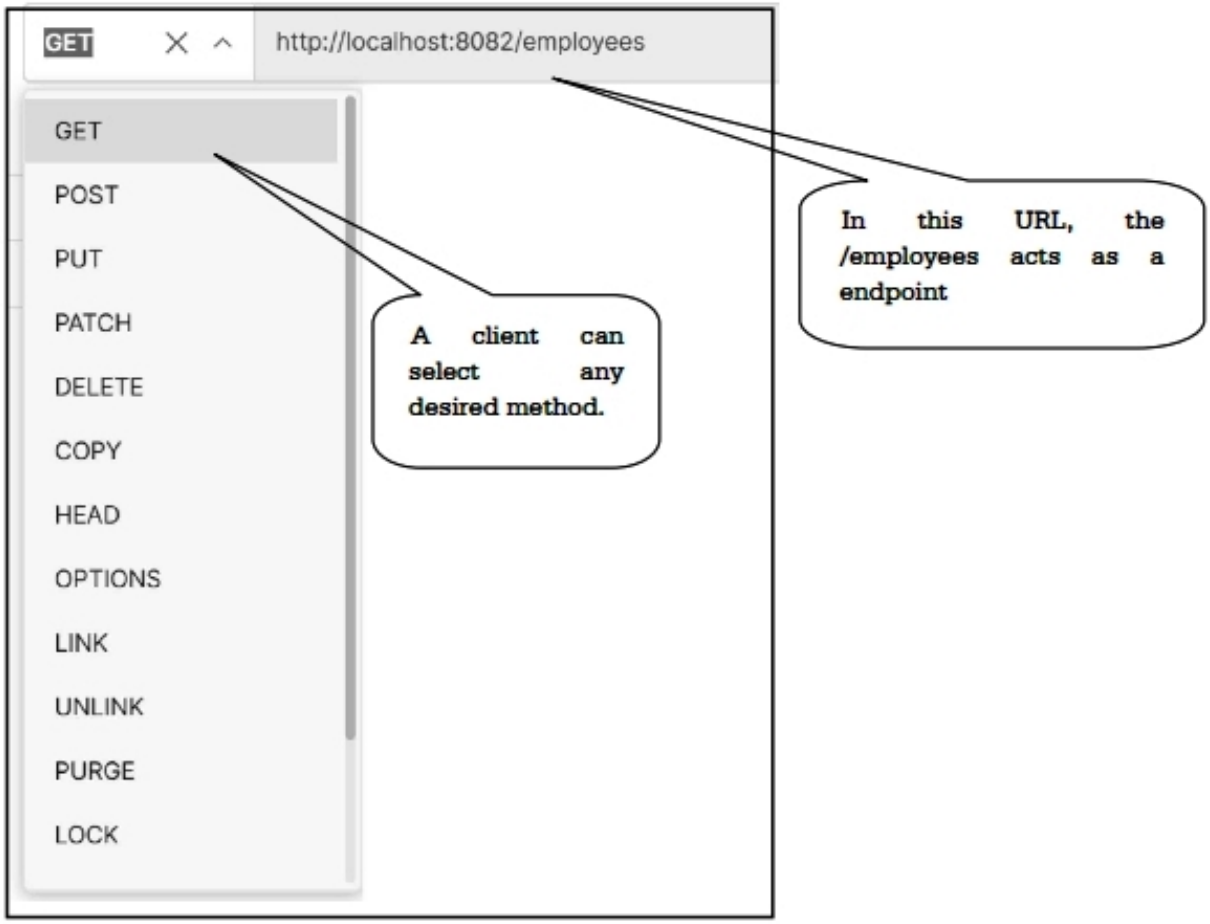
1) GET 2) POST 3) DELETE

4) PUT 5) PATCH

These methods are basically used for performing CRUD operations. Here C- stands for Create, R- stands for Read, U- stands for Update and D-stands for Delete.

Method	Purpose
GET	For getting the data from the server, this request is used. The server looks for the data the client has requested and sends it back to the client.
POST	This method is used when new resource is to be created on the server. When client uses this method and provides the data, then the server creates a new entry in the database and acknowledges the client whether the creation is successful or not.
PUT and PATCH	These are the two methods that are used when the client wants to update some resource. When PUT method is used, then that means client wants to update entire resource. And when PATCH method is used, then that means client simply wants to update small part of the resource.
DELETE	This request is used to delete a resource from the server.

There are some software tools such as Postman, that help the client to select the appropriate method and issue the URL. Following screenshot is of a software tool **Postman** , using which we can select appropriate method.



(3) Headers

Header are used to provide authentication and other useful information to client and server. The HTTP headers are property-value pairs separated by Colons(:).

For example -

"Content-Type: application/json"

(4) Data

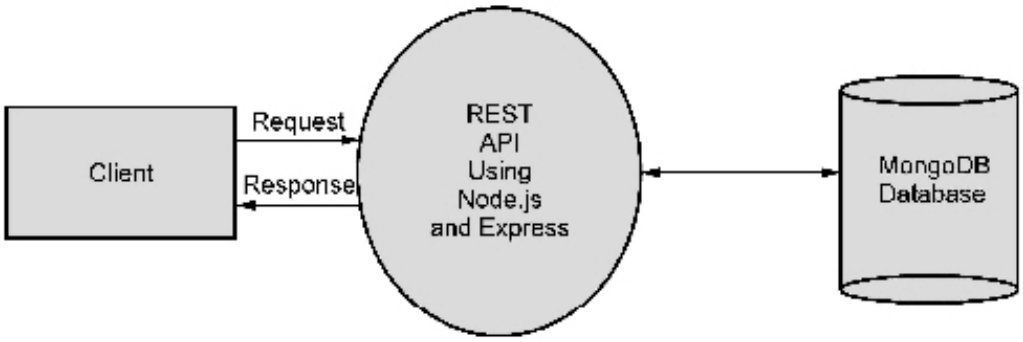
The DATA contains information which the client wants to send to the server. While sending the DATA to the server the methods such as POST, PUT, PATCH or DELETE are used.

It is preferred to send the data in JSON format. The format is,

```
{
    Property1:value1,
    Property2:value2
}
```

Concept of Node JS API

The Node.js script is popularly used to create REST API. We will write the server script in **Node.js.** The database is created using **MongoDB**. The client interface is handled with the help of **Postman.**



Let us understand how to create Node JS REST API with the help of following example

Example Code

Prerequisite: For creating the Node.js Rest API we need following software to be installed in our machine

1) **Postman :** This is an API client. This client software allows us to enter the request in JSON format and select the commands such as GET, POST, UPDATE, DELETE and so on.
2) **MongoDB:** This is a database package. We need MongoDB compass to be installed along with the mongoDB server application. The MongoDB compass is a graphical tool for handling database operations.
3) **NodeJS :** This is of course used for creating an API in .js file.

Step 1 : Make a new folder in your current working directory for creating Rest API. Here I am creating a folder named RestApiDemo

Step 2 : Now open the command prompt window and create package.json file using **npm init** command. Following screenshot illustrates the same.

At the command prompt simply type the command

```
npm init
```

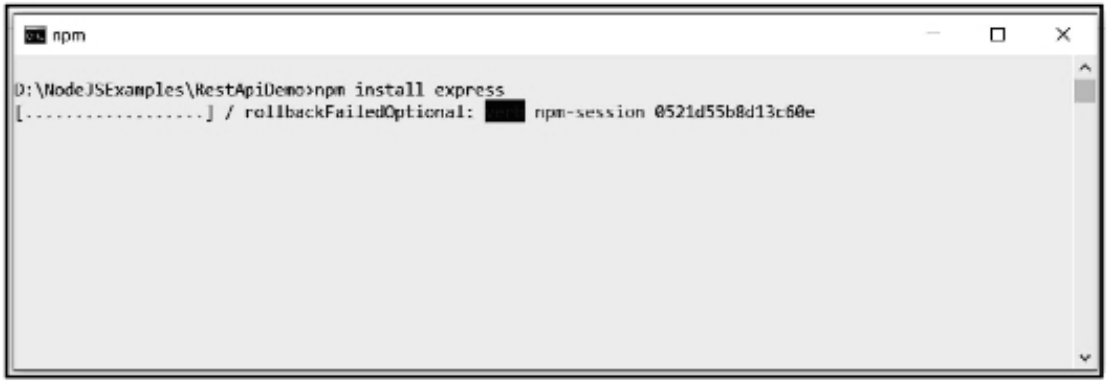
Then hit the enter button. You can add the values of your choice or simply go on pressing enter key and accept the default value.

Step 3 : Now install some more packages that are required for creating this Rest API

Installation of express module

Issue the following command

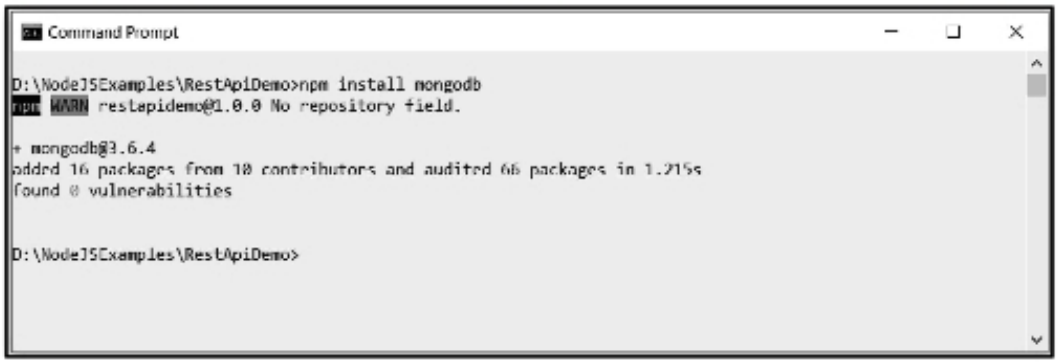
prompt:\>npm install express



Installation of mongodb module

Issue the following command

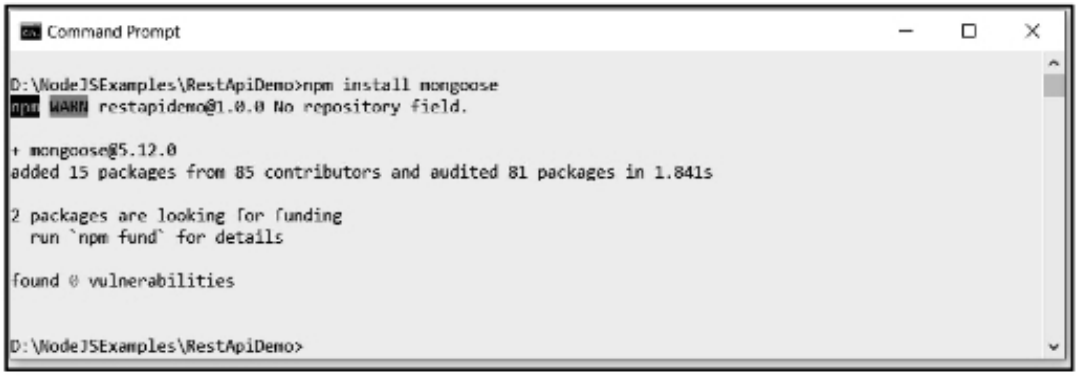
prompt:\>npm install mongodb



Installation of mongoose module

Issue the following command

prompt:\>npm install mongoose



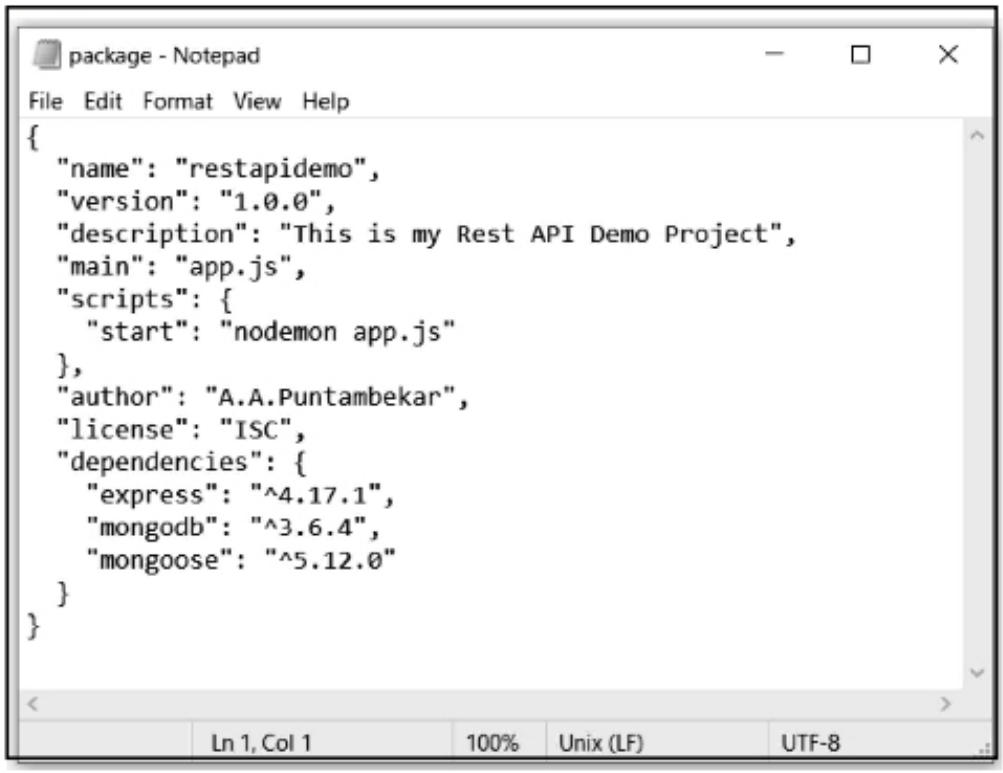
Installation of nodemon

Issue the following command at the command prompt

prompt:>npm install -g nodemon –save-dev

The sample **package.json** file will show you all these dependencies of the modules you have installed just now

package.json



Note that I have added following line in **package.json** file so that as soon as I save any changes in the file, the output gets reflected immediately. I need not have to run the **app.js** file every time.

"start" : "nodemon app.js"

Step 4 : Now create your main server in a file named **app.js**.

app.js

```
const express = require('express');
const mongoose = require('mongoose');

url = 'mongodb://localhost/EmployeeDB';

const app = express();
mongoose.connect(url, {useNewUrlParser:true})

const con =  mongoose.connection   //getting the connection object

con.on('open',() => { //on opening the connection, connecting with database
    console.log('Connected to Database')
})
```

URL for the database name

Connecting to MongoDB database using Mongoose package

```
app.use(express.json())
const employeeRouter = require('./routes/employees') //initial endpoint
app.use('/employees',employeeRouter)
app.listen(8082, () =>{
    console.log("Server Started!!!")
} )
```

Step 5 : Create a folder named **routes.** Inside the **routes** folder create a file named **employees.js**. This file will handle the GET and POST requests of REST API. Using POST request we can create the API by inserting the data into the database. The GET request will retrieve and display the data from the database. Thus routing of GET and POST requests is done in this file.

employees.js

```
const express = require('express');
const router = express.Router();
const Employee = require('../models/employee');

router.get('/', async(req,res) =>{
    try {
        const employees = await Employee.find()
        res.json(employees)
    }
    catch(err) {
        res.send('Error is: '+err)
    }
})

router.post('/', async(req,res) => {
    const employee = new Employee({
        name: req.body.name,
        designation: req.body.designation
    })
    try {
        const e1 = await employee.save();
        res.json(e1);
    }
    catch(err){
        res.send('Error is: '+err)
    }
})
module.exports = router
```

This is a router program that routes on receiving the particular type of request

Handling GET command issued by the client, and using find(), displaying data present in the Employee database

Handling the POST request. The data is received here from client. Hence we use **req.body** followed by name of the data field

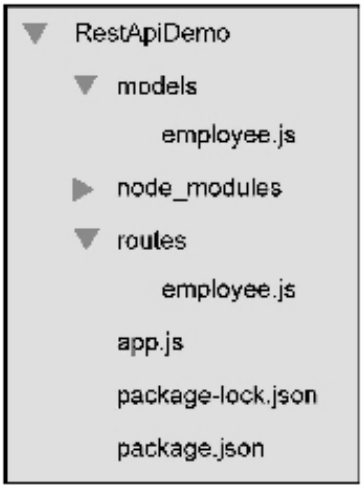
Step 6 : Now create another folder named **models**. Inside **models** folder create a file named **employee.js**. This file will describe the Schema. The fields of the document are described in this file along with data type and some other description is mentioned in this file. Note that the schema is specified in JSON format.

employee.js

```
const mongoose = require('mongoose')
const employeeSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    designation: {
        type: String,
        required: true
    }
})
module.exports = mongoose.model('Employee',employeeSchema)
```

Preparing the schema for the database

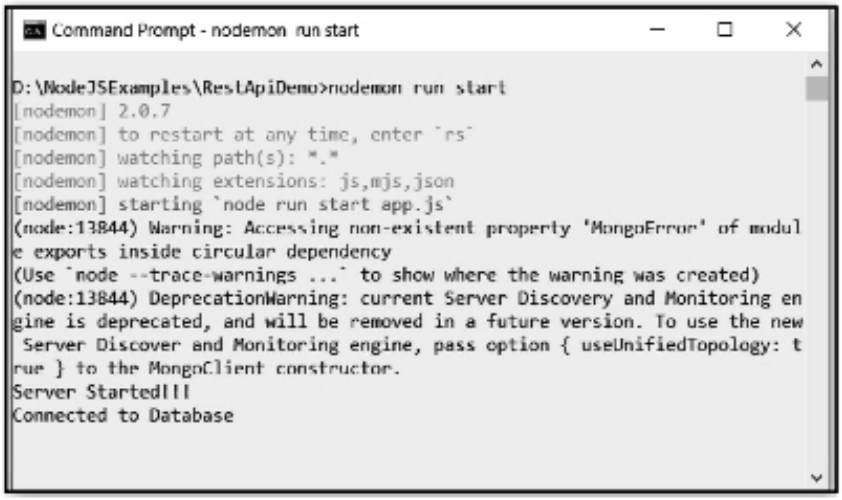
Step 7 : For better understanding, the folder structure of this project can be viewed in the following explorer window



Step 8 : For getting the output, Open the command prompt window and issue the following command.

```
D:\NodeJSExamples\RestApiDemo>nodemon run start
```

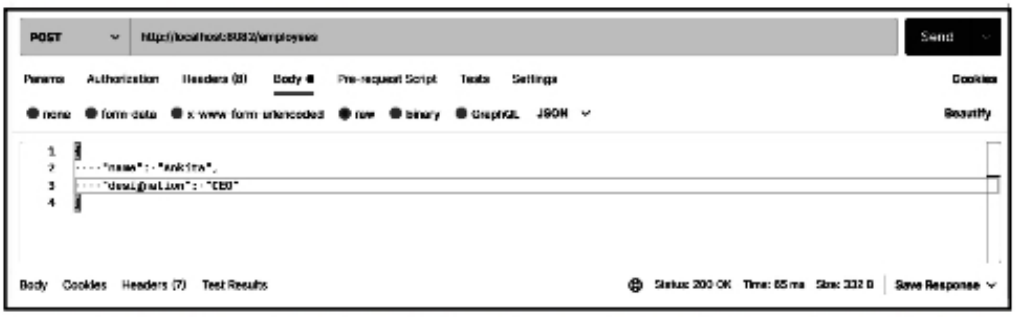
You should get the "Server started" and "Connected to Database" messages



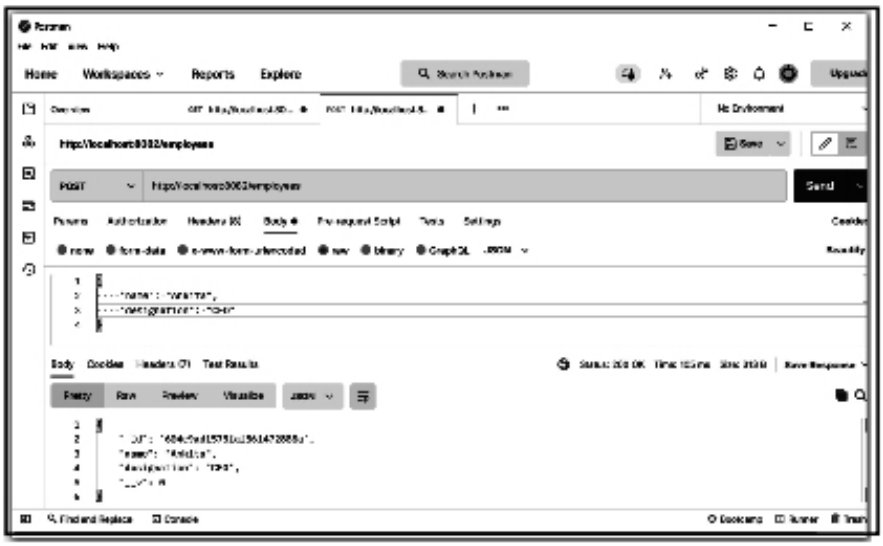
Now, Open the Postman and click on **Create Request**. Select the **POST** command issue the URL as

http://localhost:8082/employees

And then click on **Send** button



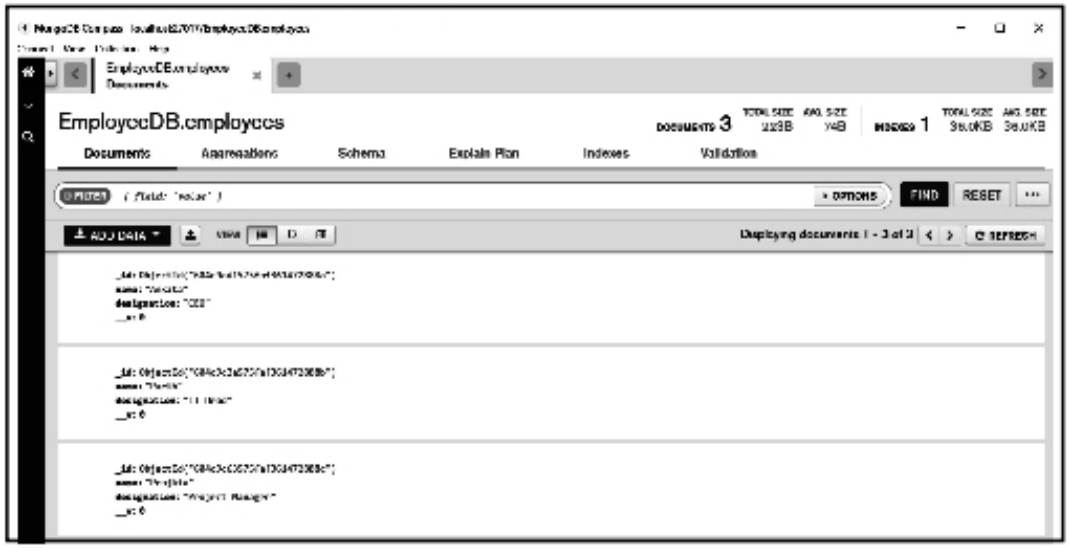
Once we hit the send button we get the result by generating the unique **_id** for the data which we have inserted. It can be viewed as follows,



In this , manner we can insert more data by selecting the Post command. This data is sent to the **app.js** server and then getting stored in the MongoDB database.

In order to retrieve data being stored , we can use **GET** method on Postman client and see the complete data in the form of JSON structure.

We can verify the data by opening the database created in MongoDB



In this manner we have created our REST API for employee details.

5.10 Sessions and Cookies

5.10.1 Sessions

Http is a **stateless** protocol. That means that when we load a page in our browser, and then navigate to another page of same website the server has no means of knowing that these requests have originated from same browser or client.

Browser communicating with the server and server responding to the client via browser is called as session.

Sessions is a mechanism which allows to store data for individuals on sever side.

Following is an example of session handling mechanism. In this example we use the **cookie-parser** package. It is basically a middleware which parses the cookies attached to the client request object. The **express-session** package is useful in creating the middleware session.

Example Code

Step 1 : Create a folder in which you can store your source code for demonstrating sessions. I have named this folder as **SessionAppDemo**.

Now create a **package.json** file using following command

```
D:\NodeJSExample\SessionAppDemo>npm init
```

Then install all the required packages such as **express, cookie-parser** and **express-session**. Use following commands

```
D:\NodeJSExample\SessionAppDemo>npm install express
D:\NodeJSExample\SessionAppDemo>npm install --save cookie-parser
D:\NodeJSExample\SessionAppDemo>npm install --save express-session
```

Step 2 : Now we will create a source file using **.js** extension in which the session is created. The source code is as follows -

sessionDemo.js

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.use(cookieParser());
app.use(session({secret: " A secret Key"}));

app.get('/', function(req, res){
  if(req.session.viewCount){
    req.session.viewCount++;
    res.send("You visited this page " + req.session.viewCount + " times");
  } else {
    req.session.viewCount = 1;
    res.send("Welcome to this page for the first time!");
```

```
  }
});
app.listen(8082);
```

Step 3 : For getting the output, first open the command prompt and Issue the command

D:\NodeJSExamples\SessionAppDemo>node sessionDemo.js

Now open the suitable web browser and enter the URL

If you just go on refreshing the above page for several times, the output will get changed. For instance it could be something like this -

Script Explanation :

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
```

- In this script we use the above three modules

```
var app = express();
```

- Create instance of express in variable **app**

```
app.use(cookieParser());
app.use(session({secret: " A secret Key"}));
```

- The **app.use** is a method used to configure the middleware such as **cookie-parser** or **session**.
- The middleware functions determine the flow of request-response cycle. They are executed after every incoming request.These functions are functions that have access to the **request object** (req), the **response object** (res).
- The **cookie-parser** and session are called **third-party middlewares**.
- **cookie-parser** is a middleware which parses cookies attached to the client request object. Using session middleware we can perform various tasks such as creating the session, setting the session cookie and creating the session object in request object(reg).
- **Secret :** Here, the secret is the hash key passed to the header file so that this information can be securely decoded at the server end only. This is important because without a session secret, any third party app can easily look into the cookie with the help of browser and hijack it. Having a secret ensures that when authenticating, the cookie data is encoded using the secret key only known to the client and the server; this data can then not be easily decoded by a third listener.

```
app.get('/', function(req, res){
  if(req.session.viewCount){
    req.session.viewCount++;
    res.send("You visited this page " + req.session.viewCount + " times");
  } else {
    req.session.viewCount = 1;
    res.send("Welcome to this page for the first time!");
  }
});
app.listen(8082);
```

- When user visits the web page a new session is created for the user and a cookie is assigned to it. So when next time when user visits the same page the cookie is checked and page view counter updated accordingly.

5.10.2 Cookie

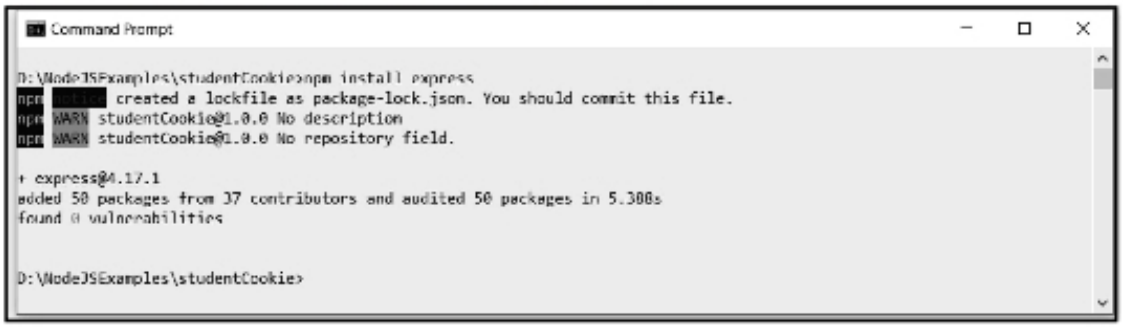
Cookie is a small piece of information used to store name-value pair. This cookie data is sent to the client with server request and stored at the client's machine. Every time when user loads the web site, this cookie is sent with the request. This helps in keeping track of the user's actions.

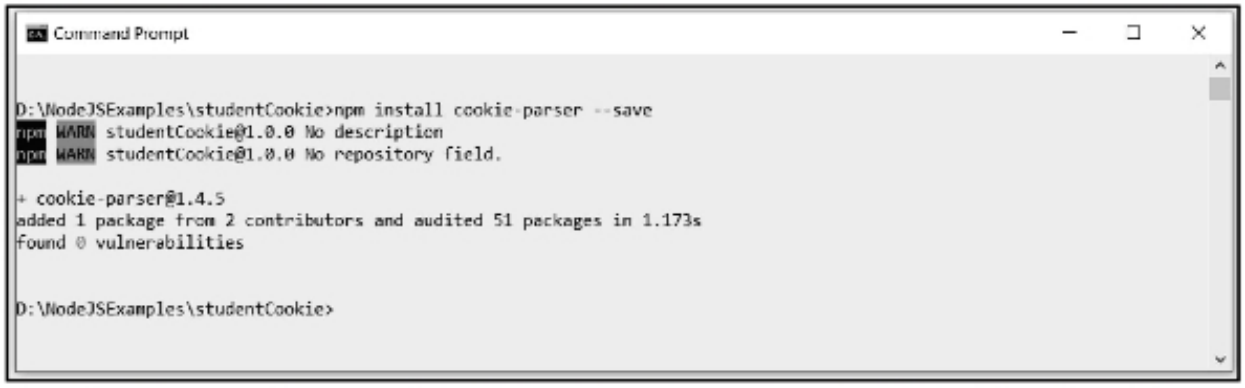
Following example illustrates how to create cookie, display cookie data and clear the cookie.

Step 1 : Create a folder by some suitable name in a current working directory. I have created a directory named **studentCookie**

Step 2 : Open the command prompt window and issue the **npm init** command for the created folder. Refer the following screenshot -

Step 2 : Install the packages express and cookie-parser





Step 3 : Now we will write a js file for setting the cookie and reading the cookie. The code is as follows -

app.js

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();
app.use(cookieParser());


app.get('/',(req,res) => {
    res.send("Cookie Demo");
});

var student = {
    rollno: "101",
    name: "Parth"
};
 //Cookie is added
app.get('/setstudent',(req,res) => {
    res.cookie("studentData",student)
    res.send("Student data added to Cookie");
});

//Display Cookie
app.get('/getstudent', (req,res) =>{
    res.send(req.cookies);

});

//server listening
app.listen(8082, ()=> {
    console.log("Server is listening at port 8082");
})
```

Step 4 : Now open the command prompt and execute the js file as follows -

Now open a web browser and type the url for setting and getting the cookie . Here is the demonstration

Destroy Cookie

For deleting the cookie we use

```
clearCookie(cookie Name)
```

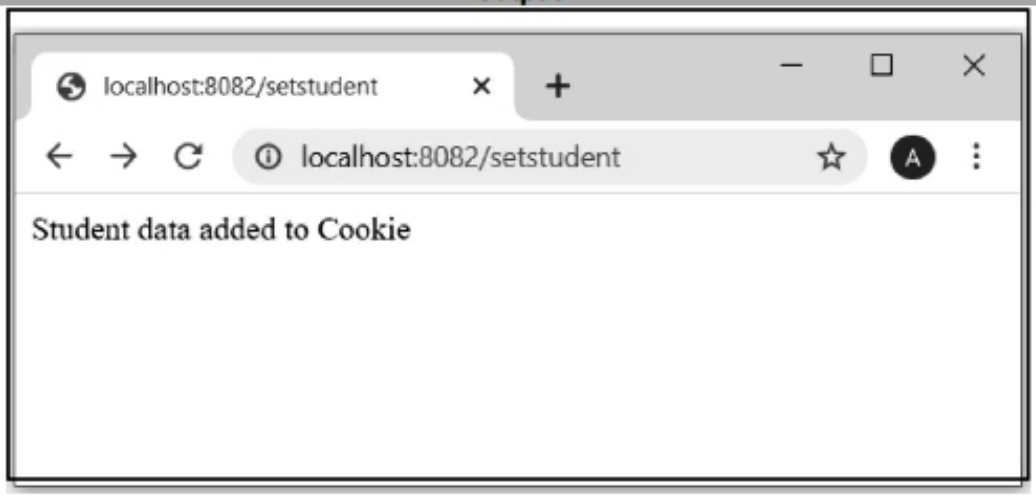
Following code shows how to delete a cookie

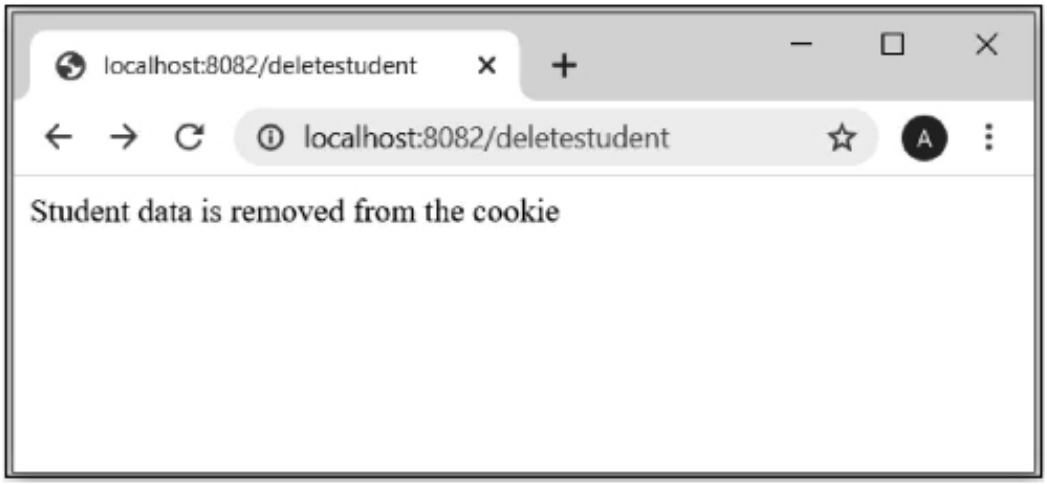
app.js

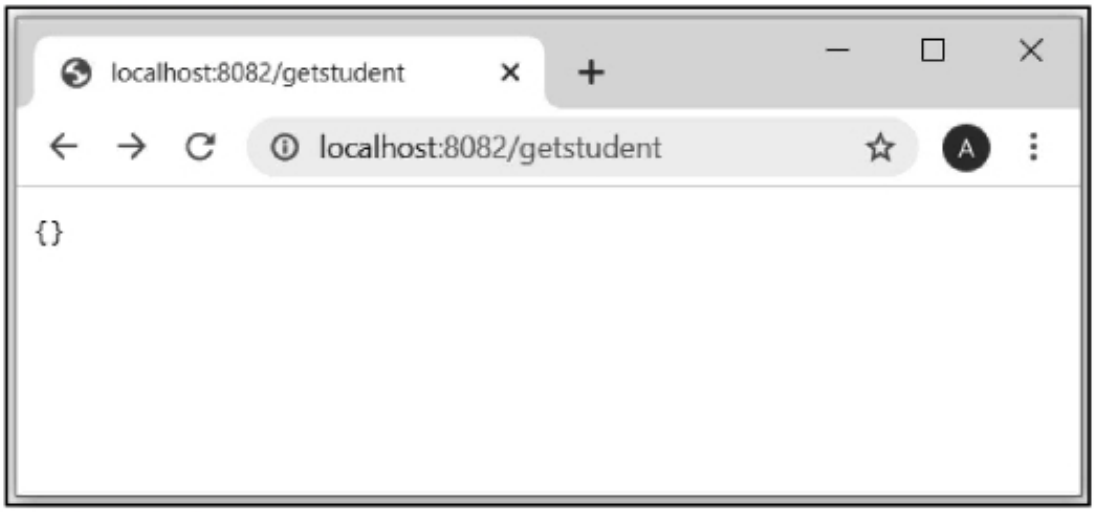
```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();
app.use(cookieParser());


app.get('/',(req,res) => {
    res.send("Cookie Demo");
});

var student = {
    rollno: "101",
    name: "Parth"
};
 //Cookie is added
app.get('/setstudent',(req,res) => {
    res.cookie("studentData",student)
    res.send("Student data added to Cookie");
});

//Display Cookie
app.get('/getstudent', (req,res) =>{
    res.send(req.cookies);
```

```
});

//Delete Cookie
app.get('/deletestudent', (req,res) =>{
    res.clearCookie("studentData");
    res.send("Student data is removed from the cookie");
});
//server listening
app.listen(8082, ()=> {
    console.log("Server is listening at port 8082");
})
```

Output







5.11 Design Patterns

Definition of design pattern : Design pattern is general reusable solution to commonly occurring problem within a given context in software design.

In general the pattern has **four essential** elements -

Pattern name : The pattern name is used as handle to describe the design pattern. The naming of the pattern increases the design vocabulary. Due to the name of the pattern it becomes easy to think about the design and to communicate the design with the others. Finding the appropriate name for the design pattern is one of the hardest and tricky part.

Problem : The problem describes when to apply the pattern. It explains the problem and its context. It might describe various things such as algorithm for the problem, the class or the object structure of the problem, or it can list down the conditions that must be followed before applying the pattern.

Solution : The solution describes the elements of the design, their relationships, responsibilities and collaborations. The solution never describes the concrete design or the implementation because the pattern is like templates and it can be applied to various problems in varied situations.

Consequence : The consequences describe the results and trade-offs of applying pattern. The consequences for software often concerned with space and time trade-offs. The listing of consequences helps in evaluation of the design pattern.

Let us discuss some of the popular design patterns and their implementations with the help of Node.js

(1) Singleton

The singleton is used when we need only **single instance of a class.** That means we can not create multiple instances. If there is no instance, a new one is created. But if there is an existing instance then it will use that one.

Following is a simple example that illustrates the singleton design pattern,

Step 1 : Create a node js file that creates and returns a single instance of a compute function. This compute function computes the sum of two numbers.

sum.js

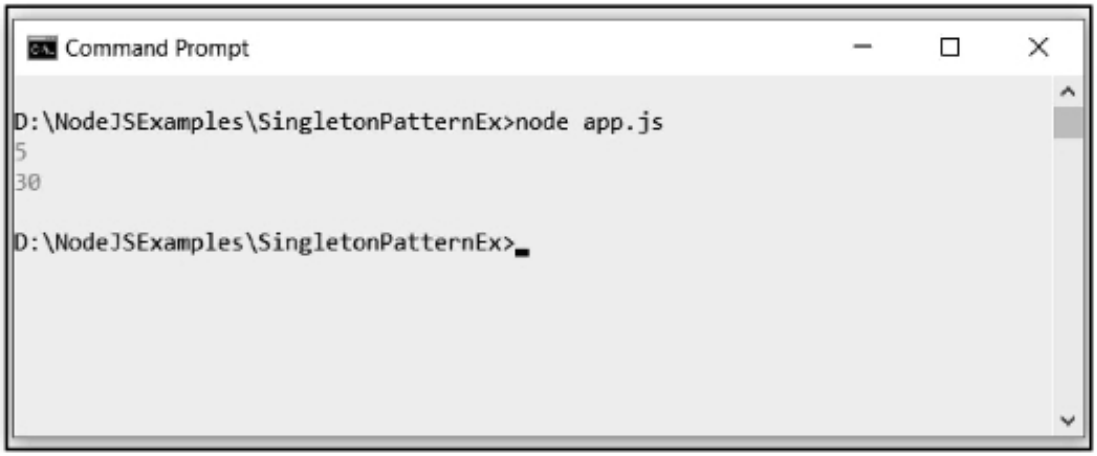
```
function compute(a,b) {
    return a+b;
}
module.exports.compute = compute;
```

Step 2 : Now create another file named **app.js**, in which we will use **require** statement to get the above defined module. It does not matter how many times you will require this module in your application **;it will only exist as a single instance**

app.js

```
var result = require('./sum'); //single instance only
//Even though used multiple times, a single instance is getting used.
console.log(result.compute(2,3));
console.log(result.compute(10,20));
```

Output



2) Observer Pattern

The observer pattern is a design pattern that defines a link between objects so that when one object's state changes, all dependent objects are updated automatically. This pattern allows communication between objects in a loosely coupled manner.

In this pattern, An object **maintains a list of dependents/observers and notifies them** automatically on state changes.

In Node.jst make an element 'observable' (our subject) we need it to use the 'EventEmitter' class. This way, our service, will get exactly the methods it needs:

1) **emit(eventName) :** To emit an event named 'eventName'.

2) on(eventName, callback) : To listen to an event and react with a callback function.

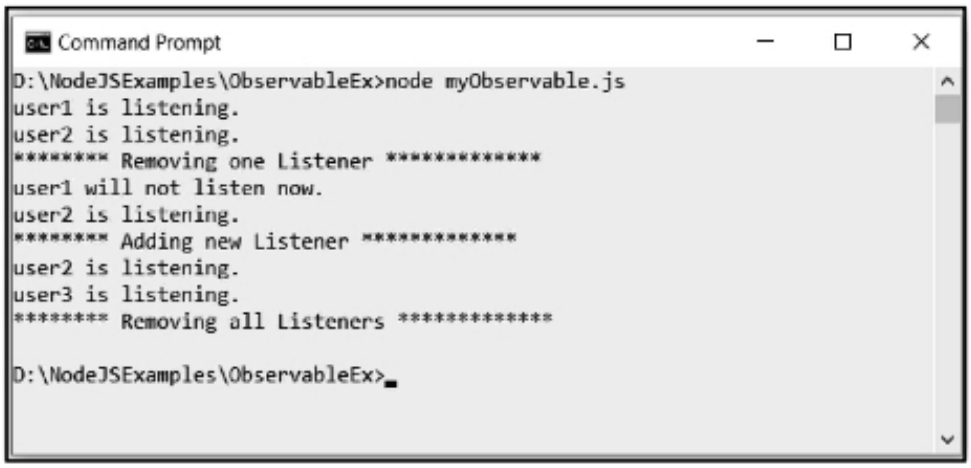
Following is a simple example that shows how observer pattern,

myObservable.js

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// user #1
var user1 = function listner1() {
  console.log('user1 is listening.');
}

// user #2
var user2 = function listner2() {
  console.log('user2 is listening.');
}

// user #3
var user3 = function listner3() {
  console.log('user3 is listening.');
}

// Bind the connection event with the listener1 function
eventEmitter.addListener('connection', user1);

// Bind the connection event with the listener2 function
eventEmitter.on('connection', user2);

// Fire the connection event
eventEmitter.emit('connection');

console.log("******** Removing one Listener **************");
// Remove the binding of listener1 function
eventEmitter.removeListener('connection', user1);
console.log("user1 will not listen now.");
eventEmitter.emit('connection');

console.log("******** Adding new Listener **************");
// Add the binding of listener3 function
eventEmitter.addListener('connection', user3);
eventEmitter.emit('connection');

console.log("******** Removing all Listeners **************");
```

```
// Remove the binding of listner1 function
eventEmitter.removeListener('connection', user2);
eventEmitter.removeListener('connection', user3);
eventEmitter.emit('connection');
```

Output



(3) Factory Pattern

Factory pattern is a design pattern which is responsible for creating the instance of several derived classes. Following example shows how to implement factory pattern.

Step 1 : We will create a parent class named Laptop. The code is as follows -

Laptop.js

```
class Laptop {
  constructor(name) {
    this.name = name.toString();
  }

  showInfo() {
    console.log("Brand Name: "+this.name)
  }
}
module.exports = Laptop;
```

Step 2 : Now we will create three more classes named: Dell, Hp and Lenovo that extend the above class Laptop

Dell.js

```
const Laptop = require('./Laptop');
class Dell extends Laptop {
  constructor () {
    super('DELL')
  }
```

```
}
module.exports = Dell;
```

Hp.js

```
const Laptop = require('./Laptop');

class Hp extends Laptop {
  constructor() {
    super('HP')
  }
}
module.exports = Hp;
```

Lenovo.js

```
const Laptop = require('./Laptop');

class Lenovo extends Laptop {
  constructor() {
    super('LENOVO')
  }
}
module.exports = Lenovo;
```

Step 3 : Now we will create one factory class named **LaptopFactory**. This class has a **create** method that generates new Laptop based on input string type.

Laptop_factory.js

```
const Dell = require('./Dell');
const Hp= require('./Hp');
const Lenovo = require('./Lenovo');

class LaptopFactory {
  create(type) {
    switch (type) {
      case 'Dell':
        return new Dell();
      case 'Hp':
        return new Hp();
      case 'Lenovo':
        return new Lenovo();
      default:
        {
          console.log('Unknown Lapto brand...');
        }
    }
```

```
 }
}
module.exports = new LaptopFactory();
```

Step 4 : Now we will call the factory classes and try to display the contents of these classes by calling **showInfo()** method in a app.js file.

app.js

```
const LaptopFactory = require('./laptop_factory');

const Dell = LaptopFactory.create('Dell');
const Hp = LaptopFactory.create('Hp');
const Lenovo = LaptopFactory.create('Lenovo');
const Dell2 = LaptopFactory.create('Dell');

Dell.showInfo();
Dell2.showInfo();
Hp.showInfo();
Lenovo.showInfo();
```

Output

5.12 Caching and Scalability

Caching

A cache is a **memory** that stores the information during tun time. By using a cache, the server application returns the information from memory instead of executing the code to process subsequent requests.

Node.js can be **optimized** with caching services. Without a cache, the server application will have to execute the entire code and retrieve information from the database each time it processes a request.

Client-side caching is the temporary storing of contents such as HTML pages, CSS stylesheets, JS scripts, and multimedia contents. Client caches help limit data cost by keeping commonly referenced data locally on the browser

Scaling

Scaling is used in node.js for increasing the performance and availability of the client server application.

Scaling allows the Node.js to handle multiple requests simultaneously without any performance issue.

There are **two approaches** used in scaling the node.js,

1) **Horizontal Scaling :** The horizontal scaling is a technique in which the application instances are duplicated to manage large number of incoming connections. This can be done on single multi-core system or different machines can also be used for this purpose.
2) **Vertical Scaling :** The vertical scaling is a technique in which the performance of machine or its memory is increased.

Thus scaling in node.js helps to increase the availability of the server.

❑❑❑

Notes

6 Database Programming with Node JS and MongoDB

Syllabus

Basics of MongoDB, Data types, Connect Node JS with MongoDB, Operations on data (Insert, Find, Query, Sort, Delete, Update) using Node JS.

Contents

6.1 Basics of MongoDB

- MongoDB is an **open source, document based database.**
- It is developed and supported by a company named 10gen which is now known as **MongoDB Inc.**
- The first ready version of MongoDB was released in March 2010.

Why MongoDB is needed ?

There are so many efficient RDBMS products available in the market, then why do we need MongoDB? Well, all the modern applications require Big data, faster development and flexible deployment. This need is satisfied by the document based database like MongoDB.

Features of MongoDB

1) It is **a schema-less, document based database system.**
2) It provides **high performance** data persistence.
3) It supports **multiple storage engines.**
4) It has a **rich query language support.**
5) MongoDB provides **high availability** and **redundancy** with the help of replication. That means it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
6) MongoDB provides horizontal **scalability** with the help of **sharding.** Sharding means to distribute data on multiple servers.
7) In MongoDB, every field in the document is indexed as primary or secondary. Due to which data can be searched very efficiently from the database.

SQL Structure Vs. MongoDB

Following figure shows the terms in SQL are treated differently in MongoDB. In MongoDB the data is not stored in tables, instead of that, the there is a concept called **collection** which is analogous to the **tables**. In the same manner the **rows** in RDBMS are called **documents** in MongoDB, likewise the **columns** of the record in RDBMS are called **fields**.

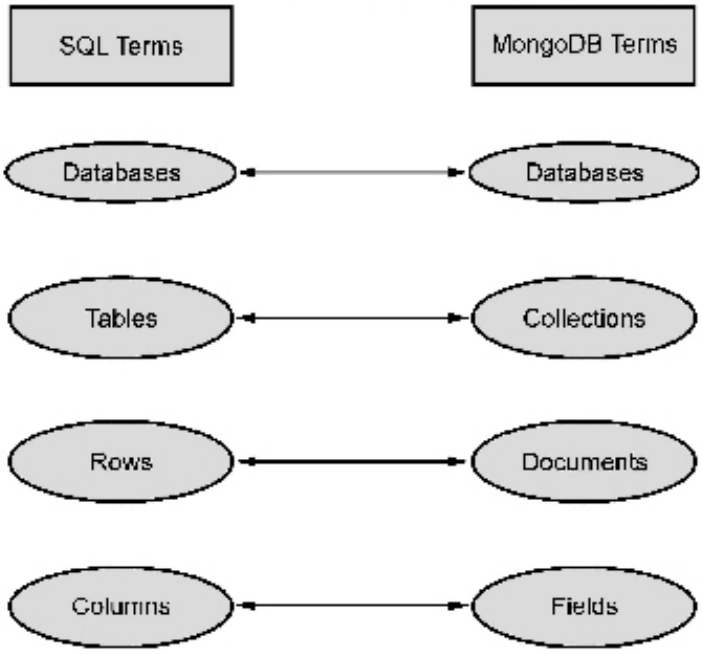


Fig. 6.1.1

Consider a student database as follows –

To the left hand side we show the database in the form of table and to the right hand side the database is shown in the form of collection.

Name	Course	AddressLine1	AddressLine2
Ankita	Computer Engineering	Lane1	Lane2
Prajkta	Mechanical Engineering	Street1	Street2
Kavita	Civil Engineering	Road1	Road2

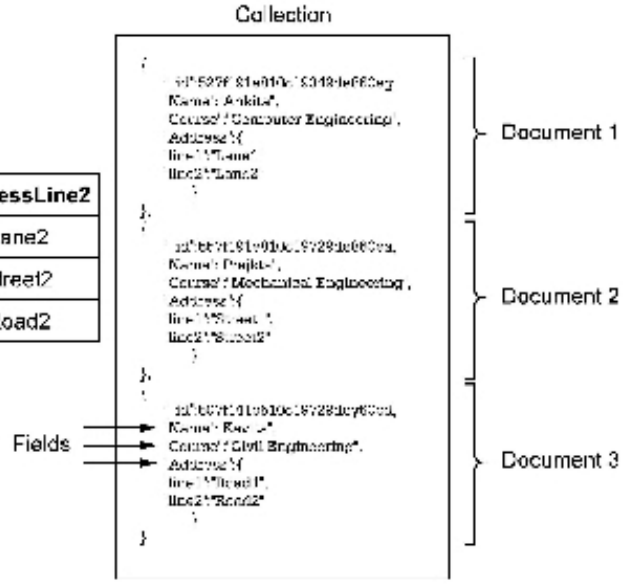


Fig. 6.1.2

Review Question

1. *List and explain various features of MongoDB.*

6.2 Data Types

Following are various types of data types supported by MongoDB.

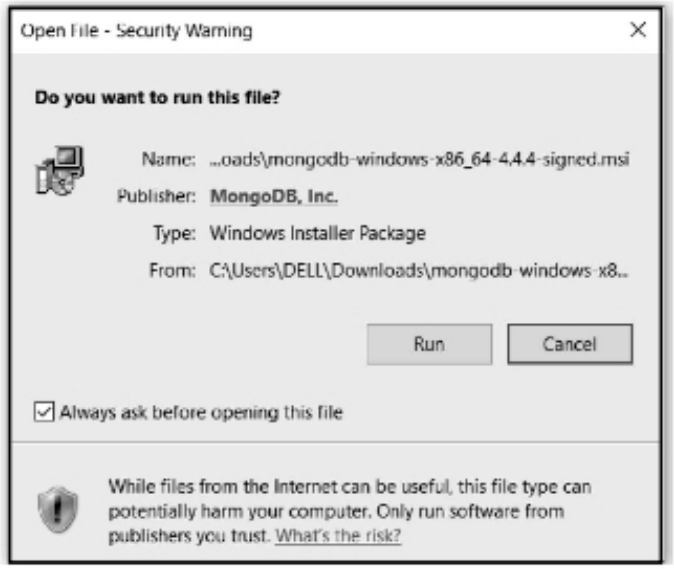
1) **Integer :** This data type is used for storing the numerical value.
2) **Boolean :** This data type is used for implementing the Boolean values i.e. true or false.
3) **Double :** Double is used for storing floating point data.
4) **String :** This is the most commonly used data type used for storing the string values.
5) **Min/Max keys :** This data type is used to compare a value against the lowest or highest BSON element.
6) **Arrays :** For storing an array or list of multiple values in one key, this data type is used.
7) **Object :** The object is implemented for embedded documents.
8) **Symbol :** This data type is similar to string data type. This data type is used to store specific symbol type.
9) **Null :** For storing the null values this data type is used.
10) **Date :** This data type is used to store current date or time. We can also create our own date or time object.
11) **Binary data :** In order to store binary data we need to use this data type.
12) **Regular expression :** This data type is used to store regular expression.

6.3 MongoDB Installation

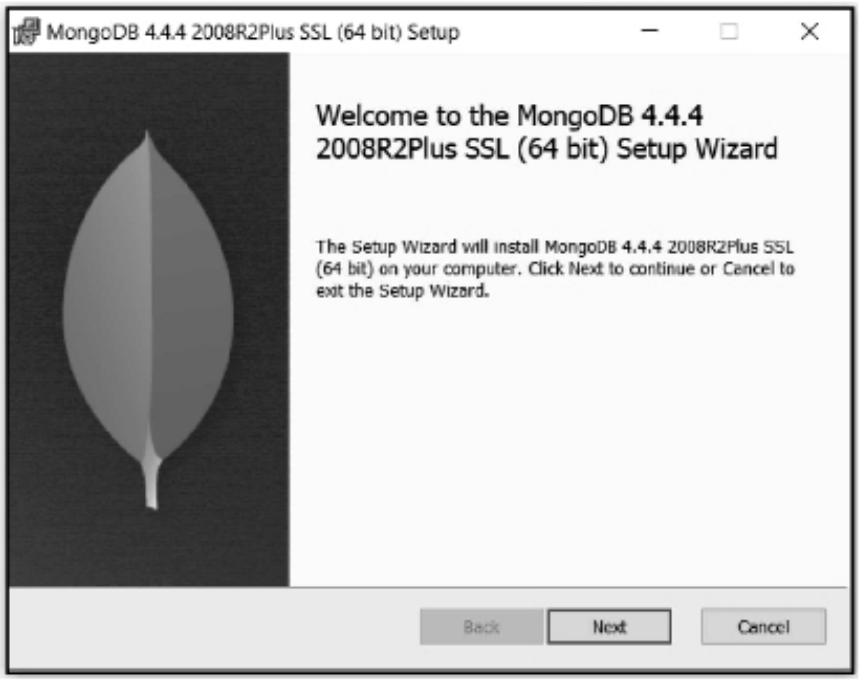
For installing the MongoDB go to the web site

https://www.mongodb.com/try/download/community

Choose **Software->Community Server.** The executable file gets downloaded. Click on **Run** to execute it.

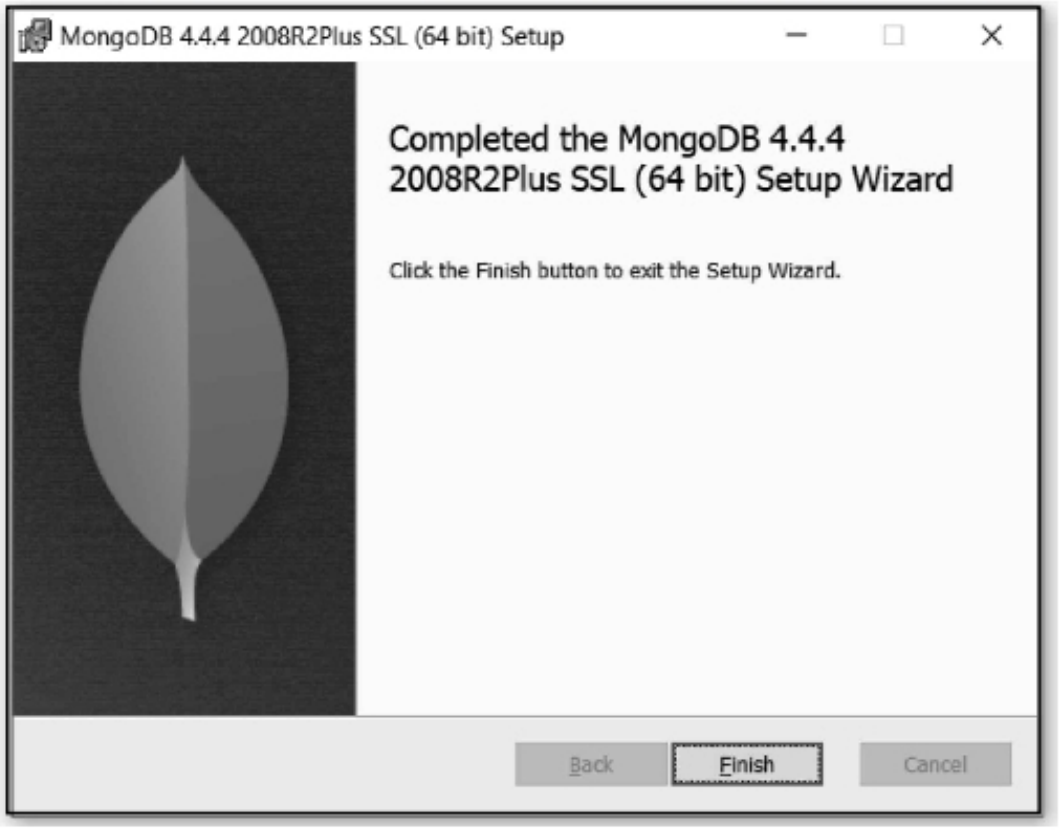


When the installation process starts following window gets popped up



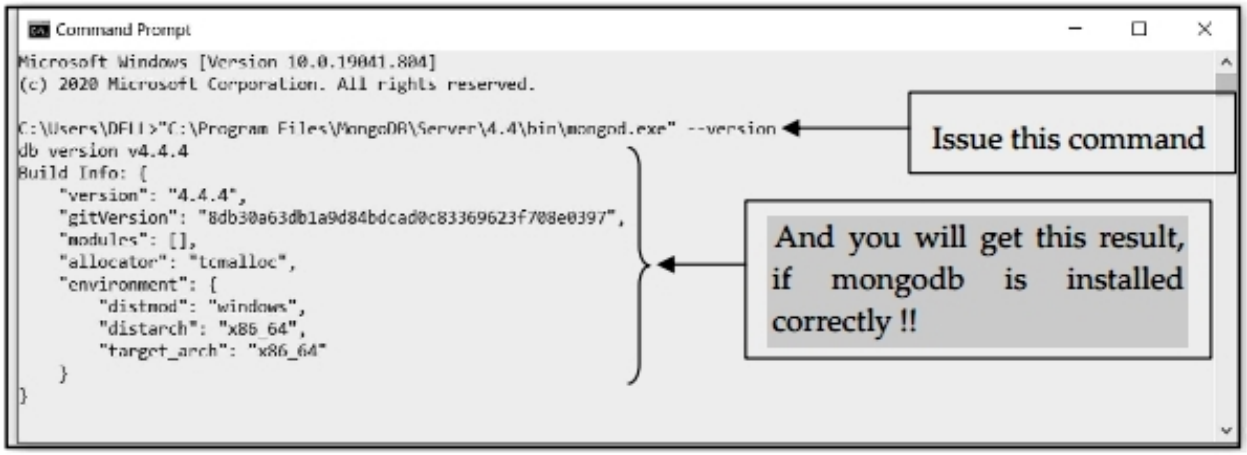
Just click on **Next** button and choose the **Complete** option for installation. Follow the normal procedure of installation by clicking the Next button.

Finally you will get following window on successful installation.

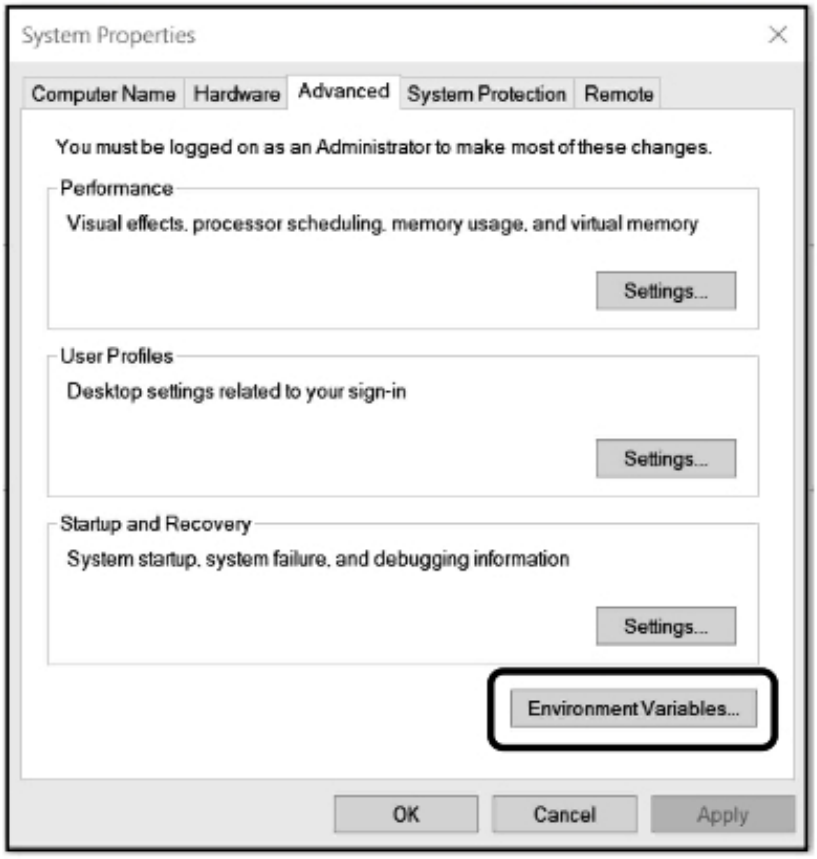


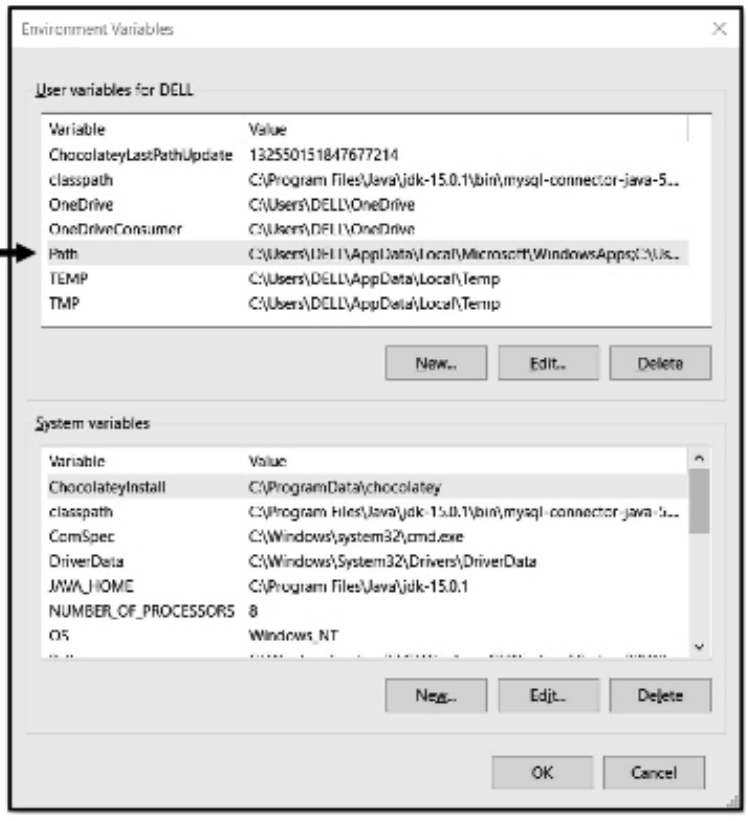
Click on Finish button, to complete the installation process.

In order to verify whether it is installed correctly or not, go to command prompt window and execute the command for mogod.exe file's version. It is illustrated as follows –

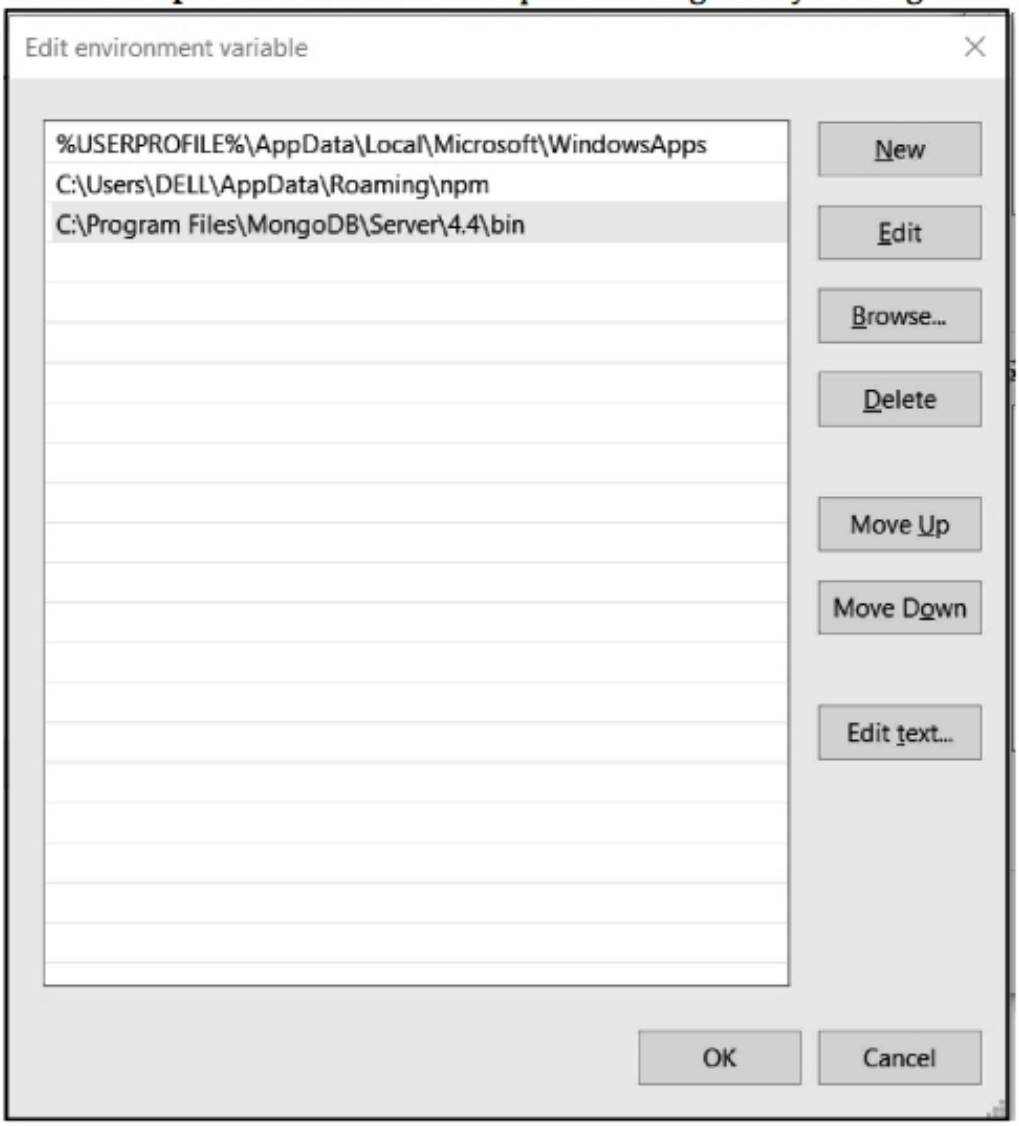


To set the environment variable, Open System Properties and click on **Environment Variables**


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply


Environment Variables
User variables for DELL
Variable
Value
ChocolateyLastPathUpdate
132550151847677214
classpath
C:\Program Files\Java\jdk-15.0.1\bin\mysql-connector-java-5...
OneDrive
C:\Users\DELL\OneDrive
Path
TEMP
C:\Users\DELL\AppData\Local\Temp
TMP
C:\Users\DELL\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ChocolateyInstall
C:\ProgramData\chocolatey
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
JAVA_HOME
C:\Program Files\Java\jdk-15.0.1
NUMBER_OF_PROCESSORS
8
OS
Windows_NT
New...
Edit...
Delete
OK
Cancel

Then click on the **path** variable and set the path of MongoDB by clicking **New** button.



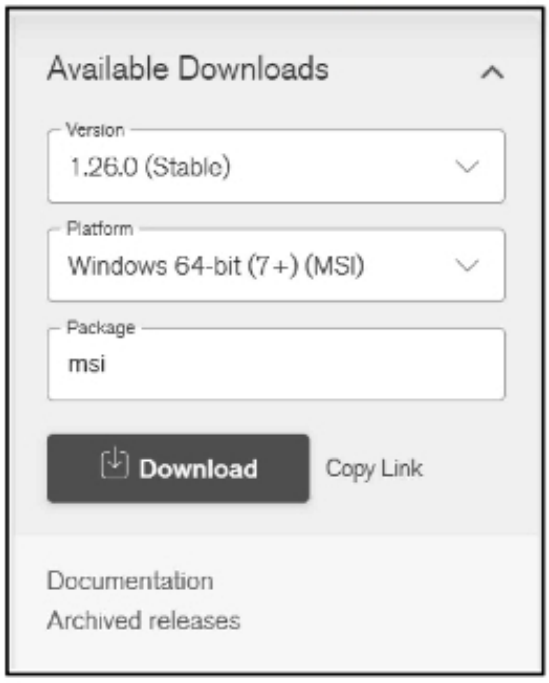
Then simply continue clicking ok button and just come out of the environment variable window.

Restart your command prompt window and now simply issue the command **mongod** and then **mongo** at the command prompt window. It will recognise this command and > prompt will appear

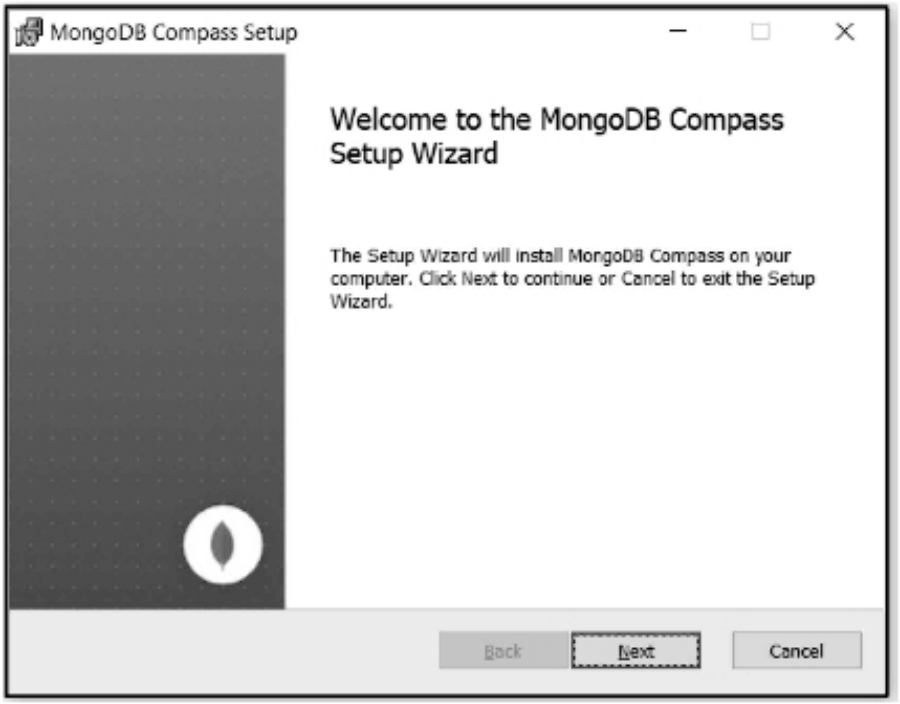
Installing Mongodb Compass (Graphical Tool)

This tool is very useful for handling MongoDB database with the help of simple graphical user interface.

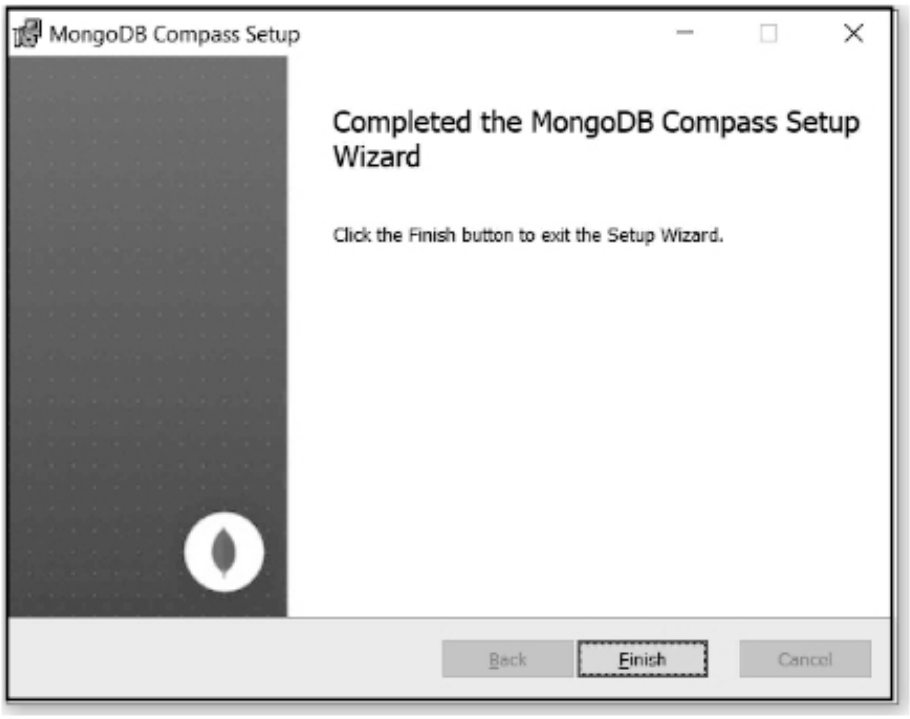
Select any suitable version as per your operating system. As mine is a Windows operating system of 64 bit, I have chosen following option.



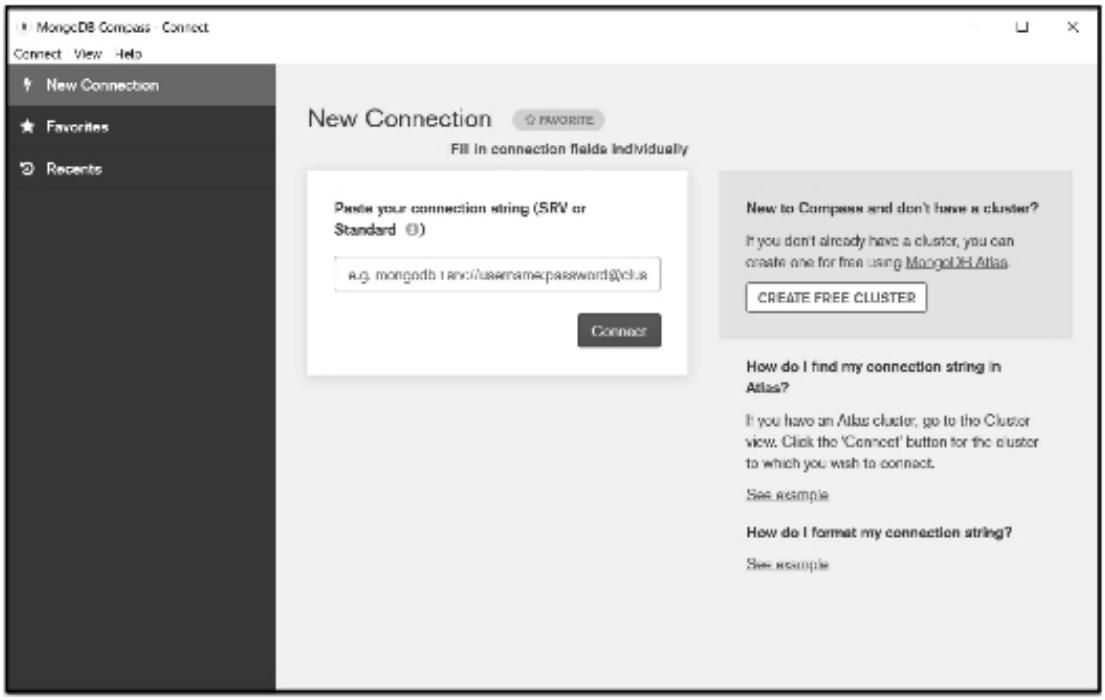
Click on the Download button. The exe file will get downloaded. Double click the installer file which is downloaded in your PC and the installation process for MongoDB Compass will start.



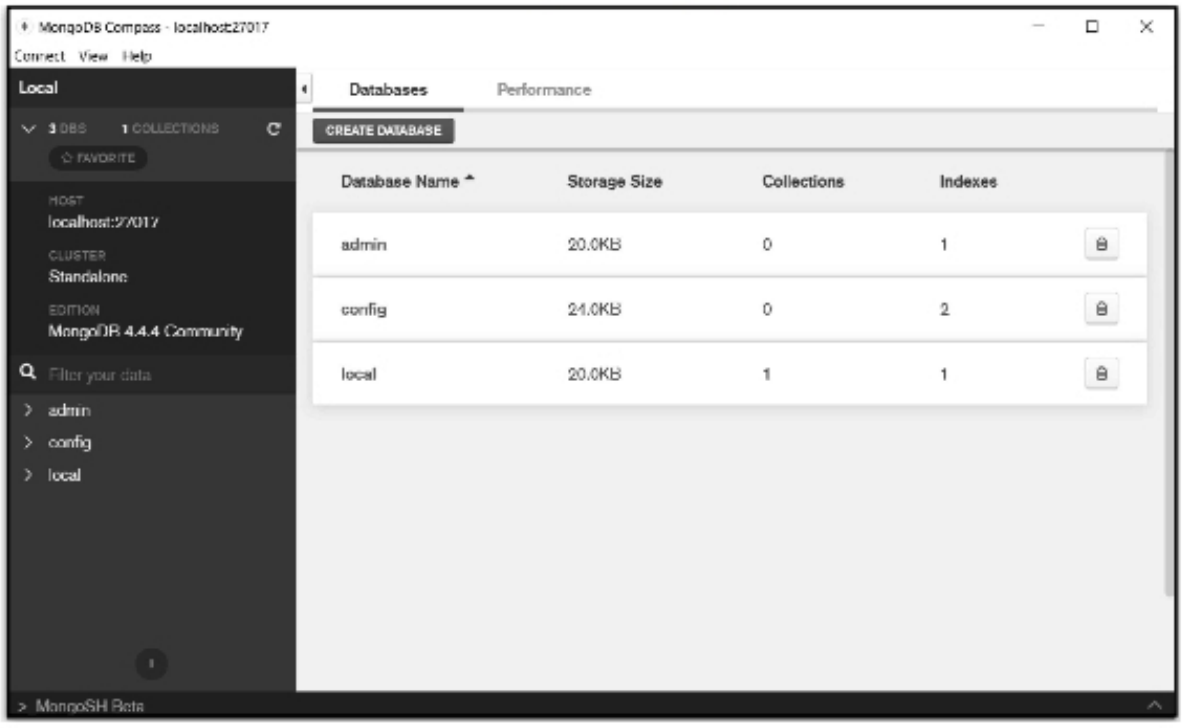
Simply go on clicking **Next** button, and then click on the install button on the subsequent window. Finally you will get the installation completion screen.



Just click Finish button. Just click the **Start** button of Windows, locate MongoDB Compass Application and simply click it to start the GUI for Mongo DB. You will get following GUI



If we click the connect button then we get following screen



6.4 Database Commands

In this section we will discuss how to create and handle database in MongoDB using various commands.

(1) Create Database

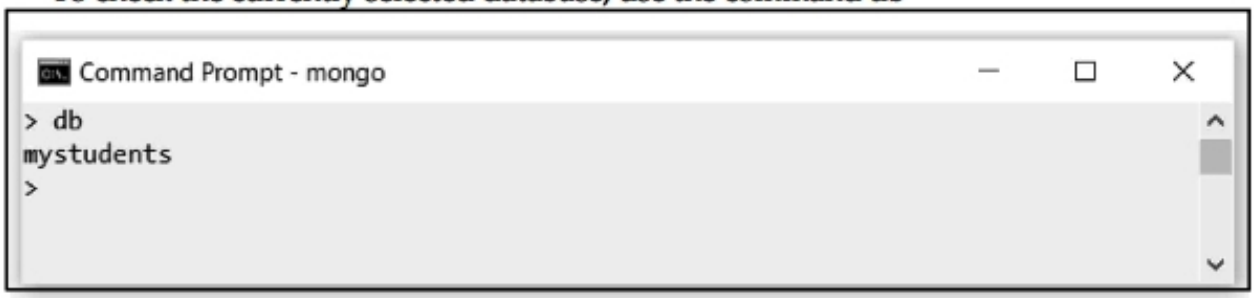
Open the command prompt and type the command mongo for starting the mongoDB. The > prompt will appear. For creating a database we need to use the "use" command.

Syntax

```
use Database_name
```

For example

To check the currently selected database, use the command **db**



We can see the list of databases present in the MongoDB using the command **show dbs**

Note that in above listing we can not see the **mystudents** database. This is because we have not inserted any **document** into it. To get the name of the database in the listing by means of **show** command, there should be some record present in the database.

(2) Drop Database

The **dropDatabase()** command is used to delete the database. For example

(3) Create Collection

There are two approaches of creating a collection

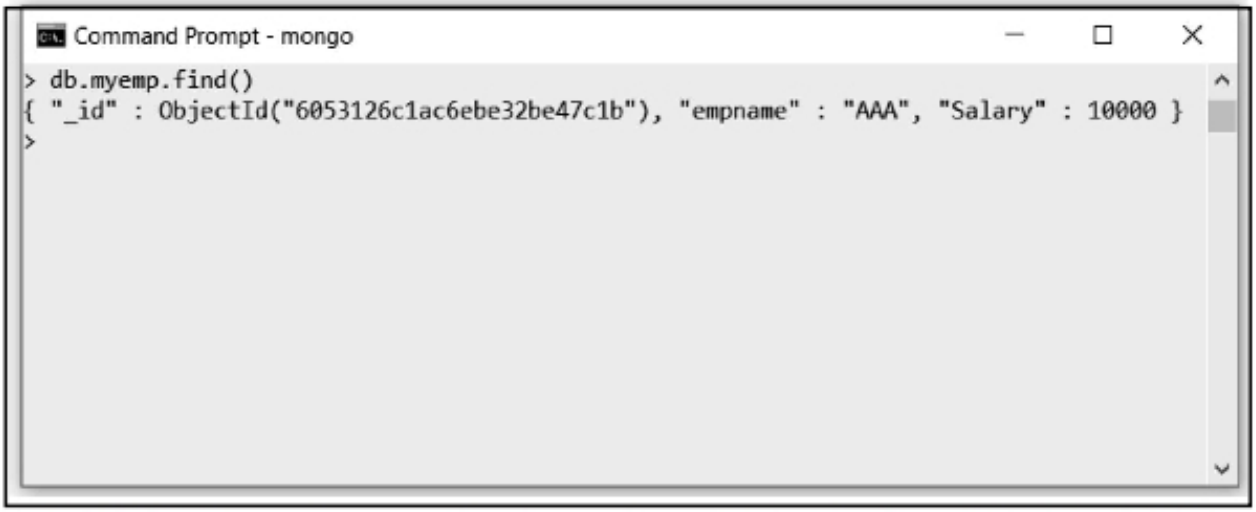
Method 1 : We can create a collection directly when we insert a document.

Syntax

```
db.collection_name.insert({key1:value1,key2:value2})
```

For example -

We can cross-verify whether the collection is created or not by using following command



Note that the one document(analogous to row) is getting inserted in the collection named **myemp**.

Method 2 : We can create **collection** explicitly using **createCollection** command.

Syntax

```
db.createCollection(name,options)
```

where

name is the name of collection

options is an optional field. This field is used to specify some parameters such as size, maximum number of documents and so on.

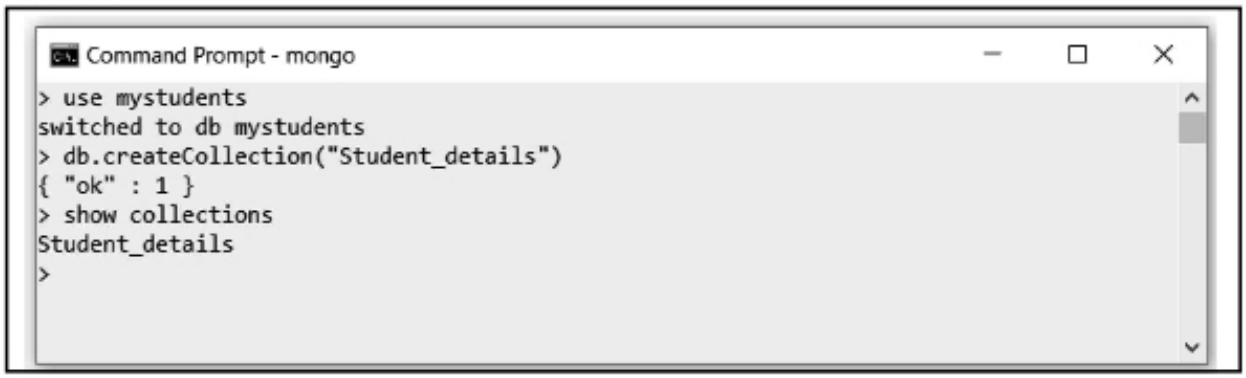
Following is a list of such options.

Field	Type	Description
capped	Boolean	Capped collection is a fixed size collection. It automatically overwrites the oldest entries when it reaches to maximum size. If it is set to true, enabled a capped collection. When you specify this value as true, you need to specify the size parameter.
autoIndexID	Boolean	This field is required to create index id automatically. Its default value is false.
size	Number	This value indicates the maximum size in bytes for a clapped collection.
Max	Number	It specifies the maximum number of documents allowed in capped collection.

For example – Following command shows how to create collection in a database using explicit command

(4) Display Collection

To check the created collection use the command "show collections" at the command prompt



(5) Drop Collection

The drop collection command is actually used to remove the collection completely from the database. The drop command removes a collection completely from database.

Syntax

```
db.collection_name.drop()
```

For example

We can verify the deletion of the collection by using "show collections" in the database.

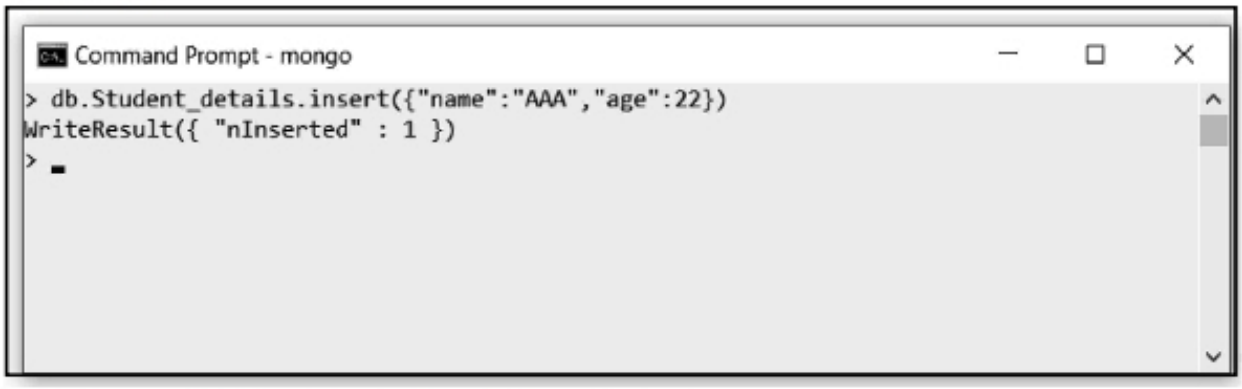
(6) Insert Documents

The document is inserted within the collection. The document is analogous **to rows** in database.

Syntax

```
db.collection_name.insert({key,value}
```

For example

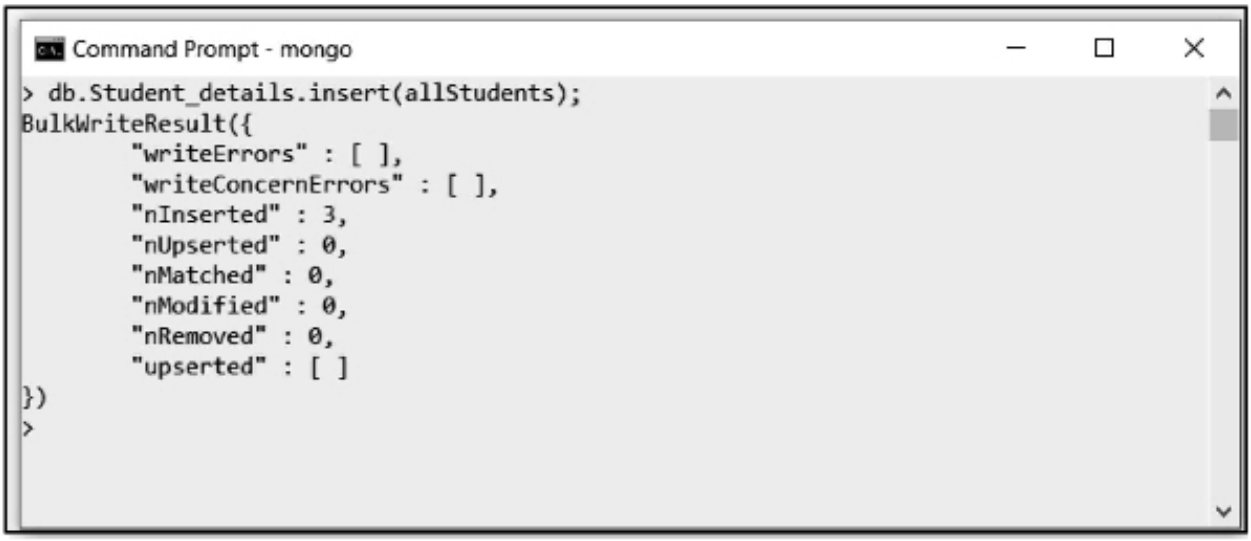


We can verify this insertion by issuing following command

Inserting Multiple Documents

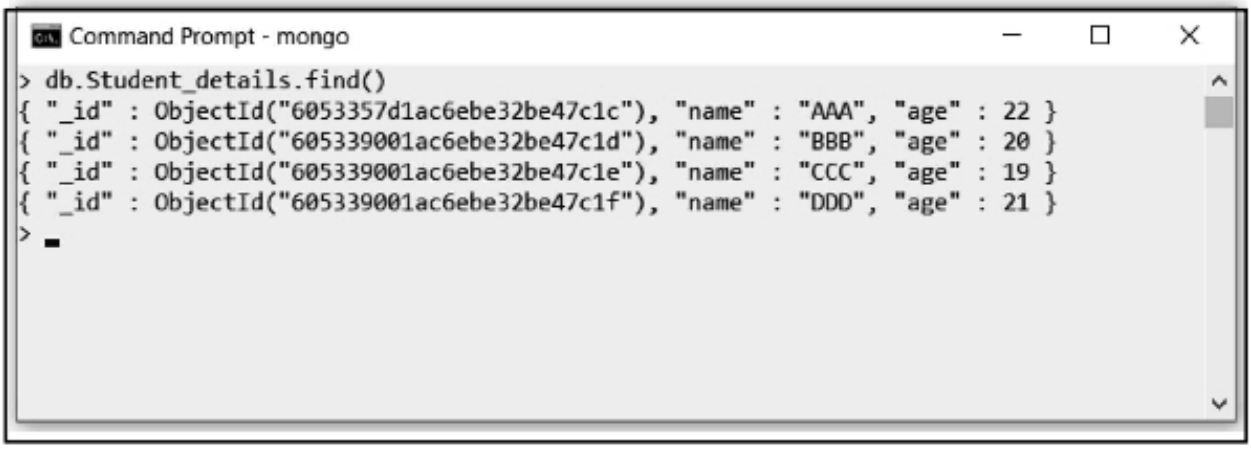
It is possible to insert multiple documents at a time using a single command. Following screenshot shows how to insert multiple documents in the existing collection.

Then you will get



In above screenshot, as you can see that it shows number 3 in front of **nInserted**. This means that the 3 documents have been inserted by this command.

To verify the existence of these documents in the collection you can use **find** command as follows –



(7) Delete Documents

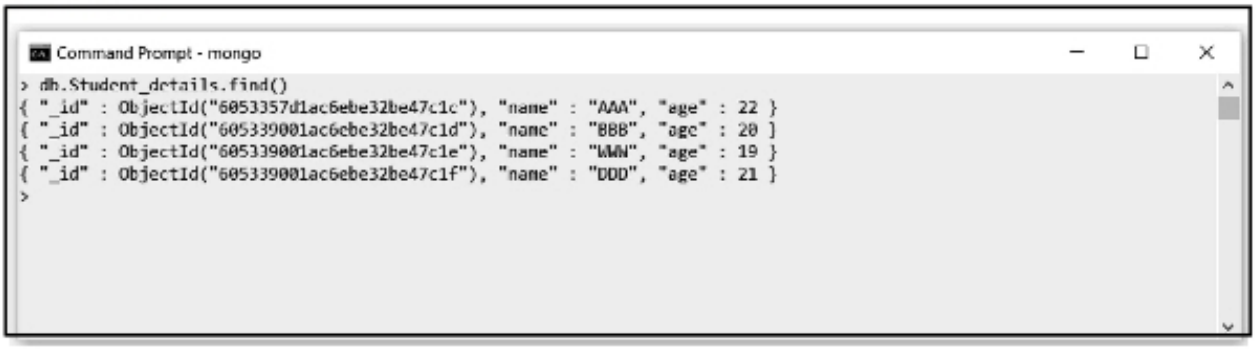
For deleting the document the **remove** command is used. This is the simplest command.

Syntax

```
db.collection_name.remove(delete_criteria)
```

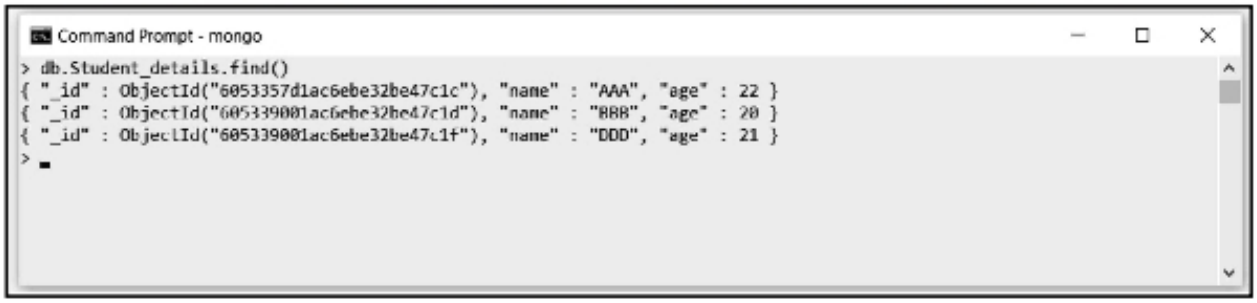
For example –

First of all we can find out the documents present in the collection using **find()** command



Now to delete a record with name "WWW" we can issue the command as follows –

Now using **find()** command we can verify if the desired data is deleted or not.



Deleting only one Document

Sometimes the delete criteria matches for more than one records and in such situation, we can forcefully tell the MongoDB to delete only one document.

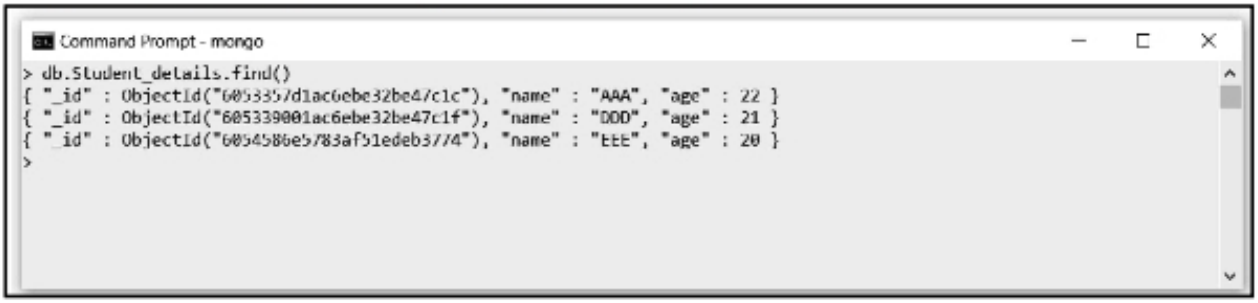
Syntax

```
db.collection_name.remove(delete_criteria, justOne)
```

The **justOne** is a Boolean value it can be 1 or 0. If we pass this value as 1 then only one document will get deleted.

For example

Now it can be verified using **find()** command as follows –



Note that there were two records that were matching with age = 20 with name "BBB" and "EEE", but since we have passed **justOne** attribute as 1, we get the result by deleting the single record having name "BBB"

Remove all the documents

It is possible to remove all the documents present in the collection with the help of single command.

Syntax

```
db.collection_name.remove({})
```

For example

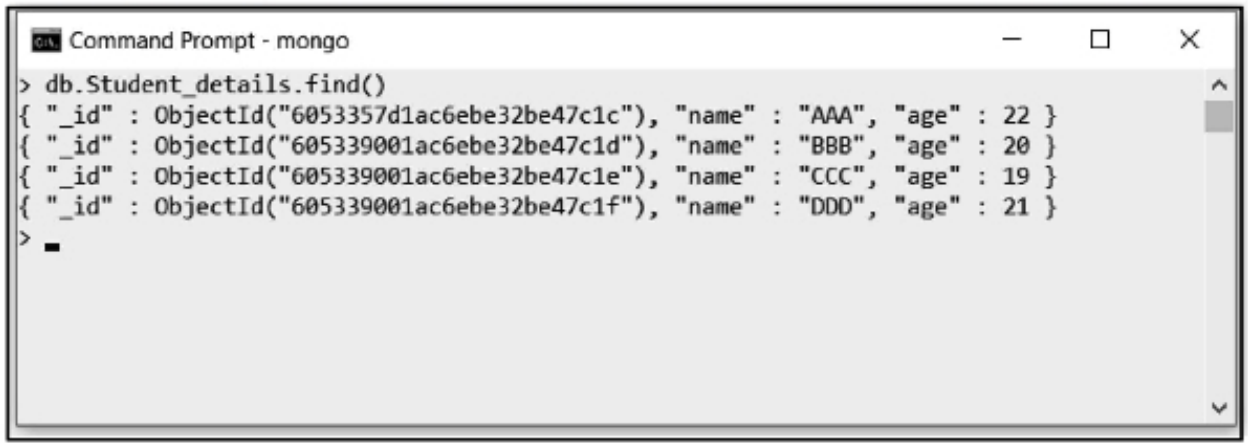
(8) Update Documents

For updating the document we have to provide some criteria based on which the document can be updated.

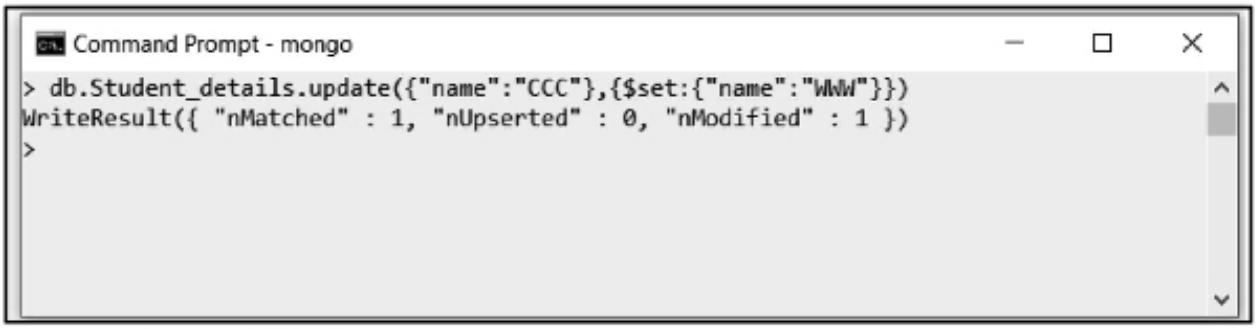
Syntax

```
db.collection_name.update(criteria,update_data)
```

For example – Suppose the collection "Student_details" contain following documents



And we want to change the name "CCC" to "WWW", then the command can be issued as



This can be verified as

Thus the document gets updated.

By default the **update** command updates a single document. But we can update multiple documents as well. For that purpose we have to add {multi:true}

For example

```
db.Student_details.update({"age":21},{$set:{"age":23}},{multi:true})
```

(9) Sorting

We can use the **sort()** method for arranging the documents in ascending or descending order based on particular field of document.

Syntax

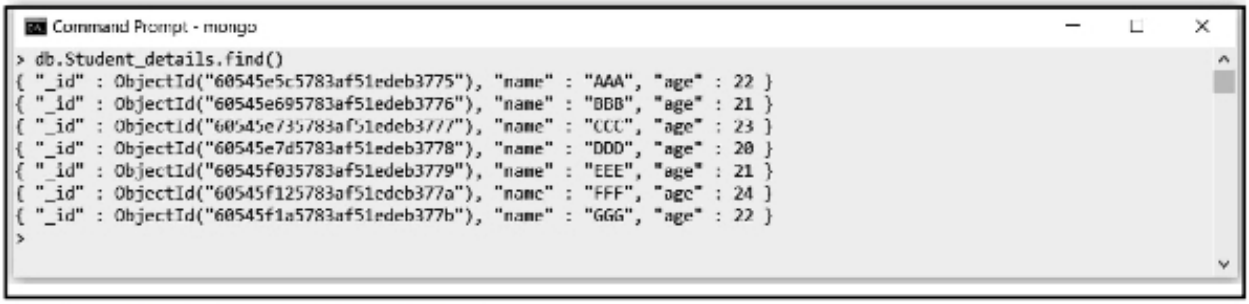
For displaying the documents in **ascending order** we should **pass value 1**

```
db.collection_name.find().sort({field_name:1})
```

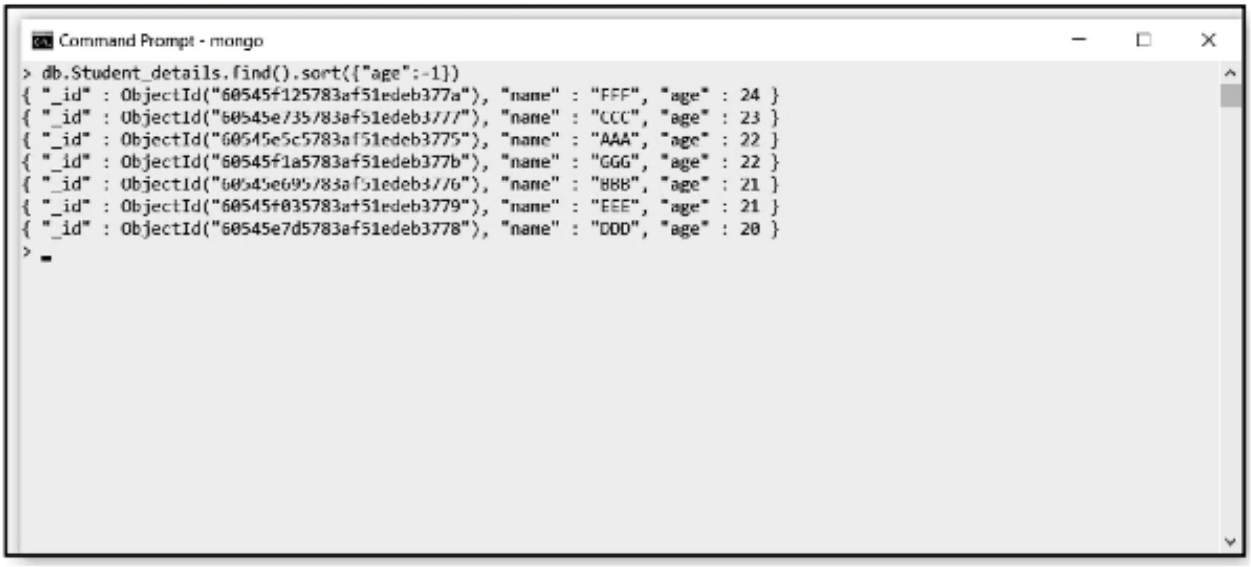
If we **pass -1** then the document will be displayed in the **descending order** of the field.

For example

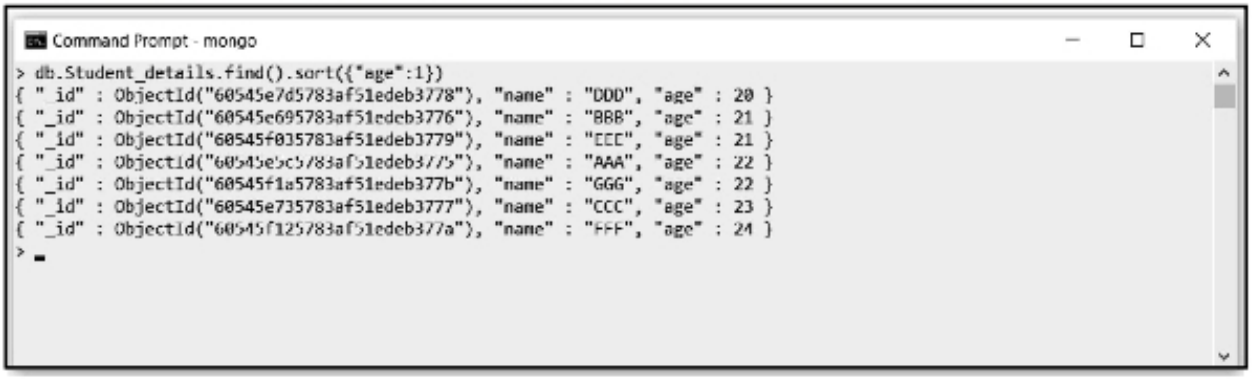
Suppose the collection contains following documents



Now to sort the data in descending order



If we want to display data in ascending order we issue following command



6.5 Connect Node JS with MongoDB

For connecting Node.js with MongoDB we need **MongoClient**. This can be created using following code.

Connect.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//create database
MongoClient.connect(url, function(err, db) {
 if (err)
            throw err;
  var dbo = db.db("studentDB");
      console.log("Connected with Database");
});
```

Program Explanation : In above code,

1) First of all, we have to import the module 'mongodb' using **require.** Thus we are creating MongoDB client.
2) Then specify the URL for MongoDB by means of hostname(localhost) and port number on which MongoDB is running (27017). This is a path at which we are going to create a database.
3) Then we are using **connect** method by passing the above URL. Inside this function a database object is created by a database name "studentDB".

Note that before execution of the above code, it is necessary to install the **mongodb** package using the following command at the current working directory.

```
Prompt:>npm install mongodb
```

6.6 Operations on Data using Node JS

The CRUD operations are performed in collaboration with Node.js and MongoDB. The CRUD stands for Create, Read, Update and Delete operations.

In this section we will discuss how to perform these operations on the database with the help of simple demo applications.

Prerequisite : For performing these operations we need to have MongoDB installed in our machine. One must also install MongoDB Compass (It is a graphical tool) to verify the operations performed by node.js on MongoDB.

Example Code : In this example, we will create a Student database with two fields – name and city.

Creation of Database

Step 1 : We will create a database studentDB and collection name as Student_Info. For that purpose make a folder in your current working directory. I have created the folder by a name **StudentDBExample.**

Step 2 : Open the command prompt go to the path of **StudentDBExample.** And issue the commands

```
D:\NodeJSExamples\StudentDBEXample>npm init
```

And press enter key to accept the default values. By this **package.json** file gets created.

Step 3 : Then install the **mongodb** module using following command

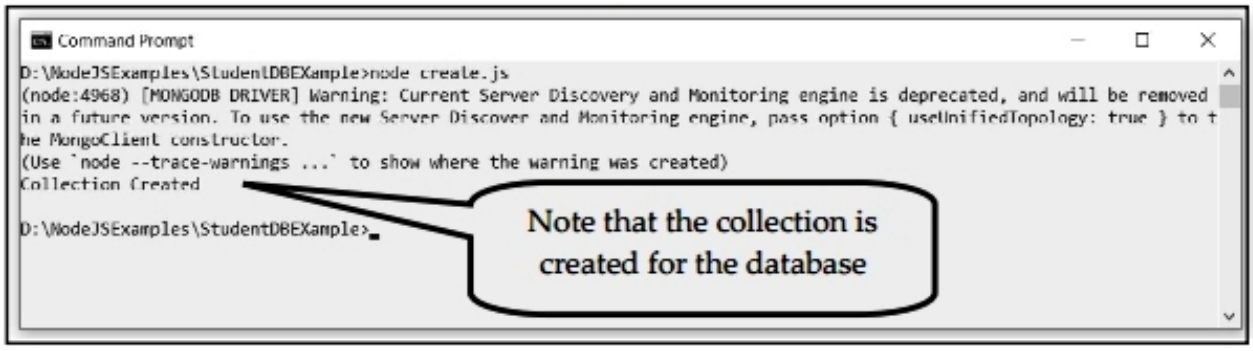
```
D:\NodeJSExamples\StudentDBEXample>npm install mongodb
```

Step 4 : Now create a node.js file for creating a database and collection. The code is as follows –

create.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//create database
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("studentDB");
  dbo.createCollection("Student_Info",function(err,res){
     if(err)
        throw err;
     console.log("Collection Created")
     });
     db.close();
  });
```

Output



Program Explanation : In above code,

1) First of all, we have to import the module 'mongodb' using **require.** Thus we are creating MongoDB client.

2) Then specify the URL for MongoDB by means of hostname (localhost) and port number on which MongoDB is running (27017). This is a path at which we are going to create a database.
3) Then we are using **connect** method by passing the above URL. Inside this function a database object is created by a database name "studentDB".
4) Then using the method **createCollection** for database object we can create a collection inside the database "studentDB". The first parameter passed to this method is name of the collection. Note that here the collection name is "Student_Info".
5) The message "Collection Created" is displayed on the console using **console.log** method.

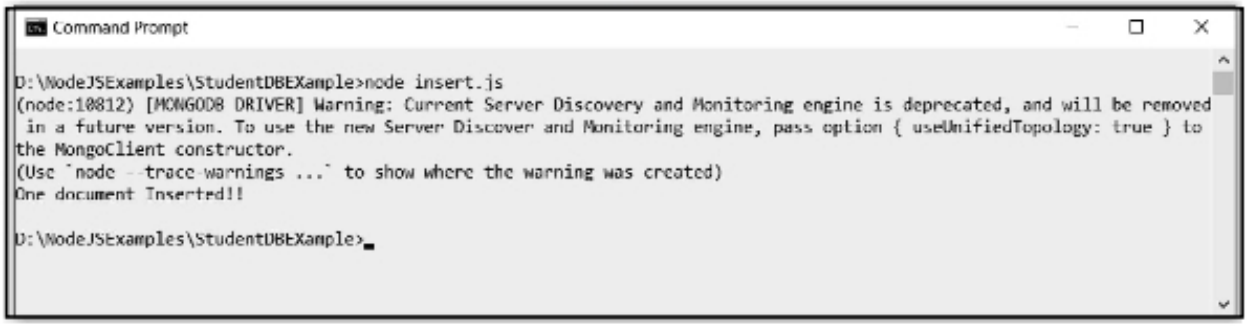
Insertion of Data

We can insert one document or multiple document at a time. First of all we will see the code for inserting one document inside the above created collection.

insert.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//Insert
MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 var dbo = db.db("studentDB");
 var mydata = { name: "AAA", city: "Pune" };
 dbo.collection("Student_Info").insertOne(mydata,function(err,res){
 if(err)
    throw err;
        console.log("One document Inserted!!");
        db.close();
    });
});
```

Output



Program Explanation : In above code,

1) First of all, we have to import the module 'mongodb' using **require.** Thus we are creating MongoDB client.
2) Then specify the URL for MongoDB by means of hostname(localhost) and port number on which MongoDB is running(27017). This is a path at which our database exists.
3) Then we are using **connect** method by passing the above URL. Inside this function a database object is created by a database name "studentDB". Thus now we can access to **studentDB**
4) Then a JSON object is in the form {name:value}. We insert data in the collection in the form of document. This document is created in the form of JSON. Hence the **mydata** object is created with some values.
5) This **mydata** is inserted into the collection using **insertOne** method.

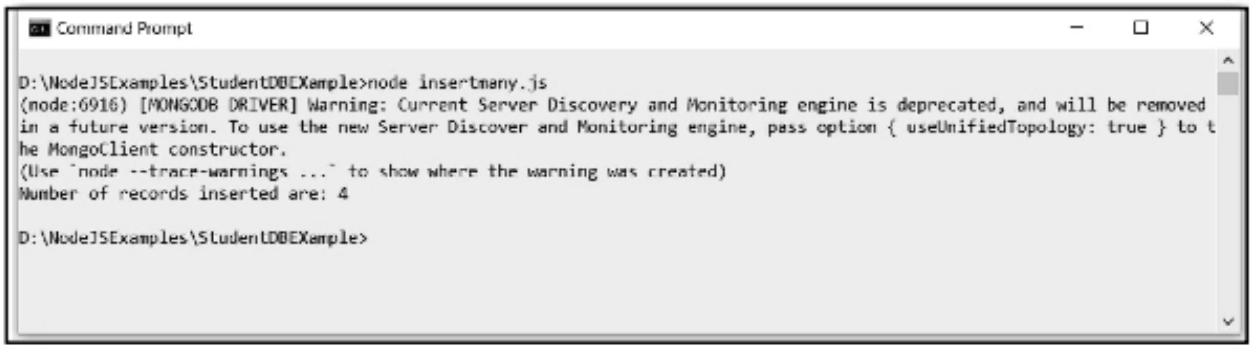
In the same manner we can insert multiple documents at a time. Following code illustrates it.

insertmany.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//Insert
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("studentDB");
  var mydata = [
        { name: "BBB", city: "Mumbai" },
        { name: "CCC", city: "Chennai" },
        { name: "DDD", city: "Delhi" },
```

```
        { name: "EEE", city: "Ahmedabad" }
 ];
 dbo.collection("Student_Info").insertMany(mydata,function(err,res){
 if(err)
    throw err;
        console.log("Number of records inserted are: "+res.insertedCount);
        db.close();
    });
});
```

Output



Program Explanation : In above code

We have created array of values in JSON object as Follows –

```
var mydata = [
        { name: "BBB", city: "Mumbai" },
        { name: "CCC", city: "Chennai" },
        { name: "DDD", city: "Delhi" },
        { name: "EEE", city: "Ahmedabad" }
 ];
```

Then using the **insertMany** command we can insert all the above documents at a time

```
dbo.collection("Student_Info").insertMany(mydata,function(err,res)
```

This program execution can be verified in MongoDB compass.

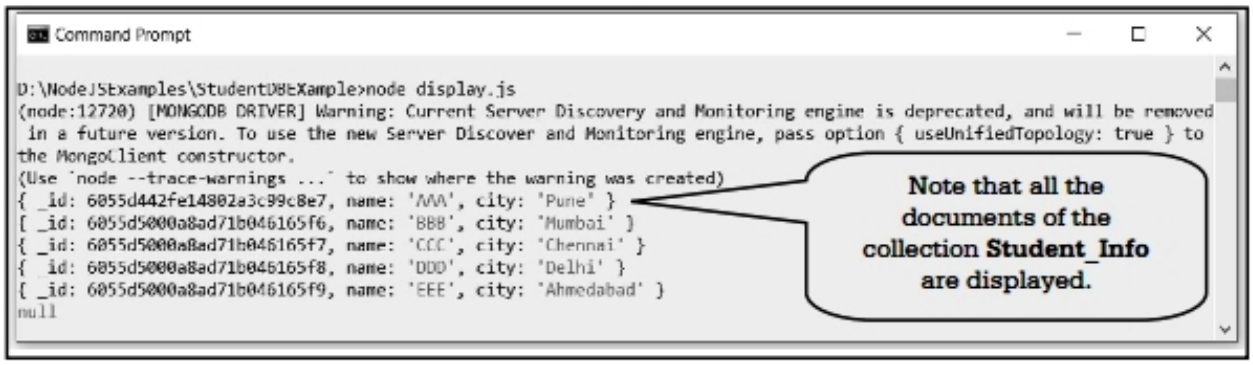
Read Data

We can read all the documents of the collection using **find** method. Following is a simple Node.js code that shows how to read the contents of the database

display.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//Read
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("studentDB");
  var cursor = dbo.collection("Student_Info").find({})
  cursor.each(function(err,doc){
      console.log(doc);
    db.close();
  });
});
```

Output



Program Explanation : In above code,

1) First of all, we have to import the module 'mongodb' using **require.** Thus we are creating MongoDB client.
2) Then specify the URL for MongoDB by means of hostname(localhost) and port number on which MongoDB is running(27017). This is a path at which our database exists.
3) Then we are using **connect** method by passing the above URL. Inside this function a database object is created by a database name "studentDB". Thus now we can access to **studentDB.**
4) Then for reading the contents present in the documents of the collection we use the **find()** method. The cursor is created for reading each record(document) of the collection.

```
var cursor = dbo.collection("Student_Info").find({})
```

Then using **cursor.each** function, each document is displayed within a callback function.

Updating Data

We can change one or more fields of the document using **updateOne** method.

update.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//Update
MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 var dbo = db.db("studentDB");
 var mydata = { name: "DDD"};
 var newdata = {$set: {name:"TTT",city:"Jaypur"}}
```

```
dbo.collection("Student_Info").updateOne(mydata,newdata,function(err,res){
if(err)
    throw err;
        console.log("One document Updated!!");
        db.close();
    });
});
```

Output

Again this updation can be verified by opening MongoDB compass

Program Explanation : In above code,

We are updating one document using **updateOne** method. This method require two parameter first one is the existing data, and second one is the new data which you want to put in place of the existing one. This can be done with the help of following lines of code

```
var mydata = { name: "DDD"};   //existing record with name = "DDD"
var newdata = {$set: {name:"TTT",city:"Jaypur"}} //replace it by a new data
```

Then **updateOne** method is called by passing these two parameters.

Deleting Data

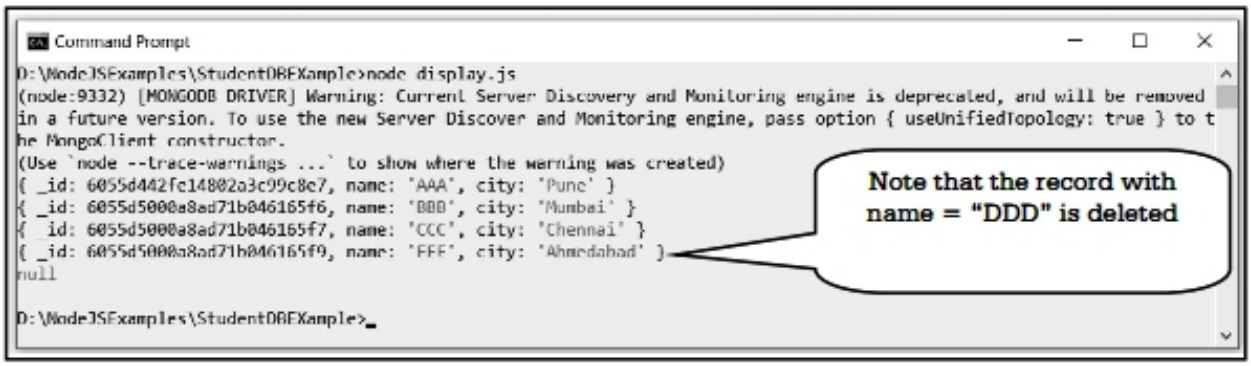
This is the operation in which we are simply deleting the desired document. Here to delete a single record we have used **deleteOne** method. The code is as follows –

delete.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//Delete
MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 var dbo = db.db("studentDB");
 var mydata = { name: "TTT"};
 dbo.collection("Student_Info").deleteOne(mydata,function(err,res){
 if(err)
    throw err;
        console.log("One document Deleted!!");
        db.close();
    });
});
```

Output

The above deletion operation can be verified by displaying the contents of the collection



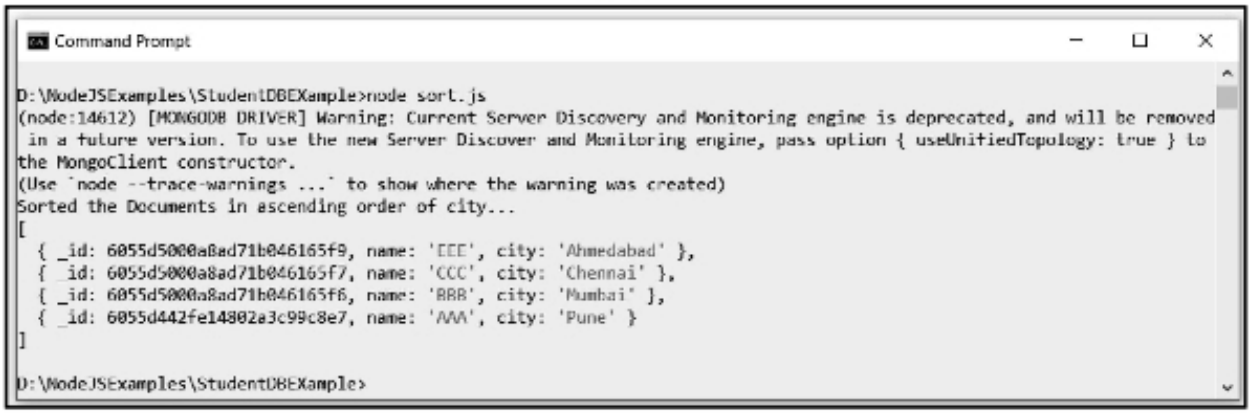
Sorting Data

Data can be arranged in sorted order in the collection. For displaying the sorted ordered data we use the method **sort**. Following code illustrates how to display the data in ascending order of the filed named **city**.

sort.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
//sort in ascending order
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("studentDB");
  var mydata = { city: 1};
  dbo.collection("Student_Info").find().sort(mydata).toArray(function(err,res){
  if(err)
     throw err;
         console.log("Sorted the Documents in ascending order of city...");
     console.log(res);
         db.close();
     });
});
```

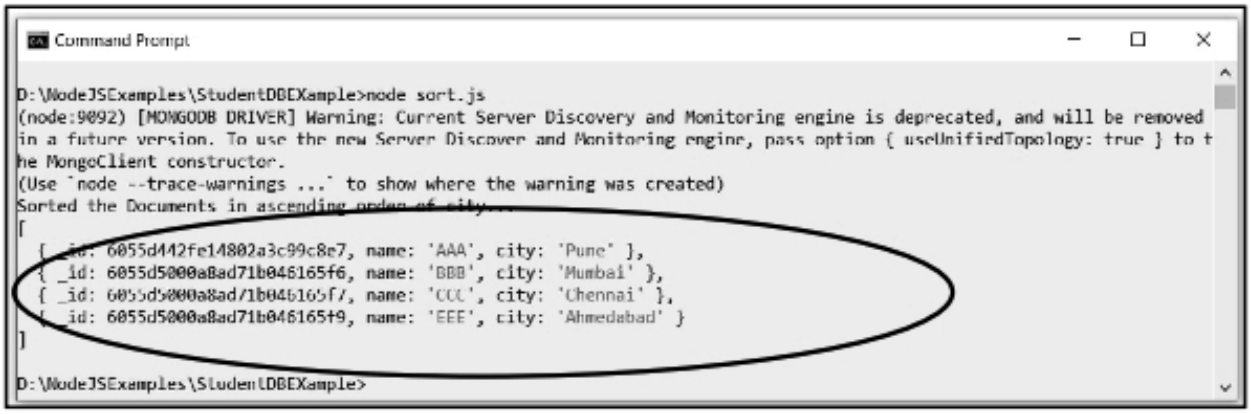
Output



Program Explanation : In above code to sort in ascending order we use **sort.** The key field based on which the document is to be sorted is **city**. We have passed 1 along with it for displaying the city names in ascending order. If we pass -1 then the cities are displayed in descending order. Simply change the above code as

```
var mydata = { city: -1};
```

Then the output will be



Review Questions

1. *Explain insertion and deletion database operation using NodeJs.*
2. *Write a program to display the sorted data in the database using NodeJs*

❑❑❑

Notes

SOLVED MODEL QUESTION PAPER

(As per New Syllabus)

Advanced Web Programming

Semester - VI (IT) Professional Elective - III

Time : 2 $\frac{1}{2}$ Hours] **[Total Marks : 70**

Instructions : 1) Attempt all questions.

2) Make suitable assumptions wherever necessary.

3) Figures to the right indicate full marks.

Q.1 **a)** *What are the benefits of CSS?* **(Refer section 1.1.1)** **[03]**

b) *Explain the use of font-families in CSS with the help of suitable example.* **(Refer section 1.7.1)** **[04]**

c) *Write a JavaScript for password verification.* **(Refer example 1.16.1)** **[07]**

Q.2 **a)** *Enlist the features of Angular JS* **(Refer section 2.2)** **[03]**

b) *Explain MVC with Angular JS* **(Refer section 2.5)** **[04]**

c) *Explain the controller directive in Angular JS with suitable example.* **(Refer section 2.8)** **[07]**

OR

c) *Explain the working of Scope object in Angular JS with suitable example* **(Refer section 2.9)** **[07]**

Q.3 **a)** *List and explain the use of three ng-directives in Angular JS.***(Refer section 3.1)** **[03]**

b) *Explain the concept of Single Page Application(SPA) in detail* **(Refer section 3.6)** **[04]**

c) *Write an Angular JS script to create a form having three radio buttons each indicating the name of particular color. Display the appropriate color name when particular radio button is pressed.* **(Refer section 3.4)** **[07]**

OR

Q.3 **a)** *List any three features of SPA* **(Refer section 3.6)** **[03]**

b) *Explain the use of routing in Angular Js with suitable programming example* **(Refer section 3.3)** **[04]**

c) *What is Data binding? Explain its types with suitable Angular JS code.* **(Refer section 3.5)** **[07]**

Q.4 **a)** *What are the various uses of Node.js?* **(Refer section 4.1)** **[03]**

b) *What is npm? Explain the use of package manage in Node.js with suitable example* **(Refer section 4.2)** **[04]**

c) *Explain the callbacks in node js with suitable example.* **(Refer section 4.5)** **[07]**

OR

Q.4 **a)** *Enlist the features of Node.js.***(Refer section 4.3)** **[03]**

b) *What is the use of npm init?Give example.* **(Refer section 4.2)** **[04]**

c) *Explain any four methods of console object in node.js with suitable examples.* **(Refer section 4.4)** **[07]**

Q.5 **a)** *Explain the concept of event emitter.* **(Refer section 5.1.1)** **[03]**

b) *What is REST API? Explain various parts of client request.* **(Refer section 5.9)** **[04]**

c) *What is event loop? Explain its working with suitable block diagram* **(Refer section 5.1.2)** **[07]**

OR

Q.5 **a)** *Enlist the features of MongoDB* **(Refer section 6.1)** **[03]**

b) *Explain the three commonly used set timer functions in node js* **(Refer section 5.2)** **[04]**

c) *Explain the creation of student database using MongoDB and Node.js. Also write the Node.js script to display the contents of student database.***(Refer section 6.6)** **[07]**

❑❑❑

Notes

Made in India