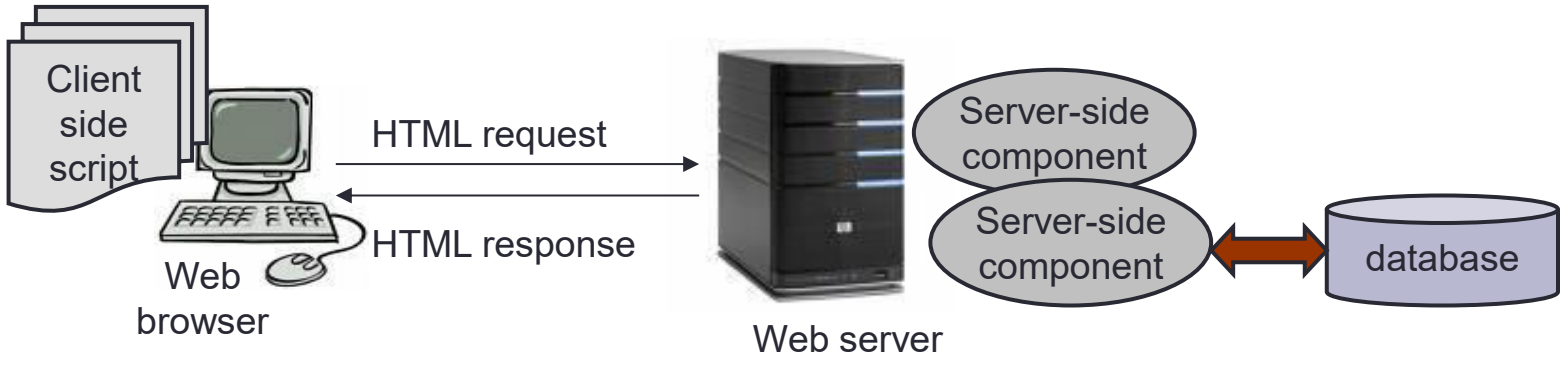
HTML

Web Applications – An Overview

- Desktop-apps versus server-side apps

- Client-side scripting versus Server-side coding:
  - Client-side scripts run on the client-side of a client-server application.
  - Server-side scripts/code run on the server-end of a client-server application.

- A web application is an application that is accessed via web browser over a network such as the Internet or an intranet
  - Web Client uses HTTP protocol to communicate with a Web Server
  - A Web component is a software entity that provides a response to a web request. For example: Servlets and JSP



What is HTML?

- HTML(Hyper Text Markup Language)
  - is a language for describing web pages.
  - not a programming language
  - uses markup tags to describe web pages.
- Most Web documents are created using HTML.
  - Documents are saved with extension .html or .htm.

- Markup?
  - Markup Tags are strings in the language surrounded by a < and a > sign.
  - Opening tag: <html> Ending tag: </html>
  - Not case sensitive.
  - Can have attributes which provide additional information about HTML elements on your page. Example
    - <body bgcolor="red">
    - <table border="0">

HTML

- An HTML document appears as follows:

```
<!DOCTYPE HTML>
<html>
  <head>
      <title>Title of page</title>
  </head>
  <body>
       This is my first homepage. <b>This text is bold</b>
   </body>
</html>
```

DOCTYPE tells type, version, language of particular document.

- **HTML Head Section**: contain information about the document. The browser does not display this information to the user. Following tags can be in the head section: <base>, <link>, <meta>, <script>, <style>, and <title>.
- **HTML Body Section**: defines the document's body. Contains all the contents of the document (like text, images, colors, graphics, etc.).
- Each document can have at most one <body> element

Document (Body) Contents

- Body Text
  - HTML truncates spaces in your text.
  - Use
 to insert new lines.
  - Use <p> tag to create paragraphs.
  - Use to hold division or a section in an HTML document
  - Use as an inline container to mark up a part of a text, or a part of a document

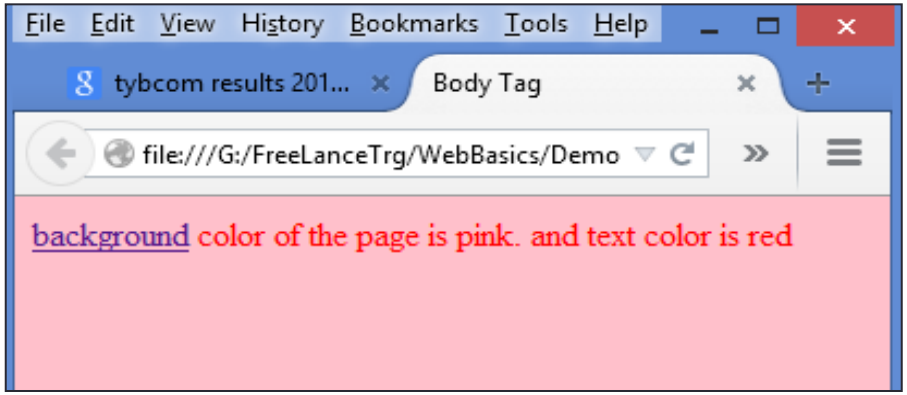
- Comments in HTML Document
  - Increase code readability; Ignored by the browser.
  - Example of HTML comment: <!-- This is a Sample HTML Comment -->

- … : Creates divisions in Web pages. Can be used to set the alignment/class for an entire section of the page. Eg:
  - This text is at the centre of the browser window

Attributes

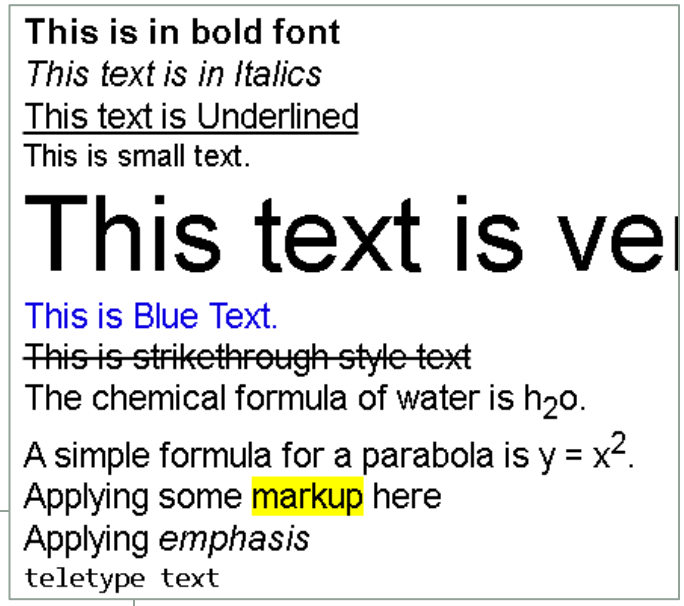
- HTML attributes are special words which provide additional information about the elements or attributes are the modifier of the HTML element.
- Eg of attributes used in <body> element :
  - BGCOLOR: Gives a background color to HTML page.
  - BACKGROUND: Use to specify images to the BACKGROUND.
  - TEXT: Specifies text color throughout the browser window
  - LINK, ALINK, VLINK: Used to specify link color, active link color & visited link color
- Examples:
  - <body text="red"> OR <body text="#FF0000">
  - <body link="red" alink="blue" vlink="purple">
  - <body bgcolor="black"> OR <body bgcolor="#000000">
  - <body background="http://www.mysite.edu/img1.gif">

```
<HTML>
 <HEAD>
   <TITLE>Body Tag</TITLE>
 </HEAD>
 <BODY  BGCOLOR="pink" text="red"
            alink="green" link="yellow">
    <a href="body.html">background</a>
       color of the page is pink. and text color is red
 </BODY>
</HTML>
```



Formatting tags

- Bold Font: …
- Italic: …
- Underline: <u>…</u>
- Strikethrough: <strike> or <s>
- Subscript: <sub>
- Superscript: <sup>

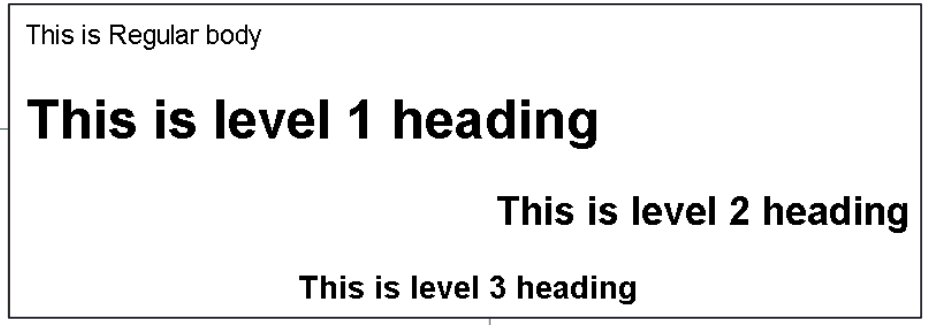



```
<b>This is in bold font</b><br>
<i>This text is in Italics</i><br>
<u>This text is Underlined</u><br>
<small>This is small text.</small><br>
<font size=7>This text is very large</font><br>
<font color=”Blue”>This is Blue Text.</font><br>
<strike>This is strikethrough style text</strike><br>
The chemical formula of water is h<sub>2</sub>o.<br>
A simple formula for a parabola is y = x<sup>2</sup>.<br>
Applying some <mark>markup</mark> here<br>
Applying <em>emphasis</em><br>
<tt>teletype text</tt>
```

Layout tags

- Heading Styles:
  - **<hn>…….</hn>**
    - Value of n can range from **1 to 6**
  - <h1 align="center">This is level 1 heading</h1>

```
<html>
<head>
   <title>Header Demo</title>
</head>
<body>
   This is Regular body
   <h1 align="left">This is level 1 heading</h1>
   <h2 align="right">This is level 2 heading</h2>
   <h3 align="center">This is level 3 heading</h3>
</body>
</html>
```



Special Characters in HTML

- Character Entities
  - Comprise following parts:
    - Ampersand (&),
    - Entity name or a #
    - Character code
    - Semicolon (;)
  - Included in HTML page using:
    - To display “>” symbol, use character code 62 i.e. &#62 or >
    - For space, use

Horizontal Lines in a Web Page

- <hr> - horizontal Rule. Attributes:
  - Size: Line thickness <hr size="5">
  - Width: Line width either in pixels or % of browser window
    - <hr width="100"> or <hr width="60%">
  - Align: Alignment values can be left, center or right <hr align="center">
  - Color: to display colored horizontal lines. Eg <hr color="red">

```
<body> <p>
      This paragraph contains a lot of lines
      in the source code,
      but the browser ignores it.  </p>
      <hr size="2" width="50%" color="blue" align = "left" >
      <p>Notice the horizontal rule occupying 50 % of the window width.
      <p>This paragraph contains <br> line breaks in the
      source code <br>
      so this  is the third line displayed  within the paragraph.
</body>
```

This paragraph contains a lot of lines in the source code, but the browser ignores it.

Notice the horizontal rule occupying 50 % of the window width.

This paragraph contains
line breaks in the source code
so this is the third line displayed within the paragraph.

Numbered List (Ordered List)

- Automatically generate numbers in front of each item in the list
  - Number placed against an item depends on the location of the item in the list.
  - Example:

```
<body>
   <ol>
     <li>INDIA</li>
     <li>SRILANKA</li>
   </ol>
</body>
```

1. INDIA
2. SRILANKA

  - To start an ordered list at a number other than 1, use **START** attribute:

Example : <ol start=11>

11. INDIA
12. SRILANKA

- Select the type of numbering system with the **type** attribute. Example:
  - A-Uppercase letters. <OL TYPE=A>
  - a-Lowercase letters. <OL TYPE=a>
  - I-Uppercase Roman letters
  - i-Lowercase Roman letters
  - 1-Standard numbers, default

```
<ol type="a">
   <li>INDIA</li>
   <li>SRILANKA</li>
</ol>
```

a. INDIA
b. SRILANKA

Bulleted List (Unordered List)

- Example:

```
<ul>
   <li>Latte</li>
   <li>Expresso</li>
   <li>Mocha</li>
</ul>
```

- Latte
- Expresso
- Mocha

- To change the type of bullet used throughout the list, place the TYPE attribute in the <UL> tag at the beginning of the list.
  - DISC (default) gives bullet in disc form.
  - SQUARE gives bullet in square form.
  - CIRCLE gives bullet in circle form.

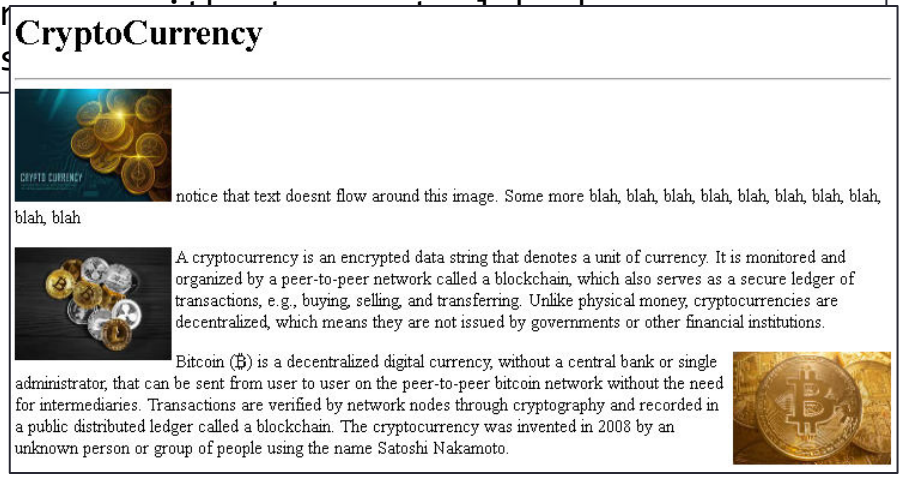
```
<ul>
   <li type='disc'>Latte</li>
   <li type='square'>Expresso</li>
   <li type='circle'>Mocha</li>
</ul>
```

- Latte
- Expresso
- Mocha

Adding Image

- Images are added into a document using <img> tag.
- Attributes :
  - alt: holds a text description of the image; its optional, but is incredibly useful for accessibility - screen readers read this description out to their users so they know what the image means
  - align: Image horizontal alignment; and also flow of text around image. Valid values: left, right
  - width/Height: Sets the width and height of the image.
  - src: Specifies the image path or source.

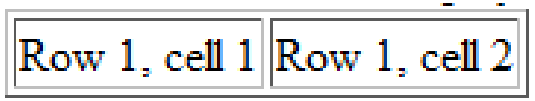
```
<h1>CryptoCurrency</h1><hr>
<img src="crypto.jfif" height="100" width="140"> notice that text doesnt flow aro ...
<p><img src="bitcoin2.jfif" align="left" height="100" width="140">
<p>A cryptocurrency is an encrypted data string that denotes a unit of currency.
 ...
<p><img src="bitcoin1.jfif" align="right" height="100" width="140">
<p>Bitcoin (₿) is a decentralized digital curr
single administrator, that can be sent from us
```



Table

- Use <table> tag to create a table.
- Table Attributes:
  - Border: applies border to table. Eg : <table border="2">……</table>
  - Align: defines the horizontal alignment of the table element.
    - Values of the align attribute are right, left and center. <table align="center">
  - Width: defines the width of the table element.
    - <table width="75%">……</table>
    - <table width="400">……</table>
  - Example:

```
<table border="1">
   <tr>
     <td>Row-1, cell-1</td>
     <td>Row-1, cell-2</td>
   </tr>
</table>
```



- To add a caption to a table, use the <caption> tag:
  - The <caption> tag must be inserted immediately after the <table> tag

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr><th>Month</th><th>Savings</th></tr>
  …
</table>
```

Table Data

- An HTML table has two kinds of cells:
  - Header Cells <th>: Contain header information; The text is bold and centered.
  - Standard Cells <td>: Contain data; The text is regular and left-aligned.

```
<table>
   <tr> <th>Column1 Header</th> <th>Column2 Header</th></tr>
   <tr> <td>Cell 1,1</td> <td>Cell 1,2</td> </tr>
   <tr> <td>Cell 2,1</td> <td>Cell 2,2</td> </tr>
</table>
```

Column1 Header	Column2 Header
Cell 1,1	Cell 1,2
Cell 2,1	Cell 2,2

- You can insert a bgcolor attribute in a <table>, <td>, <th> or <tr> tag to set the color of the particular element.

```
<table bgcolor="cyan">
   <tr bgcolor="blue">
       <th bgcolor="red">Header 1</th><th>Header 2</th>
   </tr>
   <tr>
      <td bgcolor="green">data 1</td><td>data 2</td>
   </tr>
</table>
```

Header 1	Header 2
data 1	data 2

Cell Spanning

- colspan="number of columns"
  - By default, the number of columns in a table is defined by the number of table data cells appearing in the table row that contains the most data.
  - Ideally, place the same number of data cells in each table row. If a table row does not contain the requisite number of table cells, then it will essentially be in 'error' and will be displayed with a missing cell.
- rowspan="number of rows"
  - Forces a table cell to span the number of rows specified by the given value.

```
<table border=1>
      <tr>
          <th>Column 1</th><th>Column 2</th><th>Column 3</th>
      </tr>
      <tr>
         <td rowspan=2>Column 1 Data</td>
         <td align=center colspan=2>Col 2, Row 1 Data</td>
      </tr>
      <tr>
         <td>Col 2, Row 2 Data</td><td>Col 3, Row 2 Data</td>
      </tr>
</table>
```

Column 1	Column 2	Column 3
Column 1 Data	Col 2, Row 1 Data	
	Col 2, Row 2 Data	Col 3, Row 2 Data

Hyperlink

- Hyperlinks access resources on the internet.
- Create a link with <a href=“”> (anchor)

- Login Here
- Hello, Welcome to My Site
- I have some older information about this subject.

Login Here
Hello, Welcome to My Site
I have some older information about this subject.

```
<ul>
   <li><a href='home.html'>mumbai</a></li>
   <li><a href='home.html'>pune</a></li>
   <li><a href='home.html'>nasik</a></li>
</ul>
```

- mumbai
- pune
- nasik

```
<table border=1>
  <tr><th>team<th>points<th>grade</tr>
  <tr> <td><a href='home.html'>mumbai</a></td><td>90</td><td>a</td> </tr>
  <tr> <td><a href='home.html'>pune</a></td><td>86</td><td>b</td>  </tr>
  <tr> <td><a href='home.html'>nasik</a></td><td>80</td><td>c</td>  </tr>
</table>
```

team	points	grade
mumbai	90	a
pune	86	b
nasik	80	c

Use of Image as a Hyperlink

- Images used as hyperlinks:
  - <A href="image1.html"><img src="home.gif">
- Images contained within a table:

```
<table align="center" border="2">
   <tr><th>Product</th><th>Cost</th><th>Image</th></tr>
   <tr>
      <td>Pencil</td>
      <td>$8</td>
      <td><img src="pencil.gif"></td>
   </tr>
   <tr>
      <td>Brush</td>
      <td>$15</td>
      <td><img src="paintbrush.gif"></td>
   </tr>
   <tr>
      <td>Pin</td>
      <td>$3</td>
      <td><img src="pin.gif"></td>
   </tr>
</table>
```

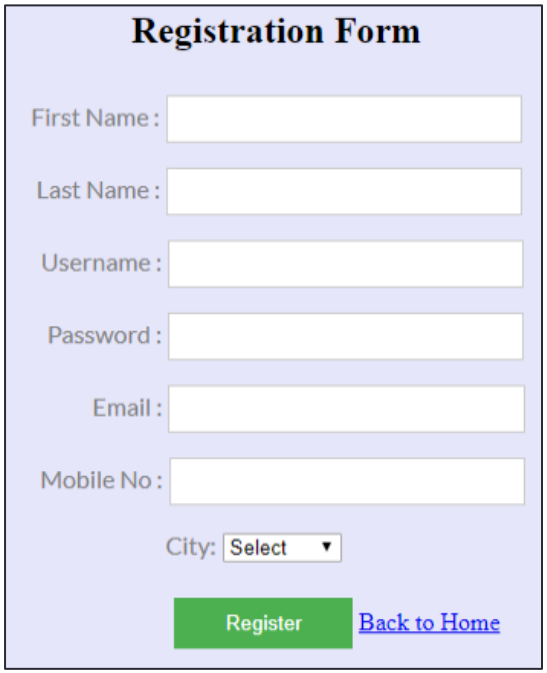
Product	Cost	Image
Pencil	$8	
Brush	$15	
Pin	$3	

HTML Forms for User Input

- Data Submission using a Form
  - HTML forms are used to accept of user input.
  - A form contains form elements.
  - Form elements are elements that allow users to enter information in a form.
  - Define a form with the <form> tag.</form>

```
<form method="get/post" action="URL">
        Field definitions
</form>
```

- <input> tag is used to create form input fields.
  - Type attribute of <input> tag specifies the field type
    - Single line text box <input type="text">
    - Password field <input type="password">
    - Hidden field <input type="hidden">
    - Radio button <input type="radio">
    - Checkbox <input type="checkbox">
    - File selector dialog box <input type="file">
    - Button <input type="button">
    - Submit/Reset <input type="submit/reset">
- <textarea>
- <select>
- <button>



Form elements

- Text fields: **Single Line** : used to type letters, numbers, etc. in a form.
  - <INPUT TYPE=”type” NAME=”name” SIZE=”number” VALUE=”value” maxlength=n>
  - Eg:

```
<form>
      First name: <input type="text" name="firstname“ value=“fname>
      Last name:<input type="text" name="lname">
</form>
```

- Text Area (Multiple Line Text Field)
  - A text area can hold an unlimited number of characters. Text renders in a fixed-width font (usually Courier).
  - You can specify text area size with cols and rows attributes.

```
<textarea name=“name” rows=“10” cols=“50” [disabled] [readonly]>
            Default-Text
</textarea>
```

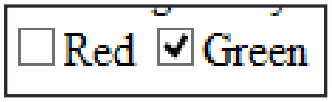
```
<textarea name=“address” rows=5 cols=10>
                Please write your address
</textarea>
<textarea rows=“4” cols=“20”>
```

- Password
  - <input type=“password” name=“name” size=n value=“value” [disabled]>
  - Eg: Enter the password:<input type=“password” name=“passwd” size=20 value=“abc”>

Form elements

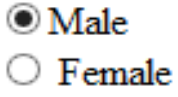
- Check box - Lets you select one or more options from a limited number of choices.
  <input type="checkbox" name="name" value="value" [checked] [disabled]> Checkbox Label
  - Content of value attribute is sent to the form's action URL.

```
<input type="checkbox" name="color1" value="0"/>Red
<input type="checkbox" name="color3" value="1" checked/>Green
```



- Radio Buttons
  - <input type="radio" name="name" value="value" [checked] [disabled]> Radio Button Label
  - *Content of the value attribute is sent to the form's action URL*.

```
<form>
<input type="radio" name="gender" value="male"/>Male<br>
<input type="radio" name="gender" value="female"/> Female
</form>
```



- Hidden form field - Allows to pass information between forms:
  - <input type="hidden" name="name" size="n" value="value"/>
  - <input type="hidden" name="valid_user" size="5" value="yes"/>

Form elements

- File Selector Dialog Box
  - <input type="file" name="name" size="width of field" value="value">

```
<FORM>
  Select file to upload:
   <INPUT name="file" type="file"> <BR>
   <INPUT type="submit" >
</FORM>
```

- Buttons:
  - To add a button to a form use:
  - <input type="button" name="btnCalculate" value="Calculate"/>
  - To submit the contents of the form use:
  - <input type="submit" name="btnSubmit" value="Submit"/>
  - To reset the field contents use:
  - <input type="reset" name="btnReset" value="Reset"/>

  - You can also create button using <button> tag ; defines a push button. Inside this element you can put content, such as text or images.
  - <button type="button">Click Me!</button>

Drop-Down List

<select name="name" multiple="true/false" size=n [disabled]>
<option [selected] [disabled] [value]>Option 1</option>…
</select>

- Multiple: States if multiple element selection is allowed.
- Size: Number of visible elements.
- Disabled: States if the option is to be disabled after it first loads.

```
<form>
<select multiple size="3" name="pref">
  <option value="ih"   selected>Internet-HTML</option>
  <option value="js">Javascript</option>
  <option   value="vbs">VBscript</option>
  <option value="as">ASP</option>
</select>
</form>
```

- Forms with labels

```
<form>
  <label for="uname">User name : </label>
  <input id="uname" name="username">
  <button>Submit</button>
</form>
```

<fieldset> and <legend>

- Group related elements in a form

```
<form>
 <fieldset>
   <legend>Personal Details:</legend>
    Name: <input type="text"><br><br>
    Email: <input type="text"><br>
  </fieldset>
</form>
```

Personal Details:

Name:

Email:

Your details

1. Name First and last name
2. Email example@domain.com
3. Phone Eg. +447500000000

Delivery address

1. Address
2. Post code
3. Country

Card details

1. Card type
   1. VISA
   2. AmEx
   3. Mastercard
2. Card number
3. Security code
4. Name on card Exact name as on the card

Formatting tags

Tag	Example	Results
	Bold Text	An example of **Bold Text**
<big>	<big>Big Text</big>	An example of Big Text
	Center Text	An example of Center Text
	Emphasized Text	An example of *Emphasized Text*
	Italic Text	An example of *Italic Text*
<small>	Small Text	An example of Small Text
	Strong Text	An example of **Strong Text**
<sub>	_{Subscript Text}	An example of Subscript Text
<sup>	^{Superscript Text}	An example of Superscript Text
<del> <s> <strike>	~~Delete Text~~ <s>Strike Text</s> <strike>Strike Text</strike>	An example of ~~Delete Text~~ An example of ~~Strike Text~~ An example of ~~Strike Text~~
<u>	<u>Underline Text</u>	An example of Underline Text
<tt>	<tt>Teletype Text</tt>	An example of `Teletype Text`

HTML5

What's new in HTML5?

- HTML5 offers new enhanced set of tags
  - New Content Tags : <nav>,<section>,<header>,<article>,<aside>,<summary>
  - New Media Tags : <video>,<audio>
  - New Dynamic drawing : <canvas>graphic tag
  - New form controls, like calendar, date, time, email, url, search

- Support for JavaScript APIs
  - Canvas element for 2D drawing API
  - Video and audio APIs
  - APIs to support offline storages
  - The Drag & Drop APIs
  - The Geolocation API
  - Web workers, WebSQL etc

- The DOCTYPE tells the browser which type and version of document to expect.
  - The DOCTYPE announcement is not a HTML label; it is a guideline to the web program about what variant of HTML the page is composed in.

```
<!DOCTYPE  html>
```

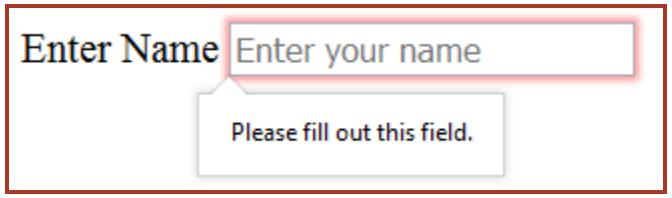
HTML5 Attributes for <input>

- A Form is one of the most basic and essential feature of any web site
  - HTML5 brings 13 new input types and 14 new attributes
  - HTML5 introduces these data types via the <input type="_NEW_TYPE_HERE_"/> format

- **Placeholder** - A placeholder is a textbox that hold a text in lighter shade when there is no value and not focused
  - <input id="first_name" placeholder="This is a placeholder">
  - Once the textbox gets focus, the text goes off and you shall input your own text

- **AutoFocus** - Autofocus is a Boolean attribute of form field that make browser set focus on it when a page is loaded
  - <input id ="Text2" type="text" autofocus/>

- **Required** - A "Required Field" is a field that must be filled in with value before submission of a form
- <input name="name" type="text" required />



New Form elements

- **Email** - Checks whether the string entered by the user is valid email id or not.
  - <input id="email" name="email" type="email" />
- **Search** - used for search fields (behaves like a regular text field).
  - <input id="mysearch" type="search" />
- **Tel** - used for input fields that should contain a telephone number.
  - <input type="tel" name="usrtel">
  - <input type="tel" name="phone“ pattern**="[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}“** title="North American format: XXX-XXX-XXXX">
- **url** - is used for input fields that should contain a URL address.
  - Depending on browser support, the url field can be automatically validated when submitted.
- **color** – displays a color palette
- **Number** - used for input fields that should contain a numeric value.
  - Min and max parameters provided to limit the values.
  - Browser will treat it as simple textfield if it doesn’t support this type.
  - <input id="movie" type="number" value="0"/>
  - <input type="number“ min="0“ max="50“ step="2“ value="6">

New Form elements

- **Range** - used for input fields that should contain a value within a range
  - Browser will treat it as simple textfield if it doesn't support this type
  - <input id="test" type="range"/>
  - <input type="range" min="1" max="20" value="0">

- **Date** - used for input fields that should contain a date.
  - Depending on browser support, a date picker can show up in the input field.
  - <input id="meeting" type="date" />

- **month** - Selects month and year
- **week** - Selects week and year
- **time** - Selects time (hour and minute)
- **datetime** - Selects time, date, month and year
- **datetime-local** - Selects time, date, month and year (local time)

```
•<input type="button">
•<input type="checkbox">
•<input type="color">
•<input type="date">
•<input type="datetime-local">
•<input type="email">
•<input type="file">
•<input type="hidden">
•<input type="image">
•<input type="month">
•<input type="number">
•<input type="password">
•<input type="radio">
•<input type="range">
•<input type="reset">
•<input type="search">
•<input type="submit">
•<input type="tel">
•<input type="text">
•<input type="time">
•<input type="url">
•<input type="week">
```

Built-in form validation

- One of the most significant features of HTML5 form controls is the ability to validate most user data without relying on JavaScript.
  - This is done by using validation attributes on form elements like required, max, min, pattern, maxlength, minlength
  - If the data entered in a form field follows all of the rules specified by the above attributes, it is considered valid. If not, it is considered invalid.

```
<form>
    <label for="uname">User Name</label>
    <input id="uname" name="uname" required pattern="[a-zA-Z]*">
    <button>Submit</button>
  </form>
```

Audio and video

- Until now, there has never been a standard for playing audio on a web page.
  - Today, most audio is played through a audio plugin (like Microsoft Windows Media player, Microsoft Silverlight ,Apple QuickTime and the famous Adobe Flash).
  - However, not all browsers have the same plugins.
  - HTML5 the audio element to play sound files, or an audio stream.
  - Other properties like auto play, loop, preload area also available

```
<audio controls>
  <source src="vincent.mp3" type="audio/mpeg"/>
  <source src="vincent.ogg" type="audio/ogg"/>
</audio>
```

- Today, most videos are shown through a plugin (like Flash). However, not all browsers have the same plugins.
  - HTML5 provides <video> element to include video
  - Supported video formats for the video element : Ogg, MP4, WebM, .flv, .avi
  - Attributes : width, height, poster, autoplay, controls, loop, src

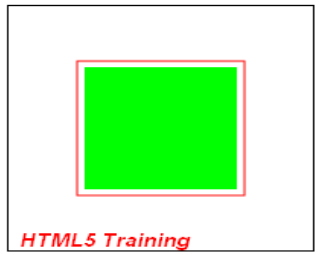
```
<video controls="controls" width="640" height="480" src="bunny.mp4" />
  Your browser does not support the video element.
</video>
```

Canvas

- A canvas is a rectangle in your web page within which you can use JavaScript to draw shapes
  - Canvas can be used to represent something visually in your browser like Simple Diagrams, Fancy user interfaces, Animations, Charts and graphs, Embedded drawing applications, Working around CSS limitations
  - The canvas element has several methods for drawing paths, boxes, circles, characters, and adding images.
  - The canvas element has no drawing abilities of its own. All drawing must be done inside a JavaScript

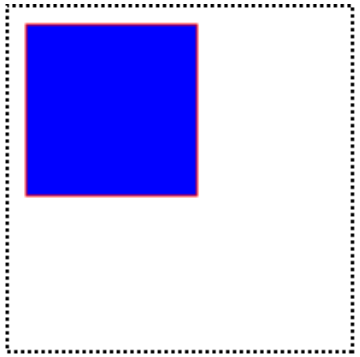
```
<canvas id="myCanvas" width="200" height="100">
</canvas>
```

```
<canvas id="myCanvas"></canvas>
<script type="text/javascript">
  var canvas=document.getElementById('myCanvas');
  var ctx=canvas.getContext('2d');
  ctx.fillStyle='#FF0000';
  ctx.fillRect(0,0,80,100);
</script>
```

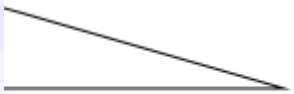


Canvas examples

```
<script>
function draw(){
    var canvas=document.getElementById('myCanvas');
    var context=canvas.getContext('2d');
    context.strokeStyle = "red";
    context.fillStyle = "blue";
    context.fillRect(10,10,100,100);
    context.strokeRect(10,10,100,100);
}
</script>
```



```
var ctx=c.getContext("2d");
ctx.moveTo(10,10);
ctx.lineTo(150,50);
ctx.lineTo(10,50);
ctx.stroke();
```

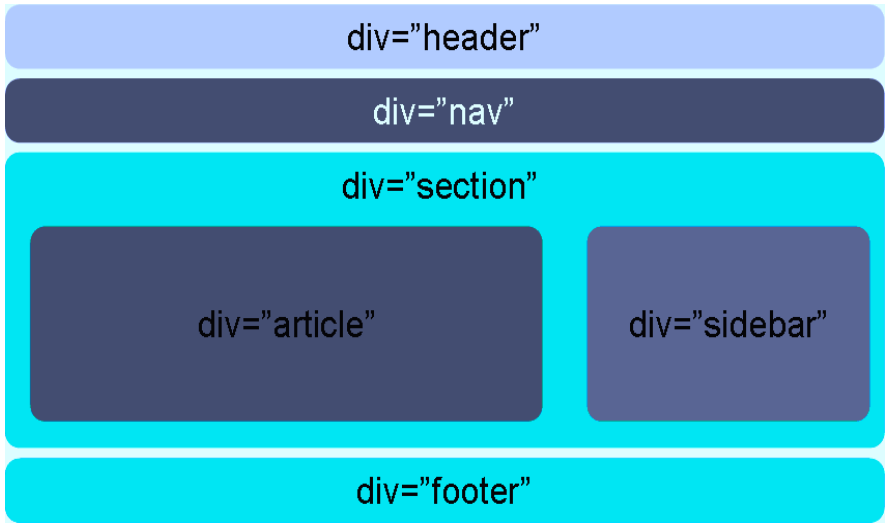


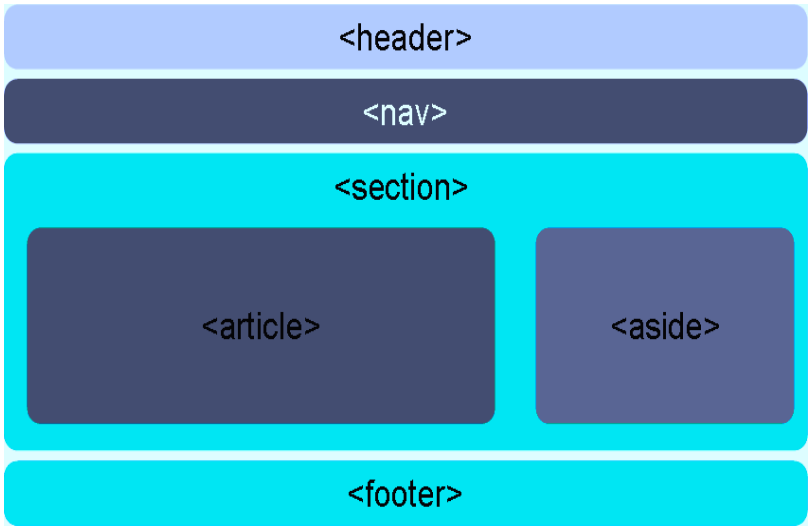
```
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.beginPath();
ctx.arc(70,18,15,0,Math.PI*2,true);
ctx.closePath();
ctx.fill();
```

```
var ctx=c.getContext("2d");
var grd=ctx.createLinearGradient(0,0,175,50);
grd.addColorStop(0,"#FF0000");
grd.addColorStop(1,"#00FF00");
ctx.fillStyle=grd;
ctx.fillRect(0,0,175,50);
```

Laying out a page with HTML5

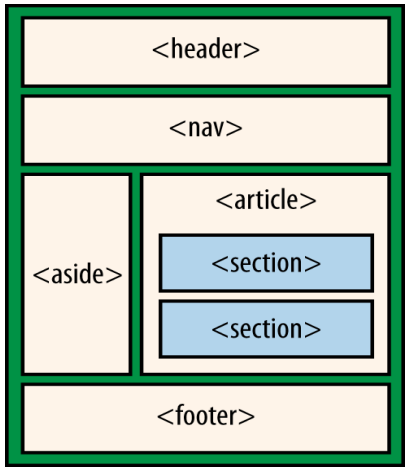
- Most HTML 4 pages include a variety of common structures, such as headers, footers and columns
- It's common to mark them up using div elements, giving each a descriptive id or class
- HTML 5 addresses this issue by introducing new elements for representing each of these different sections
- Elements that make it much easier to structure pages

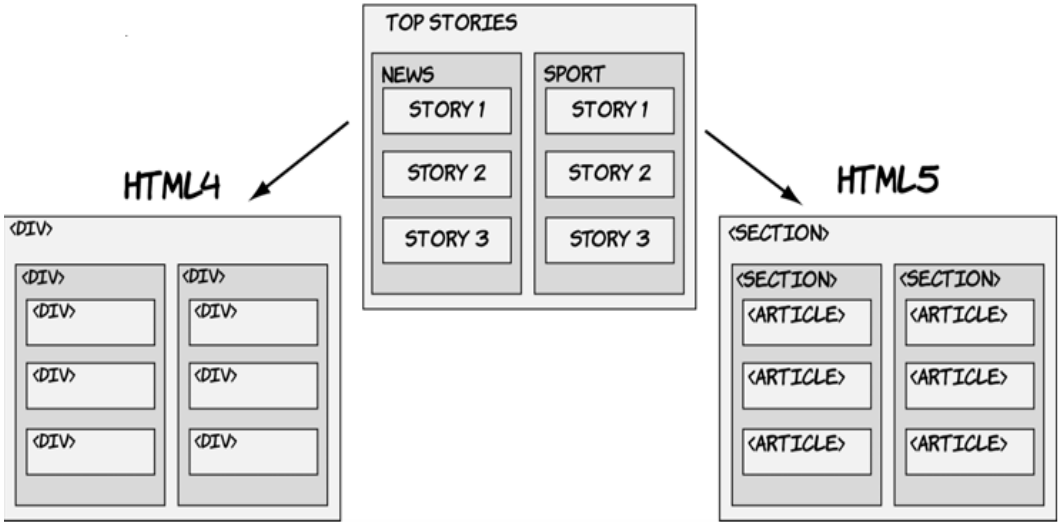




New Semantic Elements

- **<section> :** can be used to thematically group content, typically with a heading.

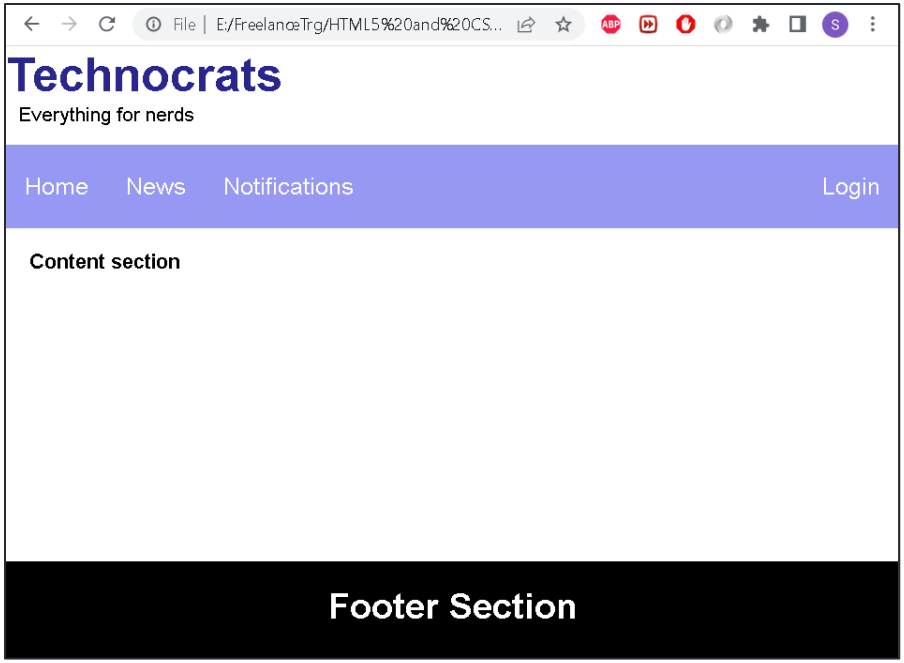




- **<article>:** element represents a self-contained composition in a document, page, application, or site that is intended to be independently distributable or reusable
  - Eg a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment

- **<nav>**: Represents a major navigation block. It groups links to other pages or to parts of the current page.

```html
<body>
    <!-- Header Section -->
    <header>
        <div class="head1">
            Technocrats
        </div>
        <div class="head2">
            Everything for nerds
        </div>
    </header>
    <!-- Menu Navigation Bar -->
    <nav class="menu">
        <a href="#home">Home</a>
        <a href="#news">News</a>
        <a href="#notification">
            Notifications
        </a>
        <div class="menu-log">
            <a href="#login">Login</a>
        </div>
    </nav>
    <!-- Body section -->
    <main class="body_sec">
        <section id="Content">
            <h3>Content section</h3>
        </section>
    </main>
    <!-- Footer Section -->
    <footer>Footer Section</footer>
</body>
```



blocking elements2.html

New Semantic Elements

- **<Header>:** tag specifies a header for a document or section. Can also be used as a heading of an blog entry or news article as every article has its title and published date and time

- **<aside>:** The "aside" element is a section that somehow related to main content, but it can be separate from that content header and footer element in an article.
- **<footer>:** Similarly to "header" element, "footer" element is often referred to the footer of a web page.
  - However, you can have a footer in every section, or every article too

- **<figure>:** The <figure> tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
  - This element can optionally contain a figcaption element to denote a caption for the figure.

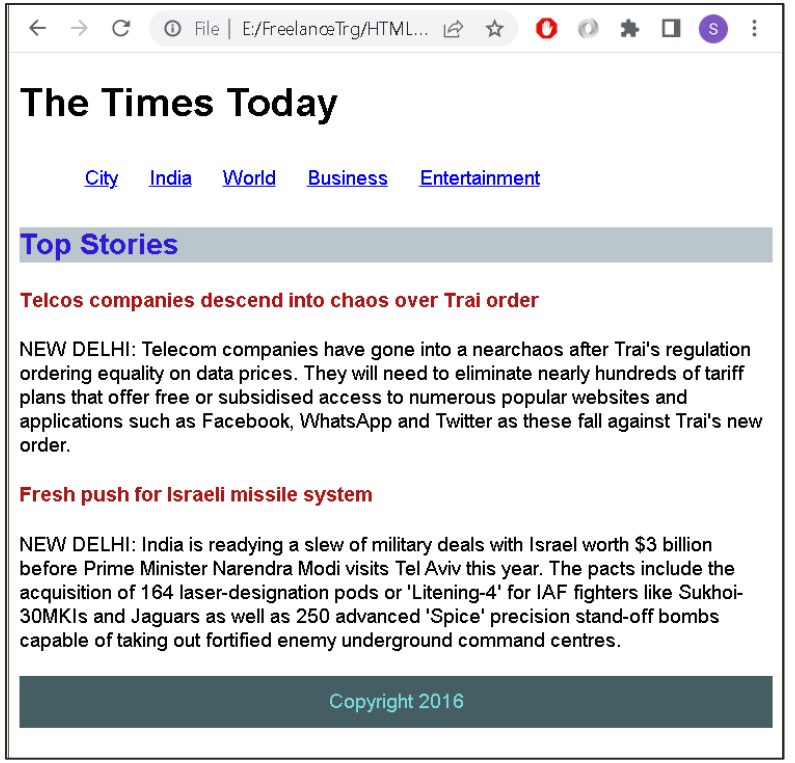
```
<figure>
  <img src="fruits.png" alt="Picture of the fave fruits" height="100" width="100">
  <figcaption>Picture of the fave fruits</figcaption>
</figure>
```

Picture of the fave fruits

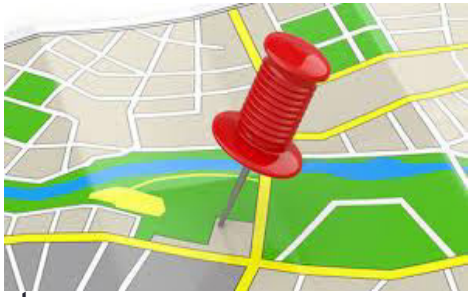
sectionarticle-eg

```
<body>
  <header><h1>The Times Today</h1>
  <nav>
      <ul>
          <li><a href="#">City</a></li>
          <li><a href="#">India</a></li>
          <li><a href="#">World</a></li>
          <li><a href="#">Business</a></li>
          <li><a href="#">Entertainment</a></li>
      </ul>
  </nav>
  </header>

  <section>
      <header style = "background-color: #607d8b70;">
          <h1 style = "color: #2a13dbf6;">Top Stories</h1>
      </header>

      <article>
          <h4 style = "color: #a80f0ff6;">Telcos companies de
              NEW DELHI: Telecom companies have gone into a nea
      </article>

      <article>
          <h4 style = "color: #a80f0ff6;">Fresh push for Isra
              NEW DELHI: India is readying a slew of military d
              The pacts include the acquisition of 164 laser-de
      </article>
  <br>
      <footer class="footer"> Copyright 2016</footer>
  </section>
```


```
<style>
    li{
      display: inline-flex;
      padding: 10px
    }
    .footer{
      height: 30px;
      color: rgb(118, 216, 219);
      background-color: #455e64;
      text-align: center;
      padding-top: 10px;
    }
  </style>
```

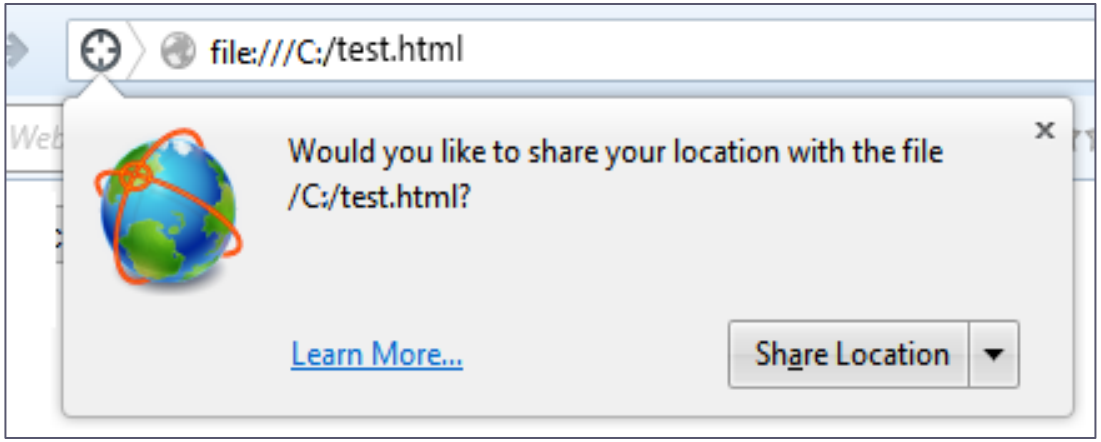


Geo Location API



- geolocation is best described as the determination of the geographic position of a person, place, or thing
- Geolocation API is used to locate a user's position
  - It also keeps the track of as they move around, always with the user's consent
  - The API is device-agnostic; it doesn't care how the browser determines location, so long as clients can request and receive location data in a standard way
    - The underlying mechanism might be via GPS, wifi, or simply asking the user to enter their location manually
    - Since any of these lookups is going to take some time, the API is asynchronous; you pass it a callback method whenever you request a location

- Since the nature of the API exposes the user's location, it could compromise their privacy.
- So the user's permission to attempt to obtain the geolocation information must be sought before proceeding



Using the Geolocation API

- Test for the presence of the geolocation object:

```
if (navigator.geolocation) // check for Geolocation support
        console.log('Geolocation is supported!');
 else   console.log('Geolocation is not supported for this Browser/OS version yet.');
```

- Obtain the geolocation object

```
var geolocation = navigator.geolocation;
```

- Geolocation Methods
  - getCurrentPosition() : retrieves the current geographic location of the user.
  - watchPosition() : retrieves periodic updates about the current geographic location of the device.
  - clearWatch() : cancels an ongoing watchPosition call.

```
function getLocation() {
   var geolocation = navigator.geolocation;
   geolocation.getCurrentPosition(showLocation, errorHandler);
}
```

showLocation and errorHandler are callback methods which would be used to get actual position

Using the Geolocation API

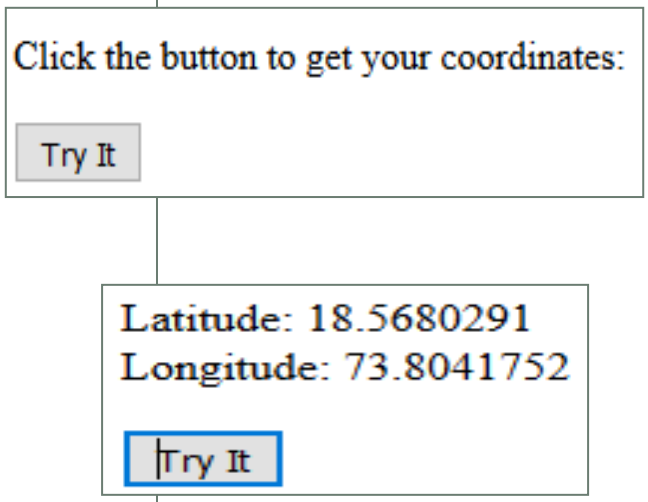
- getCurrentPosition() method is called asynchronously with an object **Position** which stores the complete location information.
  - The **Position** object specifies the current geographic location of the device

```
function showLocation( position ) {
   var latitude = position.coords.latitude;
   var longitude = position.coords.longitude;
   ...
}
```

```
function errorHandler( err ) {
   if (err.code == 1) {
      // access is denied
   }
   ...
}
```

We need to catch any error and handle it gracefully

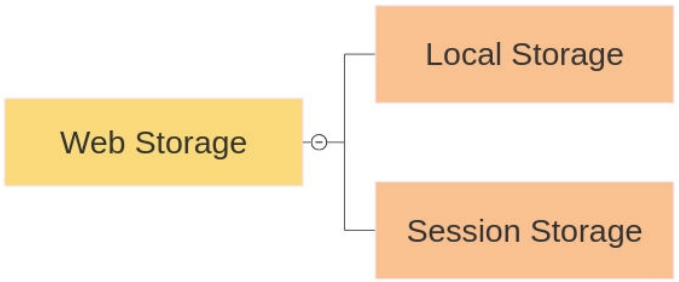
```
<body>
<p id="demo">Click the button to get your coordinates:</p>
<button onclick="getLocation()">Try It</button>
<script>
var x=document.getElementById("demo");
function getLocation()  {
    if (navigator.geolocation)
        navigator.geolocation.getCurrentPosition(showPosition);
    else{
        x.innerHTML="Geolocation is not supported by this browser.";
 }
function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
                  "<br />Longitude: " + position.coords.longitude;
 }
</script>
```



Web Storage

- With HTML5, web pages can store data on the client-side locally within the user's browser and operate offline more efficiently.
  - In earlier versions of HTML, cookies were used to store data locally.
  - Web Storage is more secure and faster since data is used only when requested by the server.
  - Large amounts of data can be stored without affecting the web page's performance.
  - Web storage allows us to store simple key/value data in the browser.
  - Web storage is similar to cookies, but better implemented and we can store greater amounts of data; up to 5MB of simple data created by our websites or web applications.

- There are two kinds of web storage

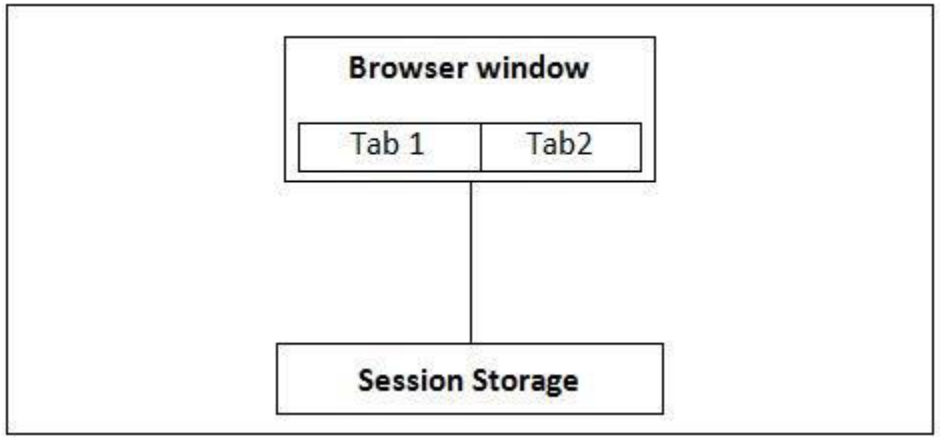


stores data with no expiration date .Used for example for user-authentication on the client side

stores data for one session only, intended to be used in a single transaction scenarios and as you close the window, session would be lost. I.e. the data is deleted when the browser is closed.

Session Storage

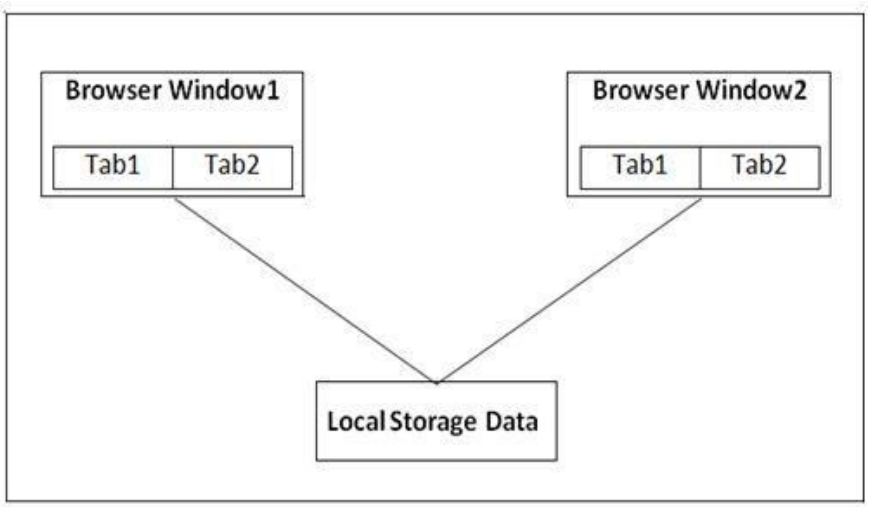
- SessionStorage is only available within the browser tab or window session.
  - It's designed to keep track of data specific to one window or tab.
- It allows us to isolate information in each window.
  - Even if the user is visiting the same site in two windows, each window will have its own individual session storage object and thus have separate, distinct data.
  - Session storage is not persistent—it only lasts for the duration of a user's session on a specific site (in other words, for the time that a browser window or tab is open and viewing that site).
  - Data is stored in KEY / VALUE pair of strings



Web Storage

- Local Storage
  - stores data with no expiration date to the user's computer.
  - localStorage is kept even between browser sessions.
    - This means data is still available when the browser is closed and reopened, and also instantly between tabs and windows.
  - The data persists until the user manually clears the browser cache or programmatically clears the storage.
  - Data is stored in KEY / VALUE pair of strings

  - Web Storage data is, in both cases, **not** available between different browsers.
    - For example, storage objects created in Firefox cannot be accessed in IE (just like cookies)

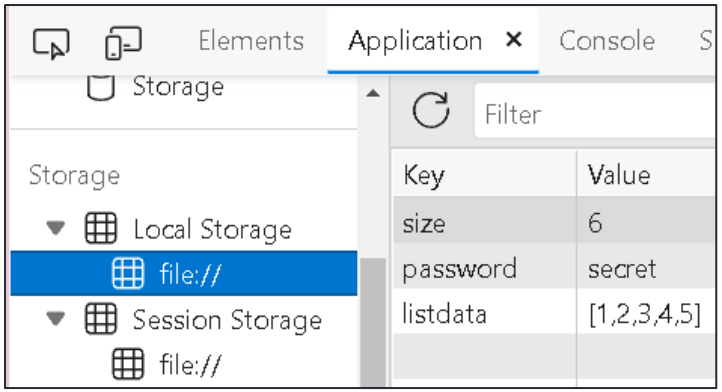


Web storage API

Some Common method and properties for local and session storage :

Name	Description	Returns
setItem(<key>,<value>)	Add a new Key/value pair and update if key is already used	void
getItem(<key>)	Retrieves the value with the specified key	string
removeItem(<key>)	Removes the Key/value pair having specified key	string
clear()	Removes the stored key/value pairs	void
key(<index>)	Retrieves the key at the specified index	String
length	Returns the no. of stored Key/value pairs	Number

```
localStorage.setItem("size", "6");
localStorage.setItem("password", "secret");
var size = localStorage.getItem("size");
localStorage.removeItem("password")

//ShortCut :
localStorage["size"]="6";
localStorage.size="6";

var size = localStorage["size"];
localStorage.clear();
```

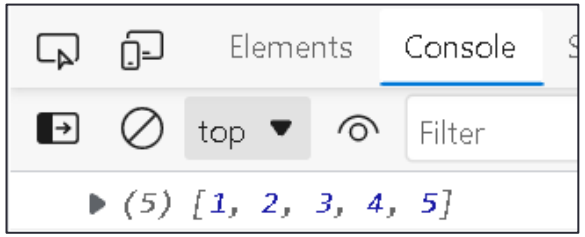


Web storage API

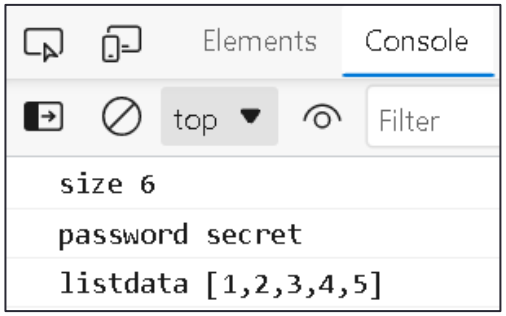
- You can only store strings via the session & local storage properties.
- To store JavaScript objects, convert them to a JSON string first.

```
var arr = [1, 2, 3, 4, 5];

// store array data to the localstorage
localStorage.setItem("listdata",  JSON.stringify(arr));

//Use JSON to retrieve the stored data and convert it
var storedData = localStorage.getItem("listdata");

if (storedData) {
  arrnew = JSON.parse(storedData);
  console.log(arrnew)
}
```

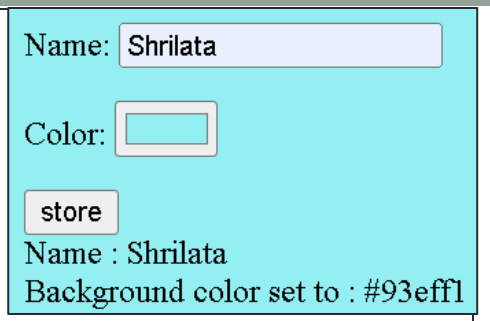
- To get all values stored in the Storage

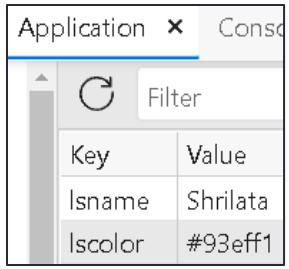



```
for (var i = 0; i <  localStorage.length ; i++) {
    var key = localStorage.key(i);
    var val = localStorage[key];
    console.log(key,val)
}
```



```
<!doctype html>
<html>
<head>
   <script>
       var name='', color='';
        function store(){
           name =  document.getElementById("name").value
           color = document.getElementById("color").value
           localStorage.setItem("lsname", name)
           localStorage.setItem("lscolor", color)
           setStyles()
        }
    function setStyles(){
       document.getElementById("body").style.backgroundColor=
                                          localStorage.getItem("lscolor")
       document.getElementById("s1").innerHTML = localStorage.getItem("lsname")
       document.getElementById("s2").innerHTML = localStorage.getItem("lscolor")
    }
  </script>
 </head>
```






```
<body id="body" onfocus="setStyles()">
<form>
   Name: <input type="text" id="name"><p>
   Color: <input type="color" id="color"><p></p>
   <input type="button" value="store" onclick="store()">
</form>
<div>Name : <span id="s1"></span></div>
<div>Background color set to : <span id="s2"></span></div>
</body>
</html>
```

```
<!doctype html>
<html>
       <head>
          <style>
               div{ background-color: wheat;
                border: 2px solid brown; padding: 5px; height: 30px;
                }
          </style>
       </head>
   <body id="body" onfocus="setStyles()">
     <h2>Click to count:</h2>
     <button onclick="counter();">Counter</button><br><br>
     <div id="output">Number of Clicks:</div>
     <script type="text/javascript">
       function counter() {
        if(sessionStorage.hits){
           sessionStorage.hits=Number(sessionStorage.hits)+1;
        }
        else{
          sessionStorage.hits=1;
        }
        document.getElementById('output').innerHTML=
         "The Counter button is clicked for"+ " "+ sessionStorage.hits +" "+"times.";
     }
     </script>
  </body>
</html>
```

(CSS) Cascading Style Sheets

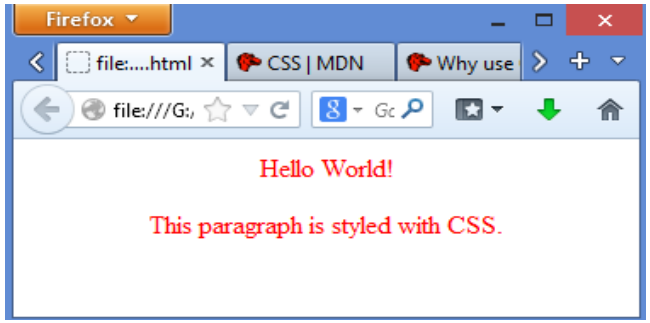
What is CSS?

- Cascading Styles Sheets - a way to style and present HTML.
  - HTML deals with content & structure, stylesheet deals with formatting & presentation of that document.
  - Allows to control the style and layout of multiple Web pages all at once.
- Why CSS?
  - saves time
  - Pages load faster
  - Easy maintenance
  - Superior styles to HTML

- A CSS rule has two parts: a selector, and one or more declarations:
- "HTML tag" { "CSS Property" : "Value" ; }
- The selector is normally the HTML element you want to style.
- Example:

```
<head>
<style>
p { color:red; text-align:center;   }
</style>
</head>
<body>
<p>Hello World!</p>
<p>This paragraph is styled with CSS.</p>
</body>
```

```
body{ background-color: gray;}
p { color: blue; }
h3{ color: white; }
```



Three Ways to Insert CSS

- Embedded Style Sheets
  - Style defined between <STYLE>..</STYLE> tags. <STYLE> tags appear either in <HEAD> section or between </HEAD> and <BODY> tags.

- Linked Style Sheets
  - Separate files (extn .CSS) that are linked to a page with the <LINK> tag. Are referenced with a URL. Placing a single <LINK> tag within the <HEAD> tags links the page that needs these styles.

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

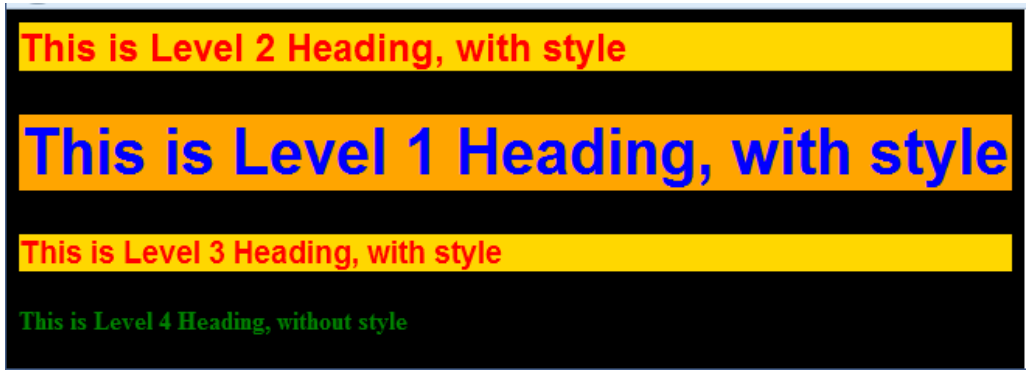
- Inline Style Sheets
  - Only applies to the tag contents that contain it. Used to control a single tag element. Tag inherits style from its parent. Egs.
  - This is a paragraph.
  - A new background and font color with inline CSS

Demo: Link Style Sheet

```
<html>
<head>
<link rel=stylesheet href="linked_ex.css"
type="text/css">
</head>
<body>
<h2>This is Level 2 Heading, with style</h2>
<h1>This is Level 1 Heading, with style</h1>
<h3>This is Level 3 Heading, with style</h3>
<h4>This is Level 4 Heading, without style</h4>
</body>
</html>
```

```
body {   background: black;
         color:green
       }
h1 {    background: orange;
        font-family: Arial, Impact;
        color: blue;
        font-size:30pt;
         text-align: center
    }
h2, h3 {   background: gold;
           font-family: Arial, Impact;
           color:red   }
```

linked_ex.css



Inline Style Sheet

- All style attribute are specified in the tag it self. It gives desired effect on that tag only. Doesn't affect any other HTML tag.

```
<body style="background: white; color:green">
<h2 style="background: gold; font-family: Arial, Impact; color:red">
This is Level 2 Heading, with style</h2>
<h1 style="background: orange; font-family: Arial, Impact; color: blue;font-size:30pt; text-align:
center">This is Level 1 Heading, with style</h1>
<h3 style="background: gold; font-family: Arial, Impact;color:red">
This is Level 3 Heading, with style</h3>
<h4>This is Level 4 Heading, without style</h4>
<h1>This is again Level 1 heading with default styles</h1>
</body>
```

Multiple Style Sheets

- If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet.
  - For example, an external style sheet has these properties for the h3 selector, & an internal style sheet has these:

```
H3 {  color:red;
      text-align:left;
       font-size:8pt;
   }
```

```
H3  {  text-align:right;
       font-size:20pt;
    }
```

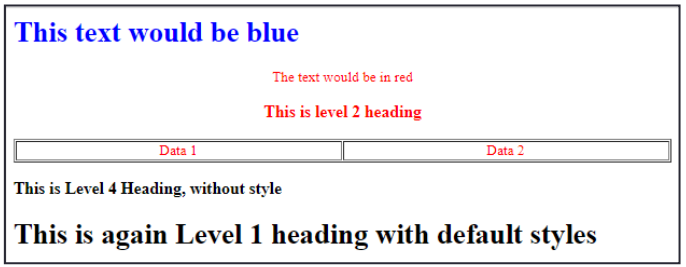
  - If the page with the internal style sheet also links to the external style sheet the properties for h3 will be:
  - color:red;
    text-align:right;
    font-size:20pt;

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

- Grouping Selectors
  - In style sheets there are often elements with the same style.
    - H1 { color:green; }
    - h2 { color:green; }
    - P { color:green; }
  - To minimize the code, you can group selectors by separating each selector with a comma. Example
    - h1,h2,p { color:green; }

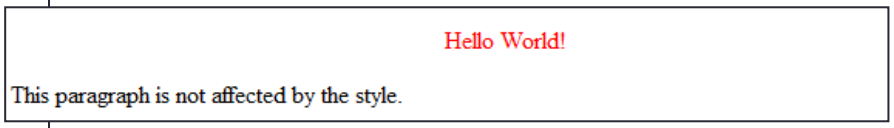
Types of selectors

- HTML selectors <tag>
  - Used to define styles associated to HTML tags. – already seen!!!!
- Class selectors (.)
  - Used to define styles that can be used without redefining plain HTML tags.
- ID selectors (#)
  - Used to define styles relating to objects with a unique id




```
<head>
   <style>
      H1.myClass {color: blue}
      .myOtherClass { color: red; text-align:center}
 </style>
 <body style="background: white; color:green">
    <H1 class="myClass">This text would be blue</H1>
    <p class="myOtherClass">The text would be in red</P>
    <H3 class="myOtherClass">This is level 2 heading</H3>
    <table class=myotherClass border width=100%>
        <td>Data 1</td><td>Data 2</td>
    </table>
    <h3>This is Level 4 Heading, without style</h3>
    <h1>This is again Level 1 heading with default styles</h1>
```

```
<head>
 <style>
   #para1 {
   text-align:center;
   color:red;
   }
 </style>
</head>
<body>
   <p id="para1">Hello World!</p>
   <p>This paragraph is not affected by the style.</p>
</body>
```



class Selector

- The class selector is used to specify a style for a group of elements.
  - The class selector uses the HTML class attribute, and is defined with a "."
  - Can also affect only specific HTML elements. Ex. all p elements with class="center" will be center-aligned (see below)

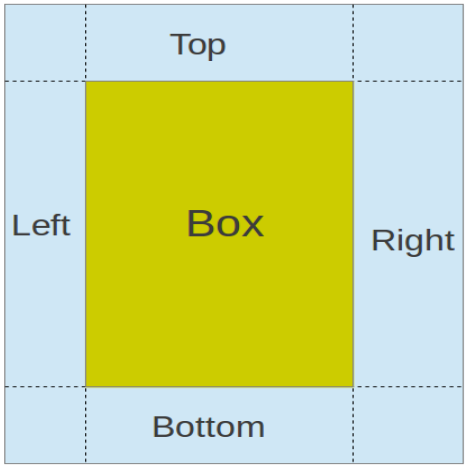
```
<head>
  <style>
     .center {  text-align:center; }
   </style>
</head>
<body>
  <h1 class="center">Center-aligned heading</h1>
  <p class="center">Center-aligned paragraph.</p>
</body>
```


```
<style>
   p.center { text-align:center; }
</style>
</head>
<body>
  <h1 class="center">This heading will not be affected</h1>
   <p class="center">This paragraph will be center-aligned.</p>
```

CSS Box Model

- All HTML elements can be considered as boxes.
- The CSS box model is essentially a box that wraps around HTML elements, and it consists of:
  - Margin - Clears an area around the border. The margin does not have a background color, it is completely transparent
  - Border - A border that goes around the padding and content. The border is affected by the background color of the box
  - Padding - Clears an area around the content. The padding is affected by the background color of the box
  - Content - The content of the box, where text and images appear

```
<h1>Header 1</h1>
<p> paragraph 1</p>
```



CSS Border

- The border property is a shorthand for the following individual border properties: border-width, border-style (required), border-color

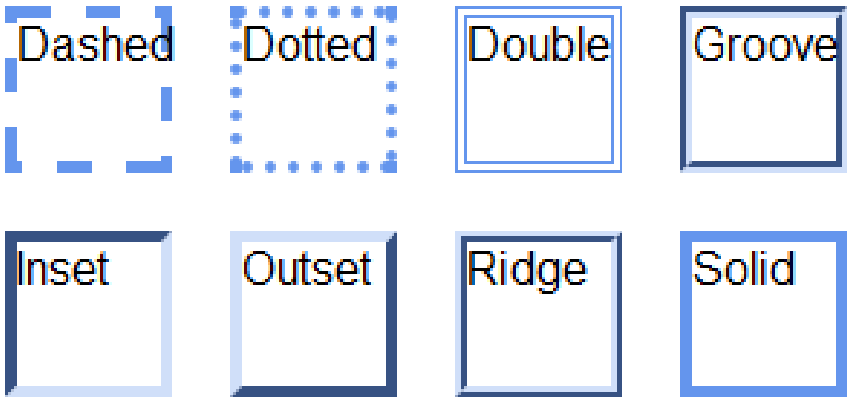
```
p { border: 5px  solid   red; }
```

This is some text in a paragraph.

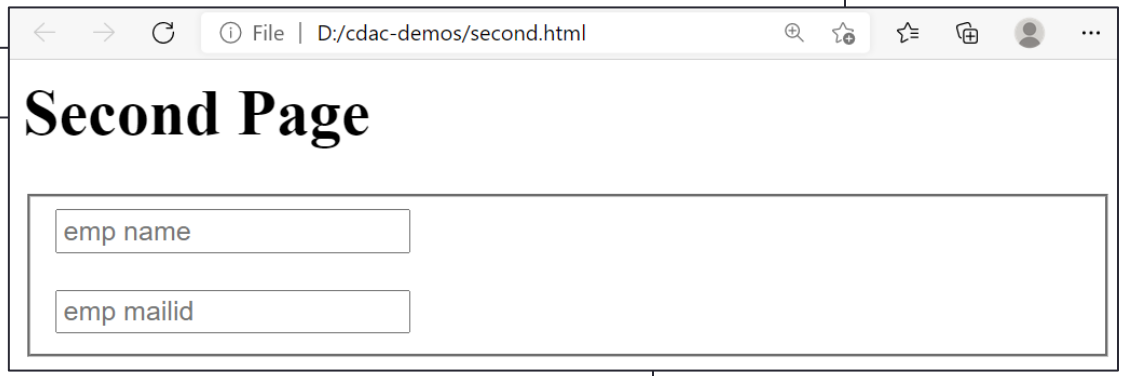
```
<style type="text/css">
.box {
    width: 100px;
    height: 100px;
    border-color: Blue;
    border-width: 2px;
    border-style: solid;
}
</style>

<div class="box">
        Hello, world!
</div>
```

Hello, world!

```
<div class="box" style="border-style: dashed;">Dashed</div>
<div class="box" style="border-style: dotted;">Dotted</div>
<div class="box" style="border-style: double;">Double</div>
<div class="box" style="border-style: groove;">Groove</div>
<div class="box" style="border-style: inset;">Inset</div>
<div class="box" style="border-style: outset;">Outset</div>
<div class="box" style="border-style: ridge;">Ridge</div>
<div class="box" style="border-style: solid;">Solid</div>
```



```
<form>
   <fieldset>
         <input type="text" placeholder="emp name"><br><br>
         <input type="text" placeholder="emp mailid"><br>
   </fieldset>
</form>
```




```
<html>
<head>
  <style>
     fieldset{width:100px}
  </style>
</head>
   <body>
      <h1> Second Page</h1>
       <form>
          <fieldset>
             <input type="text" placeholder="emp name">
             <input type="text" placeholder="emp mailid">
          </fieldset>
       </form>
   </body>
</html>
```

Collapsing borders on table

```
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
```

```
<table border="1">
    <tr><th>First Name</th><th>Last Name</th></tr>
    <tr><td>Anita</td><td>Patil</td></tr>
    <tr><td>Kartik</td><td>Rao</td></tr>
    <tr><td>Veena</td><td>Deshmukh</td></tr>
  </table>
```

First Name	Last Name
Anita	Patil
Kartik	Rao
Veena	Deshmukh

```
<head>
    <style>
      table, th, td{border:1px solid black; border-collapse:collapse;}
    </style>
  </head>
<body>
  <table border="1">
    <tr><th>First Name</th><th>Last Name</th></tr>
    <tr><td>Anita</td><td>Patil</td></tr>
    <tr><td>Kartik</td><td>Rao</td></tr>
    <tr><td>Veena</td><td>Deshmukh</td></tr>
  </table>
```

First Name	Last Name
Anita	Patil
Kartik	Rao
Veena	Deshmukh

CSS3 border

- CSS 3 defines “border radius”, giving developers the possibility to make rounded corners on their elements.

```
<style>
#rcorners1 {
    border-radius: 25px;
    background: #8AC007;
    padding: 20px;
    width: 100px;
    height: 100px;
}
#rcorners2 {
    border-radius: 25px;
    border: 2px solid #8AC007;
    padding: 20px;
    width: 100px;
    height: 100px;
}
```

```
#rcorners3 {
    border-radius: 25px;
    background: url(paper.gif);
    background-position: left top;
    background-repeat: repeat;
    padding: 20px;
    width: 100px;
    height: 100px;
}
</style>
<p id="rcorners1">Rounded corners!</p>
<p id="rcorners2">Rounded corners!</p>
<p id="rcorners3">Rounded corners!</p>
```

CSS Styling

CSS Background

- CSS background properties are used to define the background effects of an element.
- CSS properties used for background effects:

background-color	Sets the background color of an element
background-image	Sets the background image for an element
background-position	Sets the starting position of a background image
background-repeat	Sets how a background image will be repeated

With CSS, a color is specified by:
- *a HEX value - like "#ff0000"*
- *an RGB value - like "rgb(255,0,0)"*
- *a color name - like "red"*

- Example:
  - div {background-color:#b0c4de;}
  - body {background-image:url('paper.gif');}
  - body {background-image:url('gradient2.png');background-repeat:repeat-x;}
  - body {background-image:url('img_tree.png'); background-repeat:no-repeat;
    background-position:right top; }

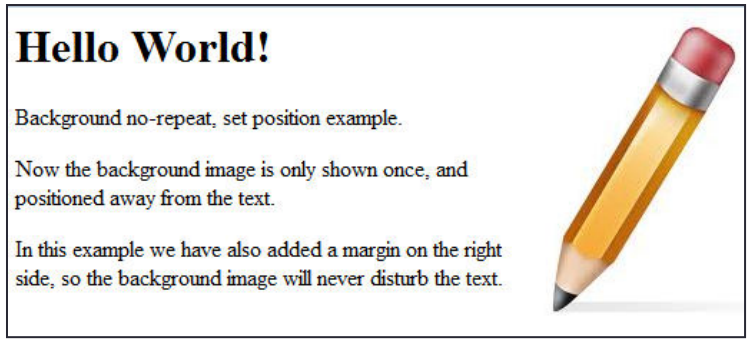
- The background-repeat property sets if/how a background image will be repeated.
  - By default, (repeat) : a background-image is repeated both vertically and horizontally.
  - no-repeat : The background-image is not repeated. The image will only be shown once

Demo : CSS Background

```
body {
    background-image: url("img_tree.gif"), url("img_flwr.gif");
    background-color: #cccccc;
}
```

```
body {
     background: #00ff00 url("smiley.gif") no-repeat fixed center;
}
```

```
<html>
<head><style>
body {
background-image:url('img_tree.png');
background-repeat:no-repeat;
background-position:right top;
margin-right:200px;
}
</style> </head>
<body>
<h1>Hello World!</h1>
<p>Background no-repeat, set position example.</p>
<p>Now the background image is only shown once, and positioned away from the text.</p>
<p>In this example we have also added a margin on the right side, so the background image will never disturb the text.</p>
</body></html>
```

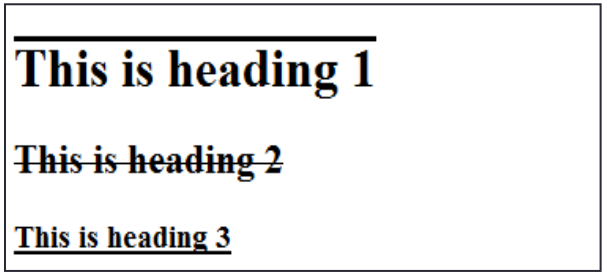


CSS Text

- CSS Text Properties

color	Sets the color of text
direction	Specifies the text direction/writing direction
letter-spacing	Increases or decreases the space between characters in a text
text-align	Specifies the horizontal alignment of text
text-decoration	Specifies the decoration added to text
text-indent	Specifies the indentation of the first line in a text-block
text-shadow	Specifies the shadow effect added to text
text-transform	Controls the capitalization of text
white-space	Specifies how white-space inside an element is handled
word-spacing	Increases or decreases the space between words in a text

```
<style>
h1 {text-decoration:overline;}
h2 {text-decoration:line-through;}
h3 {text-decoration:underline;}
</style>
<body>
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```



Demo :

```
<style>
p.uppercase {text-transform:uppercase;}
p.lowercase {text-transform:lowercase;}
p.capitalize {text-transform:capitalize;}
</style>
<body>
<p class="uppercase">This is some text.</p>
<p class="lowercase">This is some text.</p>
<p class="capitalize">This is some text.</p>
</body>
```

THIS IS SOME TEXT.

this is some text.

This Is Some Text.

```
<head> <style>
h1 {text-align:center;color:#00ff00;}
p.date {text-align:right;}
p.main {text-align:justify;}
p.ex {color:rgb(0,0,255);}
p.indent {text-indent:50px;}
</style> </head>
<body>
<h1>Hello World!</h1>
<p class="date"> Sep 2013</p>
<p class="main indent">The CSS property text-align corresponds to the attribute align used in old versions of HTML. Text can either be aligned to the left, to the right or centred. In addition to this, the value justify will stretch each line so that both the right and left margins are straight. You know this layout from for example newspapers and magazines. </p>
<p class="ex">The property text-decoration makes it is possible to add different "decorations" or "effects" to text. </p>
</body>
```

Hello World!

Sep 2013

The CSS property text-align corresponds to the attribute align used in old versions of HTML. Text can either be aligned to the left, to the right or centred. In addition to this, the value justify will stretch each line so that both the right and left margins are straight. You know this layout from for example newspapers and magazines.

The property text-decoration makes it is possible to add different "decorations" or "effects" to text.

CSS : Styling Fonts

- CSS font properties define the font family, boldness, size, and the style of a text.

font-family	Specifies the font family for text
font-size	Specifies the font size of text
font-style	Specifies the font style for text
font-variant	Specifies if a text should be displayed in a small-caps font
font-weight	Specifies the weight of a font

- Example:
  - p.normal {font-weight:normal;}
  - p{font-family:"Times New Roman", Times;}
  - p.italic {font-style:italic;}
  - h1 {font-size:40px;}
  - p.small { font-variant:small-caps; }

Introduction to Javascript

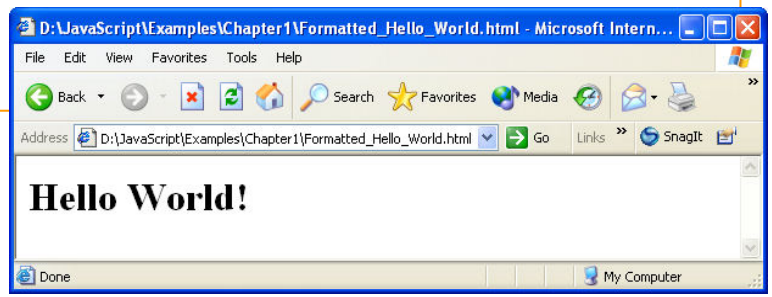
Overview

- JavaScript is Netscape's cross-platform, object-based scripting language
  - JavaScript code is embedded into HTML pages
  - It is a lightweight programming language
  - Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model
- Why use Javascript?
  - Provides HTML designers a programming tool :
  - Puts dynamic text into an HTML page
  - Reacts to events
  - Reads and writes to HTML elements :
  - Can be used to perform Client side validation

- The <SCRIPT> tag

```
<SCRIPT>
   JavaScript statements …
</SCRIPT>
```

```
<html>
<body>
<script type="text/javascript">
      document.write("<H1>Hello World!</H1>")
</script>
</body>
</html>
```



Where to Write JavaScript?

- Head Section
- Body Section
- External File

```
<html>
<head><title>script tag in body</title></head>
<body >
<script language="javascript">
      document.write("Hello!")
</script>
</body>
</html>
```

```
<html>
<head>
<script type="text/javascript">
function message()  {
        alert("Alert box called")
  }
</script>
</head>
<body onload="message()">
</body>
</html>
```

```
<head>
<script src="common.js">
      <!– no javascript statements →
   </ script>
</head>
<body>
   <script>
        document.write("display value of a variable"+msg)
   </script>
</body>
```

```
//common.js file contents
var msg
msg="<h1>in external file</h1>"
```

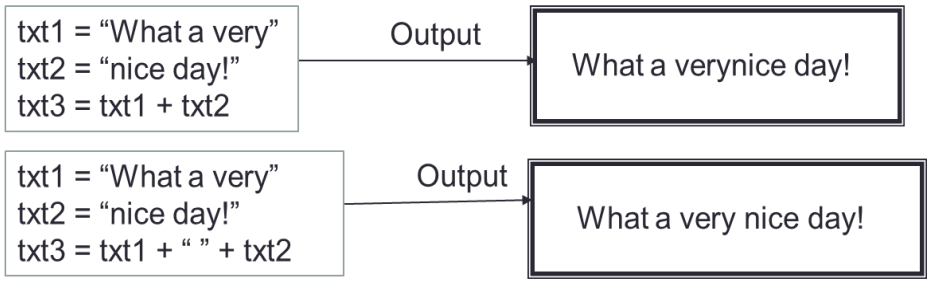
Data Types in JavaScript

- JavaScript is a free-form language. Need not declare all variables, classes, and methods
- Variables in JavaScript can be of type:
  - Number (4.156, 39)
  - String (“This is JavaScript”)
  - Boolean (true or false)
  - Null (null) → usually used to indicate the absence of a value

- **Defining variables**. var variableName = value
- JavaScript variables are said to be un-typed or loosely typed
  - letters of the alphabet, digits 0-9 and the underscore (_) character and is case-sensitive.
  - Cannot include spaces or any other punctuation characters.
  - First character of name must be either a letter or the underscore character.
  - No official limit on the length of a variable name, but must fit within a line.

Javascript operators:

- Arithmetic Operators (+ , -, * , / , %)
- Assignment Operators(=,+=,-=,*=,/=,%=)
- Comparison Operators (==,!=,<,<=,>,>=)
- Boolean Operators(&&,||,!)
- Bitwise Operators(&,|,!,^,<<,>>,>>>)
- String Operators(=,+,+=)



Typeof Operator

x	typeof x
undefined	"undefined"
null	"object"
true or false	"boolean"
any number or NaN	"number"
any string	"string"
any function	"function"
any nonfunction native object	"object"

typeof	undefinedvariable	“undefined”
typeof	33	“number”
typeof	“abcdef”	“string”
typeof	true	“boolean”
typeof	null	“object”

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined"
typeof null                   // Returns "object"
```

```
<SCRIPT LANGUAGE="JavaScript">

  var num1=20
  var str1="abc"
  var bool1=true
  var num2=null
  var var1;
  document.write("type of str1 : "+typeof(str1)+"<BR>")
  document.write("type of num1 : "+typeof(num1)+"<BR>")
  document.write("type of bool1 : "+typeof(bool1)+"<BR>")
  document.write("type of num2 : "+typeof(num2)+"<BR>")
  document.write("type of var1 : "+typeof(var1)+"<BR>")
</SCRIPT>
```

type of str1 : string
type of num1 : number
type of bool1 : boolean
type of num2 : object
type of var1 : undefined

Control Structures and Loops

- JavaScript supports the usual control structures:
- the conditionals:
  - if,
  - if...else
  - If … else if … else
  - Switch

```
if(condition) {
      statement 1
} else {
      statement 2
}
```

```
if(a>10) {
document.write("Greater than 10")
} else {
document.write("Less than 10")
}
```

- iterations:
  - for
  - while

```
document.write( (a>10) ? "Greater than 10" : "Less than 10" );
```

```
switch (variable) {
      case outcome1 :{
            //stmts for outcome 1
             break; }
      case outcome2 :{
            //stmts outcome 2
             break; }
       default: {
             //none of the outcomes is chosen
          }
}
```

```
<script>
    var n=20;
    switch(n){
       case 10: document.write("Ten");
                    break;
       case 20: document.write("Twenty");
                    break;
       case 30: document.write("Thirty");
                    break;
       default: document.write("Invalid!");
     }
</script>
```

Example

```
<script type = "text/javascript">
        <!--
           var grade = 'A';
           switch (grade) {
              case 'A': document.write("Good job<br />");
              break;

              case 'B': document.write("Pretty good<br />");
              break;

              case 'C': document.write("Passed<br />");
              break;

              case 'D': document.write("Not so good<br />");
              break;

              case 'F': document.write("Failed<br />");
              break;

              default:  document.write("Unknown grade<br />")
           }
           document.write("Exiting switch block");
        //-->
     </script>
```

Loop Statements

- For loop:

```
for( [initial expression;][condition;][increment expression] ) {

   statements
}
```

```
for(var i=0;i<10;i++)
{
        document.write("Hello");
}
```

- While loop:

```
while(condition) {
                statements
}
```

```
while(i<10) {
     document.write("Hello");
       i++;
}
```

- Break & Continue Statements supported

JavaScript Functions

```
function myFunction (arg1, arg2, arg3)
{
        statements
            return
}
```

Calling the function :

```
myFunction( “abc”, “xyz”, 4 )
myFunction()
```

```
function area(w1, w2, h) {
    var area=(w1+w2)*h/2;
    alert(area+" sq ft");
}
area(2,3,7);  //calling the function
```

```
function diameter(radius){
    return radius * 2;
}

var d=diameter(5); //calling the function
```

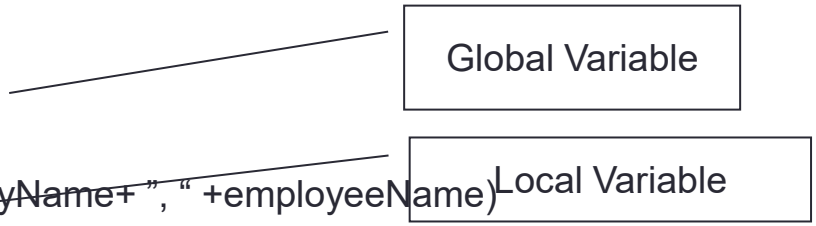
- **Function expressions -** functions are assigned to variables

```
var myFunction = function() {
    statements
}
```

```
var area = function (radius) {
      return Math.PI * radius * radius;
  };
alert(area(5));        // => 78.5
```

Global and Local Variables

```
<script language="Javascript">
  var companyName="TechnoFlo"
  function f(){
    var employeeName="Henry"
    document.write("Welcome to "+companyName+ ", " +employeeName)
  }
</script>
```



- Variables that exist only inside a function are called Local variables
  - The values of such variables can't be changed by main code or other functions
  - Within the body of a function, a local variable takes precedence over a global variable with the same name.
  - However, if I do not use "var" on a local variable, it is considered as a global variable!!
- Variables that exist throughout the script are called Global variables
  - Their values can be changed anytime in the code and even by other functions

Predefined Functions

- isFinite: evaluates an argument to determine if it is a finite number.
- If needed, the parameter is first converted to a number.

```
isFinite (number)      //where number is the number to evaluate
```

```
var a = isFinite(123) + "<br>";        //true
var b = isFinite(-1.23) + "<br>";      //true
var d = isFinite(0) + "<br>";          //true
var e = isFinite("123") + "<br>";      //true
var f = isFinite("Hello") + "<br>";    //false
```

- isNaN : Evaluates an argument to determine if it is “NaN” (not a number)

```
isNaN(0)                  //false
isNaN('123')              //false
isNaN('Hello')            //true
isNaN('2005/12/12') //true
```

- Parseint and parsefloat
  - Returns a numeric value for string argument.
  - parseInt (str)
  - parseFloat (str)

```
parseInt("3 blind mice")          // => 3
parseFloat(" 3.14 meters")        // => 3.14
```

ES6 - Next gen Javascript

- **const** : from JS 1.5 onwards.- to define constants
  - Eg :

```
const myBirthday = '18.04.1982';
myBirthday = '01.01.2001';   // error, can't reassign the constant!
```

```
const LANGUAGES = ['Js', 'Ruby', 'Python', 'Go'];
LANGUAGES = "Javascript";   // shows error.
LANGUAGES.push('Java');     // Works fine.
console.log(LANGUAGES);     // ['Js', 'Ruby', 'Python', 'Go', 'Java']
```

- **let** : to define block-scoped variables; can be used in four ways:
  - as a variable declaration like var; in a for or for/in loop, as a substitute for var;
  - as a block statement, to define new variables and explicitly delimit their scope
  - to define variables that are scoped to a single expression.
  - Eg : let message = 'Hello!';
  - Eg : let user = 'John', age = 25, message = 'Hello';

```
if (true) {
    let a = 40;
    console.log(a); //40
   }
   console.log(a); // undefined
```

```
let a = 50;  let b = 100;
if (true) {
   let a = 60;
   var c = 10;
   console.log(a/c); // 6
   console.log(b/c); // 10
}
console.log(c); // 10
console.log(a); // 50
```

ES6 - Next gen Javascript

- Arrow functions : allows you to create functions in a cleaner way compared to regular functions
  - is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

```
<script>
//non lambda
    function greet(name){
        console.log(name);
    }
    greet("shrilata");

    //lambda-eg1
    const greet1 = name => console.log(name);
    greet1("sandeep");

    //lambda-eg2
    const add = (a,b) => a + b;

    console.log(add(10,20));

    //lambda-eg3
    const strOp = str => {
                        console.log(str.length);
                        console.log(str.toUpperCase());
                        console.log(str.charAt(0));
                        };
     strOp("Hello");
```

```
function f1(){
    console.log("Hello-1");
  }
  f1()

  f2 = function(){
    console.log("Hello-2");
  }
  f2()

  f3 = () => console.log("Hello-3");
  f3()

  f4 = (name) => console.log("Hello-4 " + name);
  f4("shrilata")

  f5 = (a,b) => a+b;
  console.log("Hello-5 " + f5(3,4))

  f6 = (str) => {
    console.log(str.length);
    console.log(str.toUpperCase())
  }
  f6("hello world")
```

Console

```
"Hello-1"
"Hello-2"
"Hello-3"
"Hello-4 shrilata"
"Hello-5 7"
11
"HELLO WORLD"
```

Working with Predefined Core Objects

String Objects

- Creating a string object:
  - var myString = new String(“characters”)
  - var myString = “fred”

- Properties of a string object:
- length: returns the number of characters in a string.

```
• “Lincoln”.length // result = 7
• “Four score”.length // result = 10
• “One\ntwo”.length // result = 7
• “”.length // result = 0
```

- prototype
  - Allows you to add properties and methods to an object.
  - Syntax : object.prototype.name=value

Prototype

- A prototype is a property or method that becomes a part of every new object created after the prototype items have been added.
  - Sometimes you want to add new properties/methods to all existing objects of a given type.
  - For strings, as an example, you may want to define a new method for converting a string into a new type of HTML font tag not already defined by JavaScript's string object.
  - A function definition (makeItHot()) accumulates string data to be returned to the object when the function is invoked as the object's method.
  - The this keyword extracts the object making the call, which you convert to a string for concatenation with the rest of the strings to be returned.

```
function makeItHot() {
    return "<FONT COLOR='red'>" + this.toString() + "</FONT>“
}

String.prototype.hot = makeItHot
document.write("<H1>This site is on " + "FIRE".hot() + "!!</H1>")
```

String functions

- charAt(index) : returns the character at a specified position.
  - Eg : var str = "Hello world!";
    - str.charAt(0); //returns H
    - str.charAt(str.length-1)); //returns !

- concat() : joins two or more strings
  - stringObject.concat(stringX,stringX,...,stringX)
  - Eg: var str1="Hello ";
    var str2="world!";
    document.write(str1.concat(str2));

- indexOf () : returns the position of the first occurrence of a specified string value in a string.
  - index values start their count with 0.
  - If no match occurs within the main string, the returned value is -1.
  - string.indexOf(searchString [, startIndex])

```
Eg : var str="Hello world, welcome";
str.indexOf("Hello");  //returns 0
str.indexOf("wor"));  //returns 6
str.indexOf("e",5); //returns 14
```

String Objects

- match(regExpression)
  - Searches for a specified value in a string
  - string.match(regExpression)

```
str="rain in SPAIN is mainly in plain";
var patt1=/ain/gi;
document.write(str.match(patt1));
```

- replace(regExpression, replaceString)
  - Replaces some characters with some other characters in a string.
  - string.replace(regExpression, replaceString)
  - Eg: var str="Hello World";
    document.write(str.replace("World", "Everyone"));

```
var str = "To be, or not to be"
var regexp = /be/
str.relace(regexp, "exist")
```

- search(regExpression) : Searches a string for a specified value
  - Eg : var str="Hello World";
    str.search("World") //returns 6

```
var text = "testing: 1, 2, 3"; // Sample text
var pattern = /\d+/g // Matches all instances of one or more digits
text.search(pattern) // => 9: position of first match
text.match(pattern) // => ["1", "2", "3"]: array of all matches
text.replace(pattern, "#"); // => "testing: #, #, #"
```

String Objects (Parsing Methods Cont…)

- split(“delimiterCharacter”[, limitInteger]) - Splits a string into array of strings
  - string.split(“delimiterCharacter”[, limitInteger])

```
var str = "zero one two three four";
var arr = str.split(" ");
for(i = 0; i < str.length; i++){  document.write(“<br>” + arr[i]); }
```

Output :
zero
one
two
three
four

```
var myString = “Anderson,Smith,Johnson,Washington”
var myArray = myString.split(“,”)
var itemCount = myArray.length // result: 4
```

- toLowerCase() / toUpperCase()

```
Eg: var str="Hello World!";
str.toLowerCase()  //returns hello world
str.toUpperCase()  //returns HELLO WORLD
```

- slice(startIndex [, endIndex])
  - Extracts a part of a string and returns the extracted part in a new string

```
Eg : var str=“Hello World";
     str.slice(6)  //returns World
     str.slice(0,1)  //returns H
```

Math Properties

Property	Value	Description
Math.E	2.718281828459045091	Euler's constant
Math.LN	0.6931471805599452862	Natural log of 2
Math.LN10	2.302585092994045901	Natural log of 10
Math.LOG2E	1.442695040888963387	Log base-2 of E
Math.LOG10E	0.4342944819032518167	Log base-10 of E
Math.PI	3.141592653589793116	PI
Math.SQRT1_2	0.7071067811865475727	Square root of 0.5
Math.SQRT2	1.414213562373095145	Square root of 2

```
<script>
    document.write(Math.PI + "<br>");
    document.write(Math.SQRT2 + "<br>");
    document.write(Math.SQRT1_2 + "<br>");
    document.write(Math.E + "<br>");
</script>
```

3.141592653589793
1.4142135623730951
0.7071067811865476
2.718281828459045

Math Objects (Methods)

Method syntax	Returns
Math.abs(val)	Absolute value of *val*
Math.acos(val)	Arc cosine (in radians) of *val*
Math.asin(val)	Arc sine (in radians) of *val*
Math.atan(val)	Arc tangent (in radians) of *val*
Math.atan2(val1, val2)	Angle of polar coordinates *x* and *y*
Math.ceil(val)	Returns the value of x rounded up to its nearest integer
Math.cos(val)	Cosine of *val*
Math.exp(val)	Euler's constant to the power of *val*
Math.floor(val)	Next integer less than or equal to *val*
Math.max(val1, val2)	The greater of *val1* or *val2*
Math.min(val1, val2)	The lesser of *val1* or *val2*
Math.pow(val1, val2)	*Val1* to the *val2* power
Math.random()	Random number between 0 and 1
Math.round(val)	returns the nearest integer: N+1 when *val* >= n.5; otherwise N
Math.sin(val)	Sine (in radians) of *val*
Math.sqrt(val)	Square root of *val*
Math.tan(val)	Tangent (in radians) of *val*
Math.log(val)	Natural logarithm (base e) of val

Date

- Date object allows the handling of date and time information.
  - All dates are in milliseconds from January 1, 1970, 00:00:00.
  - Dates before 1970 are invalid dates.

- There are different ways to define a new instance of the date object:

```
var d = new Date()       //Current date
var d = new Date(milliseconds)
var d = new Date(dateString)
var d = new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

```
<script>
   var d=new Date();
   document.write(d);
</script>
```

Tue Sep 24 2013 12:48:39 GMT+0530 (India Standard Time)

```
var d = new Date(86400000);
var d = new Date(99,5,24,11,33,30,0);
```

Date Object - Methods

getDate( )	Date of the month (1 - 31)
getDay( )	Day of the week (0 - 6, 0-Sunday)
getMonth( )	The month (0 - 11, 0 - Jan.)
getFullYear( )	The year (4 digits)
getHours( )	Hour of the day (0 - 23)
getMinutes( )	Minutes (0 - 59)
getSeconds( )	Seconds (0 - 59)
getTime( )	Milliseconds since 1/1/1970
getTimezoneOffset( )	Offset between local time and GMT
setDate(dayValue)	1-31
setHours(hoursValue)	0-23
setMinutes(minutesValue)	0-59
setMonth(monthValue)	0-11
setSeconds(secondsValue)	0-59
setTime(timeValue)	>=0
setYear(yearValue)	>=1970

Array

- An array is the sole JavaScript data structure provided for storing and manipulating ordered collections of data.

- An array can be created in several ways:
  1. var empty = [];
  2. Regular:

```
var cars=new Array();   //empty array
cars[0]="Spark";
cars[1]="Volvo";
cars[2]="BMW";
```

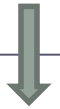
  3. Condensed: var cars=**new Array**("Spark","Volvo","BMW");
  4. Literal: var cars=["Spark","Volvo","BMW"];
  5. Misc : var misc = [1.1, true, "a",];
  6. var matrix = [[1,2,3], [4,5,6], [7,8,9]];
  7. var sparseArray = [1,,,,5];

Enumerating and Modifying Array Values

```
var myArray = ["Andrew","Monica","Catie","Jenna","Christopher"];
for(var i = 0; i < myArray.length; i++) {
console.log(myArray[i]);
}
```

Andrew
Monica
Catie
Jenna
Christopher

```
var myArray = ["Andrew","Monica","Catie","Jenna","Christopher"];
console.log("Before: ", myArray);
for(var i = 0; i < myArray.length; i++) {
    myArray[i] = myArray[i] + " Grant";
}
console.log("After: ", myArray);
```



Before: ["Andrew", "Monica", "Catie", "Jenna", "Christopher"]
After: ["Andrew Grant", "Monica Grant", "Catie Grant", "Jenna Grant", "Christopher Grant"]

```
var arr = [44,55,34,21,89];
for(var i=0; i < arr.length; i++)
    console.log(arr[i]);

for(var i in arr)
    console.log(i,arr[i]);

for(var i of arr)
    console.log(i)
```

```
44
55
34
21
89
0 44
1 55
2 34
3 21
4 89
44
55
34
21
89
```

Array Object Methods

- arrayObject.reverse()
- arrayObject.slice(startIndex, [endIndex])
- arrayObject.join(separatorString) : array contents will be joined and placed into arrayText by using the comma separator“
- arrayObject.push(): add one or more values to the end of an array

```
arrayObject.slice(startIndex [, endIndex])          //Returns: Array
var solarSys = new Array (“Mercury”,”Venus”,”Earth”,”Mars”,”Jupiter”,”Saturn”)
var nearby = solarSys.slice(1,4)
// result: new array of “Venus”, “Earth”, “Mars”
```

```
Andrew|Monica|Catie|Jenna
joinedarr type :  string
```

```
arrayObject.concat(array2)
var a1 = new Array(1,2,3)
var a2 = new Array(“a”,”b”,”c”)
var a3 = a1.concat(a2)
// result: array with values 1,2,3,”a”,”b”,”c”
```

```
var names = ["Andrew","Monica","Catie","Jenna"];
var joined_arr = names.join("|")
console.log(joined_arr)
console.log("joinedarr type :",typeof(joined_arr))
```

```
a = []; // Start with an empty array
a.push("zero") // Add a value at the end. a = ["zero"]
a.push("one", "two") // Add two more values. a = ["zero", "one", "two"]
```

Associative arrays

- Associative arrays are basically objects in JavaScript where indexes are replaced by user-defined keys.
- They do not have an index, or a length property like a normal array and cannot be traversed using a normal for loop.

```
var person = new Array()
person["name"]="shrilata"
person["age"]=50
person["email"]="shri@gmail.com"
person["city"]="Pune"
//alternate syntax - with dot(.) notation
person.phno = 9988776655

for(var prop in person)
    console.log(prop,person[prop])

console.log(person[0])  //undefined
```

```
name shrilata
age 50
email shri@gmail.com
city Pune
phno 9988776655
undefined
```

Creating New Objects

- **Using Object Initializers**
  - Syntax : objName = {property1:value1, property2:value2, … }
  - person = { "name ":"amit", "age":23};
  - myHonda = {color:“red”, wheels:4, engine:{cylinders:4, size:2}}

```
// Example 1
var myFirstObject = {};
myFirstObject.firstName = "Andrew";
myFirstObject.lastName = "Grant";
console.log(myFirstObject.firstName);
```

```
// Example 2
var mySecondObject = {
    firstName: "Andrew",
    lastName: "Grant"
};
console.log(mySecondObject.firstName);
```

```
// Example 3
var myFirstObject = {};
myFirstObject.firstName = "Andrew";
console.log(myFirstObject.firstName);
myFirstObject.firstName = "Monica";
console.log(myFirstObject.firstName);
myFirstObject["firstName"] = "Catie";
console.log(myFirstObject["firstName"]);
```

```
//Adding Methods to Objects
var person= {
  name: "Andrew",
  age: 21,
  info: function () {
    console.log(“Name" , this.name, “Age“, this.age );
   }
};
person.info();
for (var prop in person)
    console.log(person[prop]);
```

ES6 Javascript : Classes, properties and methods

- To declare a class, you use the class keyword with the name of the class
- There can only be one "constructor" in a class; SyntaxError will be thrown if the class contains more than one occurrence of a constructor method.

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }
  }
```

constructor method is always defined with the name "constructor"

Classes can have methods, which defined as functions, albeit without needing to use the function keyword.

```
class Rectangle {
    constructor(height, width) {
      this.height = height;
      this.width = width;
    }

    // Method
    calcArea() {
      return this.height * this.width;
    }
  }

  const square = new Rectangle(10, 10);

  console.log(square.calcArea()); // 100
```

ES6 Javascript : Classes and inheritance

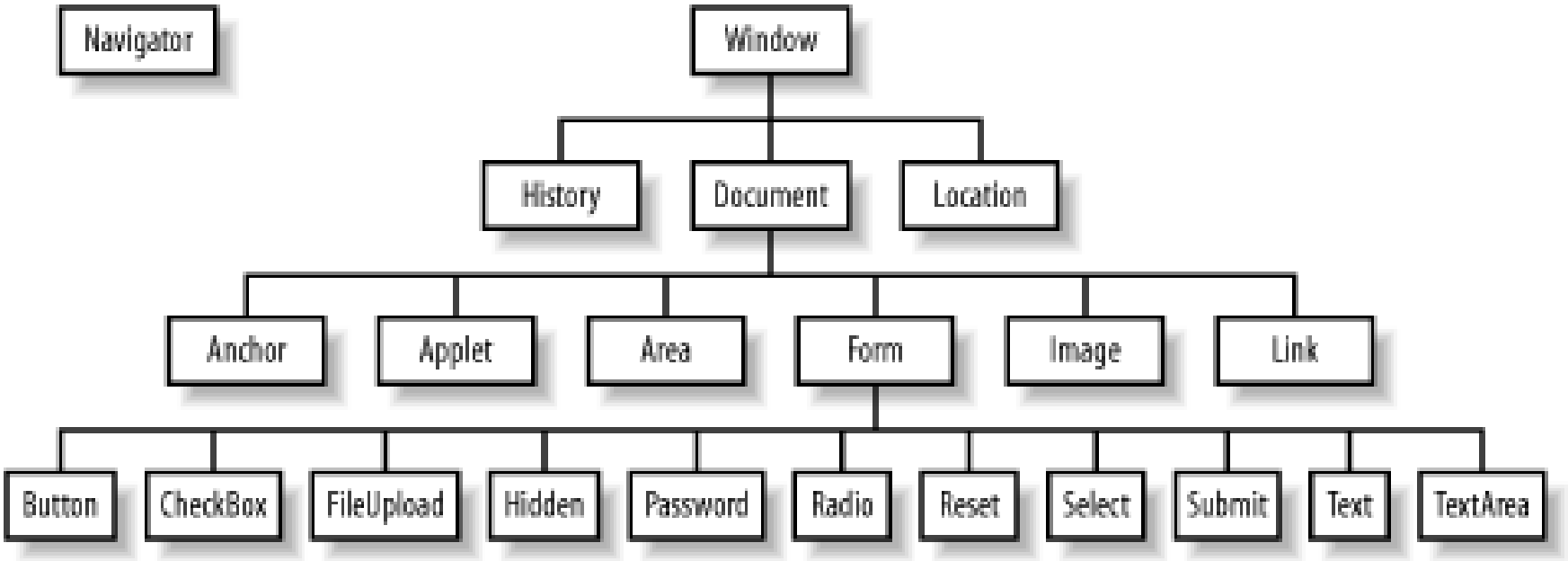
```
class Person{
      fname = "Anil";
      lname = "Patil";
      getFullName = () => this.fname + " " + this.lname;
      }

      const p1 = new Person();
      console.log(p1.getFullName()); //Anil Patil


class Stud{
   constructor(name){
       this.sname=name;
   }
   getName(){
       console.log(this.sname); //sun
    }
}
const s1 = new Stud("sunita");
s1.getName();
```

constructor method is always defined with the name "constructor"

Note that the function keyword is not used here

```
class GradStud extends Stud{
    constructor(){
        super("Kavita");
        this.gpa = 4.9;
    }
    getDetails(){
         super.getName();    //Kavita
         console.log("GPA : " + this.gpa);
    }
}

const gs = new GradStud();
gs.getDetails();
```

JavaScript Document Object Model



Navigator Object Hierarchy

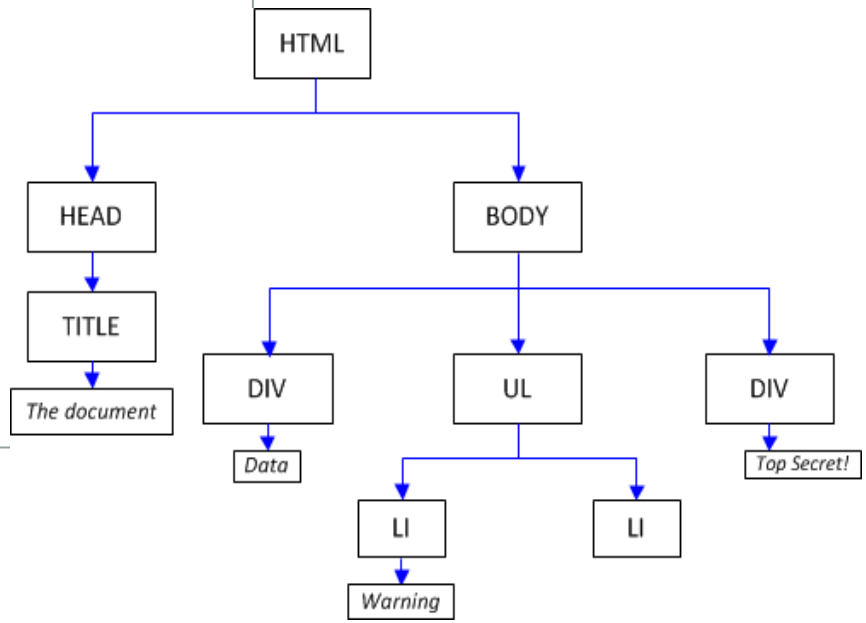
DOM tree example

```
<html>
    <head>
        <title>The document</title>
    </head>
    <body>
        <div>Data</div>
        <ul>
            <li>Warning</li>
            <li></li>
        </ul>
        <div>Top Secret!</div>
    </body>
</html>
```

Data

- Warning
-

Top Secret!



JavaScript Objects, Methods and Properties

- With a programmable object model, JavaScript gets all the power it needs to create dynamic HTML:
  - JavaScript can change all the HTML elements in the page
  - JavaScript can change all the HTML attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can react to all the events in the page

- Objects in an HTML page have:
  - methods (actions like opening a new window or submitting a form)
    - Eg : document.write(), window.alert(), window.prompt()
  - properties (attributes or qualities, such as color and size).
    - Eg:document.forms[0].phone.value = “555-1212”, array.length, document.bgColor,
  - Events
    - Specify how an object reacts to an event.
    - Eg : document.formName.button1.onclick=f1()
    - <INPUT TYPE=”button” NAME=”button1” onClick=”f1()”>

Working with Window Object

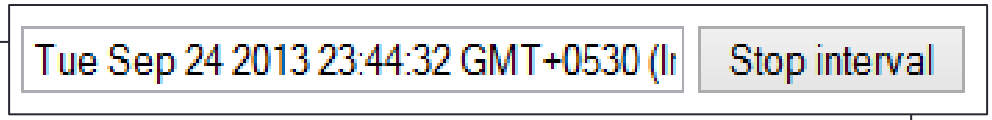
- Window object: Unique position at the top of the JavaScript object hierarchy.
- Window Object methods

alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message & OK and Cancel button
createPopup()	Creates a pop-up window
focus()	Sets focus to the current window
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
prompt()	Displays a dialog box that prompts the visitor for input
setInterval()	Calls a function/evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

- alert(message) : eg : window.alert(“Display Message”)
- confirm(message) : eg : window.confirm(“Exit Application ?”)
- prompt(message,[defaultReply]) : eg : var input=window.prompt(“Enter value of X”)

setInterval and setTimeout methods

```
<body>
<input type="text" id="clock" size="35" />
<script language=javascript>
var int=self.setInterval("clock()",50)
function clock()  {
    var ctime=new Date()
    document.getElementById("clock").value=ctime
 }
</script>
<button onclick="int=window.clearInterval(int)">Stop interval</button>
</body>
```




```
<head> <script type="text/javascript">
function timedMsg()   {
     var t=setTimeout("alert('5 seconds!')",5000)
 }
</script> </head>
<body> <p>Click on the button. An alert box will be  displayed after 5 seconds.</p>
<form>
<input type="button" value="Display timed alertbox!“ onClick="timedMsg()">
</form>
</body>
```

Document Object

- When an HTML document is loaded into a web browser, it becomes a document object; root node of the HTML document and owns all other nodes

document.anchors	Returns a collection of all the anchors in the document
document.baseURI	Returns the absolute base URI of a document
document.cookie	Returns all name/value pairs of cookies in the document
document.forms	Returns a collection of all the forms in the document
document.getElementById()	Returns the element that has the ID attribute with the specified value
document.getElementsByName()	Accesses all elements with a specified name
document.getElementsByTagName()	Returns a NodeList containing all elements with the specified tagname
document.images	Returns a collection of all the images in the document
document.lastModified	Returns the date and time the document was last modified
document.links	Returns a collection of all the links in the document
document.referrer	Returns the URL of document that loaded current document
document.title	Sets or returns the title of the document
document.URL	Returns the full URL of the document
document.write()	Writes HTML expressions or JavaScript code to a document
document.writeln()	Same as write(), but adds a newline character after each statement

Examples:

```
<html>
<body>
<p id="intro">Hello World!</p>
<p>This example demonstrates the <b>getElementById</b> method!</p>
<script>
    x=document.getElementById("intro");
document.write("<p>The text from the intro paragraph: " + x.innerHTML + "</p>");
</script>
</body>
</html>
```

Hello World!

This example demonstrates the **getElementById** method!

The text from the intro paragraph: Hello World!

```
<body>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>

<p>The paragraph above was changed by a script.</p>
</body>
```

Hello World!

Hello World!

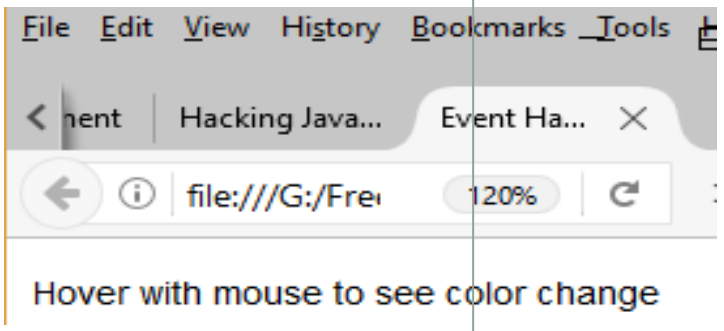
The paragraph above was changed by a script.

Examples

```
<script>
    function f1(){
        var str = "";
        var plist = document.getElementsByName("c1");
        for(i=0;i<plist.length;i++){
                if(plist[i].checked)
                   str = plist[i].value + "  " + str ;
         }
        document.getElementById("s1").innerHTML = "Skills : " + str;
     }
</script>
</head>
<body>
    <H1> Welcome to Javascript </H1>
   Skills:
    <input type="checkbox" name="c1" value="java">Java
    <input type="checkbox" name="c1" value="JS">JavaScript
    <input type="checkbox" name="c1" value="JSP">JSP

  <input type="button" value="click" onclick="f1()"><br>
  <span id="s1"></span>
</body>
```

Mouse events

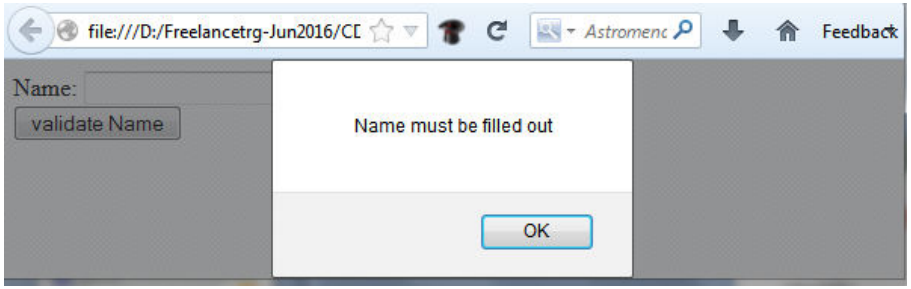
```
<html>
  <head>
    <SCRIPT>
      function changeColor(){
         var para = document.getElementById("p1");
        para.style.color="blue";
        para.style.backgroundColor = "lightgray";
        para.style.font = "italic bold 30px arial,serif";
      }
      function revertColor(){
        var para = document.getElementById("p1");
        para.style.color="black";
        para.style.backgroundColor = "white";
        para.style.font = "12px arial,serif";
      }
       </SCRIPT>
  </head>
<body>
  <div>
    <p id="p1" onmouseover="changeColor()"
               onmouseout="revertColor()"
               onclick="f1()" >
          Hover with mouse to see color change
    </p>
    </div>
</body></html>
```



Hover with mouse to see color change

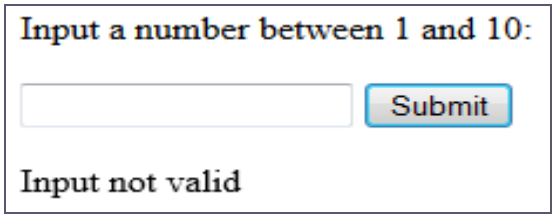
Example

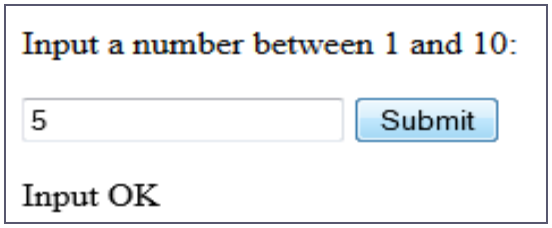
```
<html>
<head>
<script >
function validate(){
    var x=document.getElementById("fname").value;
    if (x == null || x == "") {
        alert("Name must be filled out");
        return false;
    }
}
</script>
</head>
<body>
<form id="form1"  onsubmit="return validate()">
Name: <input type="text" name="fname" id="fname" /><br />
<input type="submit" value="validate Name"  />
</form>
</body></html>
```



Example

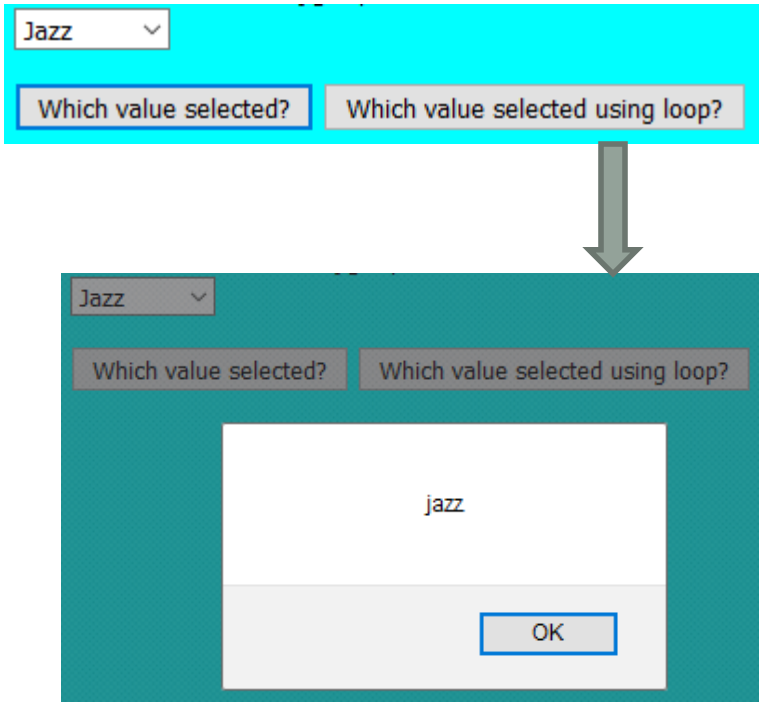
```
<body>
< Input a number between 1 and 10:</p>
<input id="numb">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
    var x, text;
   x = document.getElementById("numb").value;
   if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>
</body>
```



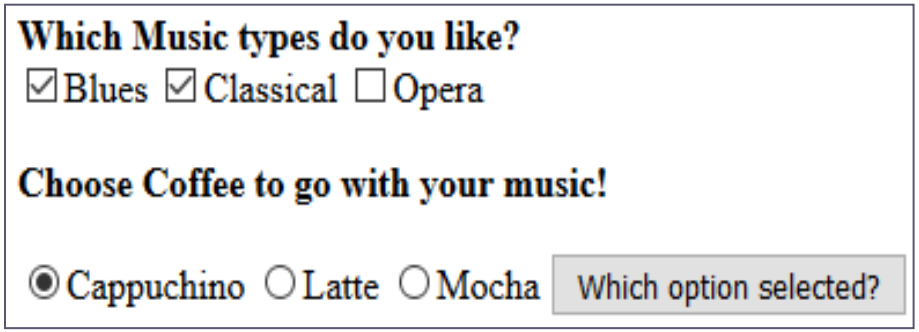


Example

```
<SCRIPT>
function valSelected1(){
  var sel = document.getElementById("musicTypes");
  alert(sel.value);       // prints value, not text
  var opt = sel.options[sel.selectedIndex];
  alert(opt.text);      //option.text prints text
}
function valSelected3(){
  var sel = document.getElementById("musicTypes");   var opt;
    for ( var i = 0, len =; i < sel.options.length; i++ ) {
        opt = sel.options[i];
        if ( opt.selected == true ) {   break;   }
    }
    alert(opt.value);
}
</SCRIPT>
<FORM NAME="selectForm">
  <SELECT name="musicTypes" id="musicTypes">
    <OPTION VALUE="rnb" SELECTED> R&B </OPTION>
    <OPTION VALUE="jazz"> Jazz </OPTION>
    <OPTION VALUE="blues"> Blues </OPTION>
</SELECT>
<INPUT TYPE="button" VALUE="value selected?"onClick="valSelected1()">
<INPUT TYPE="button" VALUE="value selected using loop?"onClick="valSelected3()">
</FORM>
</BODY>
```



Example



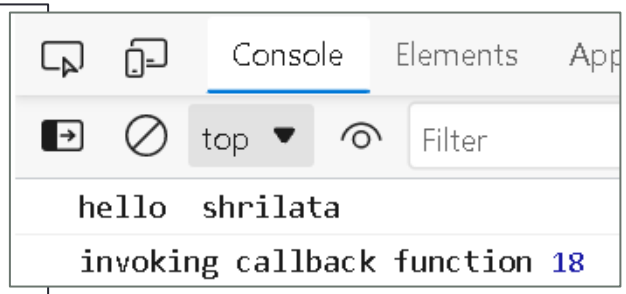

```
coffee selected : cappuchino
Music selected : blues
Music selected : classical
```

```
<SCRIPT>
 function valSelected(){
    var radio = document.getElementsByName("coffee");
    for(var i = 0; i < radio.length; i++){
       if(radio[i].checked)  console.log("coffee selected : " + radio[i].value);
    }
    var checklist = document.getElementsByClassName("c1");
    for(i=0;  i<checklist.length; i++){
        if (checklist[i].checked == true)  console.log("Music selected : " + checklist[i].value);
 }
</SCRIPT>
<FORM NAME="selectForm">
 <B>Which Music types do you like?</B>
 <input type="checkbox" class="c1" id="c1" value="blues">Blues</input>
 <input type="checkbox" class="c1" id="c2" value="classical">Classical</input>
 <input type="checkbox" class="c1" id="c3" value="opera">Opera</input>
 <b>Choose Coffee to go with your music!</b><br>
 <INPUT TYPE="radio" name="coffee"  id="coffee" VALUE="cappuchino">Cappuchino</input>
 <INPUT TYPE="radio" name="coffee"  id="coffee" VALUE="latte">Latte</input>
 <INPUT TYPE="radio" name="coffee"  id="coffee" VALUE="Mocha">Mocha</input>
 <INPUT TYPE="button" VALUE="Which option selected?"   onClick="valSelected()">
</FORM>
```

Callback functions

- A callback is a function passed as an argument to another function.

```
<script>
        function greet(name,age, myfunc){
          console.log("hello ", name)
          myfunc(age)
        }

        function callme(age){
          console.log("invoking callback function", age)
        }

        greet("shrilata", 18, callme)
    </script>
```



- The benefit of using a callback function is that you can wait for the result of a previous function call and then execute another function call.
  - So many examples seen so far : setTimeout(), setInterval(), geolocation API's getCurrentPosition(callback), $document.ready(callback), $btn.onclick(callback), most of newer functions in arrays like filter(), reduce(), map() etc

Callback functions

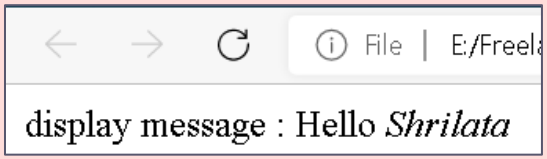
```
function greet() {
    console.log('Hello world');
}

function sayName(name) {
    console.log('Hello', name);
}

// calling the function
setTimeout(greet, 2000);
sayName('John');
```

Unobtrusive JavaScript

- Unobtrusive JavaScript, is JS that is separated from the web site's html markup.
  - It is the way of writing JavaScript language in which we properly separate Document Content and Script Content thus allowing us to make a clear distinction between them.
  - Promotes overall best practice

```
<body>
<script src="comm.js"></script>
<script>
  document.write("display message : "+msg)
</script>
</body>
```

```
//comm.js
var msg
msg="Hello <i>Shrilata</i>"
```




```
<head>
    <script>
    function greet(){
      console.log("Hello world")
    }
  </script>
</head>
<body>
  <input type="button" id="btn"
         value="click here"
         onclick="greet()" />
</body>
```

obtrusive js

```
<body>
  <script src="comm.js"></script>
  <input type="button" id="btn"
         value="click here"/>
  <script>
    btn1 = document.getElementById("btn")
    //btn1.addEventListener('click', () =>
console.log("hello"))
    btn1.onclick = greet; //func in comm.js
  </script>
</body>
```

unobtrusive js

AJAX & JSON

Introduction

- JSON (JavaScript Object Notation), is a simple and easy to read and write data exchange format.
  - It is easy for humans to read and write.
  - It is easy for machines to parse and generate.
  - It is based on a subset of the JavaScript, Standard ECMA-262
  - JSON is a text format that is completely language independent; can be used with most of the modern programming languages.
  - The filename extension is .json
  - JSON Internet Media type is application/json
  - It's popular and implemented in countless projects worldwide, for those don't like XML, JSON is a very good alternative solution.

  - Eg:

```
var JSONObjectName = { "name ":"amit", "age":23};
```

```
var JSONObject= {
                    "name":"John Johnson",
                    "street":"Oslo West 555",
                    "age":33,
                    "phone":"555 1234567"
                };
```

Values supported by JSON

- Strings : double-quoted Unicode, with backslash escaping
- Numbers:
  - double-precision floating-point format in JavaScript
- Booleans : true or false
- Objects: an unordered, comma-separated collection of key:value pairs enclosed in curly braces, with the ':' character separating the key and the value; the keys must be strings and should be distinct from each other
- Arrays : an ordered, comma-separated sequence of values enclosed in square brackets; the values do not need to be of the same type
- Null : A value that isn't anything

```
{
 "Students": [
              { "Name": "Amit Goenka",
                "Major": "Physics"
              },
              { "Name": "Smita Pallod",
                 "Major": "Chemistry"
              },
              { "Name": "Rajeev Sen",
                "Major": "Mathematics"
              } ]
 }
```

```
var obj = {
    "name":"Amit",
    "age": 37.5,
    "married": true,
    "address":{ "city":"Pune" , "state":"Mah" },
    "hobbies":["swimming","reading","music"]
}
```

Demo

```
<script language="javascript">
var JSONObject = {  "name" : "Amit",
       "address"  : "B-123 Bangalow",
       "age"  : 23,
       "phone"   : "011-4565763",
       "MobileNo"  : 0981100092
  };
var str =
"<h2><font color='blue'>Name </font>::“       +JSONObject.name+"</h2>“ +
 "<h2><font color='blue'>Address </font>::“   + JSONObject.address+"</h2>“ +
"<h2><font color='blue'>Age </font>::“          +JSONObject.age+"</h2>“ +
"<h2><font color='blue'>Phone No </font>::“  +JSONObject.phone+"</h2>“ +
"<h2><font color='blue'>Mobile No </font>::“  + JSONObject.MobileNo+"</h2>“;

document.write(str);
</script>
```

```
{
   "Title": "The Cuckoo's Calling",
   "Author": "Robert Galbraith",
   "Genre": "classic crime novel",
   "Detail": {
      "Publisher": "Little Brown",
      "Publication_Year": 2013,
      "ISBN-13": 9781408704004,
      "Language": "English",
      "Pages": 494
   },
   "Price": [
      {
         "type": "Hardcover",
         "price": 16.65
      },
      {
         "type": "Kidle Edition",
         "price": 7.03
      } ]
}
```

Name ::Amit

Address ::B-123 Bungalow

Age ::23

Phone No ::011-4565763

Mobile No ::981100092

Demo

```
<script >
var students = {
    "Students": [
            { "Name": "Amit Goenka",
              "Major": "Physics"
            },
            { "Name": "Smita Pallod",
               "Major": "Chemistry"
              },
              { "Name": "Rajeev Sen",
                "Major": "Mathematics"
              }] }
var i=0
document.writeln("students.Students.length : " + students.Students.length);
for(i=1;i<students.Students.length+1;i++)   {
    document.writeln("<b>Name : </b>" + students.Students[i].Name + "    ");
    document.writeln("<b>Majoring in : </b>" + students.Students[i].Major);
    document.writeln("</br>");
}
</script>
```

students.Students.length : 3
Name : Amit Goenka **Majoring in :** Physics
Name : Smita Pallod **Majoring in :** Chemistry
Name : Rajeev Sen **Majoring in :** Mathematics

Serializing Objects

- serialization is the process of converting an object's state to a string from which it can later be restored.
  - Sometimes we receive a raw JSON string, and we need to convert it to an object. And when we want to send a JavaScript object across the network, we need to convert it to JSON (a string) before sending.
  - **parse():** Accepts a JSON string as a parameter, and returns the corresponding JavaScript object.
  - **stringify():** Accepts an object as a parameter, and returns the equivalent JSON string.

```
o = {x:1, y:{z:[false,null,""]}};                    // Define a test object
s = JSON.stringify(o);              // s is '{"x":1,"y":{"z":[false,null,""]}}'
p = JSON.parse(s);                                   // p is a deep copy of o
console.log(p.x);                                    //1
```

```
<body>
<script>
//Imagine we received this text from a web server:
  var str = '{ "name":"John", "age":30, "city":"NY"}';
//Use JSON.parse() to convert text into JavaScript object
  var obj = JSON.parse(str);
</script>
<p id="p1"></p>
<script>
document.getElementById("p1").innerHTML = obj.name;
</script>
```

typeof(obj) returns object

What is Ajax ?

- “Asynchronous JavaScript And XML”
  - AJAX is not a programming language, but a technique for making the user interfaces of web applications more responsive and interactive
  - It provide a simple and standard means for a web page to communicate with the server without a complete page refresh.

- Why Ajax?
- Intuitive and natural user interaction
  - No clicking required. Call can be triggered on any event
  - Mouse movement is a sufficient event trigger
- "Partial screen update" replaces the "click, wait, and refresh" user interaction model
  - Only user interface elements that contain new information are updated (fast response)
  - The rest of the user interface remains displayed as it is without interruption (no loss of operational context)

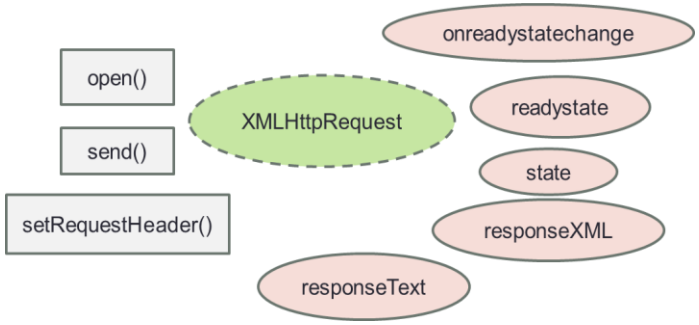
XMLHttpRequest

- JavaScript object
  - XMLHttpRequest object for asynchronously exchanging the XML data between the client and the server
  - Communicates with a server via standard HTTP GET/POST
  - XMLHttpRequest object works in the background ; Does not interrupt user operation

- XMLHttpRequest methods:
  - open(“method”, “URL”, booelan - syn/asyn) : Assigns destination URL, method, mode
  - send(content) : Sends request including string or DOM object data
  - abort() : Terminates current request
  - getAllResponseHeaders() : Returns headers (labels + values) as a string
  - getResponseHeader(“header”) : Returns value of a given header
  - setRequestHeader(“label”,”value”) : Sets Request Headers before sending

- XMLHttpRequest properties:
  - Onreadystatechange : Event handler that fires at each state change
  - readyState values – current status of request
  - Status : HTTP Status returned from server: 200 = OK
  - responseText : get the response data as a string
  - responseXML : get the response data as XML data

0: request not initialized
1: server connection established
2: request received
3: processing request
4: request finished and response is ready



Creating an AJAX application

- Step 1: Get an instance of XHR object

```
xhr = new XMLHttpRequest();
```

- Step 2: Make the request

```
xhr.open('GET', 'http://www.example.org/some.file');
xhr.send(null);
```

```
xhr.open('GET', 'AddNos.jsp?n1=100&n2=200');
xhr.send(null);
```

```
xhr.open("POST", "AddNos.jsp");
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhr.send("tno1=100&tno2=200");
```

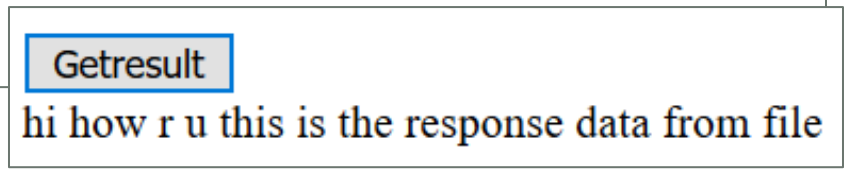
- Step 3 : Attach callback function to xhr object

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Ajax Demo

```
<script >
var xhr;
function getData(){
   xhr = new XMLHttpRequest();

   if(xhr){
      xhr.open("GET", "Sample.txt", true);
      xhr.send();

      xhr.onreadystatechange = function(){
          if(xhr.readyState == 4 &&  xhr.status == 200){
              document.getElementById("lblresult").innerHTML=xhr.responseText;
           }
      }  //end of callback function
}}

</script> <body>
    <input type="button" onclick="getData()"  value="Getresult"/>
    <div id="lblresult"></div>
</body>
```

```
//sample.txt
hi how r u this is the
response data from file
```



AJAX Demo with JSON

```
<script >
 var xmlhttp;
 function getData(){
   xmlhttp = new XMLHttpRequest();
   if(xmlhttp){
      xmlhttp.open("GET", "EmpJSONData.txt", true);
     xmlhttp.onreadystatechange = function(){
       if(xmlhttp.readyState == 4 && xmlhttp.status == 200){
            var obj = JSON.parse(xmlhttp.responseText);
            var displaytext = "";
            displaytext += "Emp name : " + obj.name + "<br>" +
                        "Designation : " + obj.desig + "<br>" +
                        "Age : " + obj.age + "<br>" +
                        "Salary : " + obj.sal;
           document.getElementById("lblres").innerHTML = displaytext;
        }
} }
</script><body>
<h3 id="lblres">Result</h3>
<input type="button" id="btngetjsondata" onclick="getData()"  value="GetData">
</body>
```

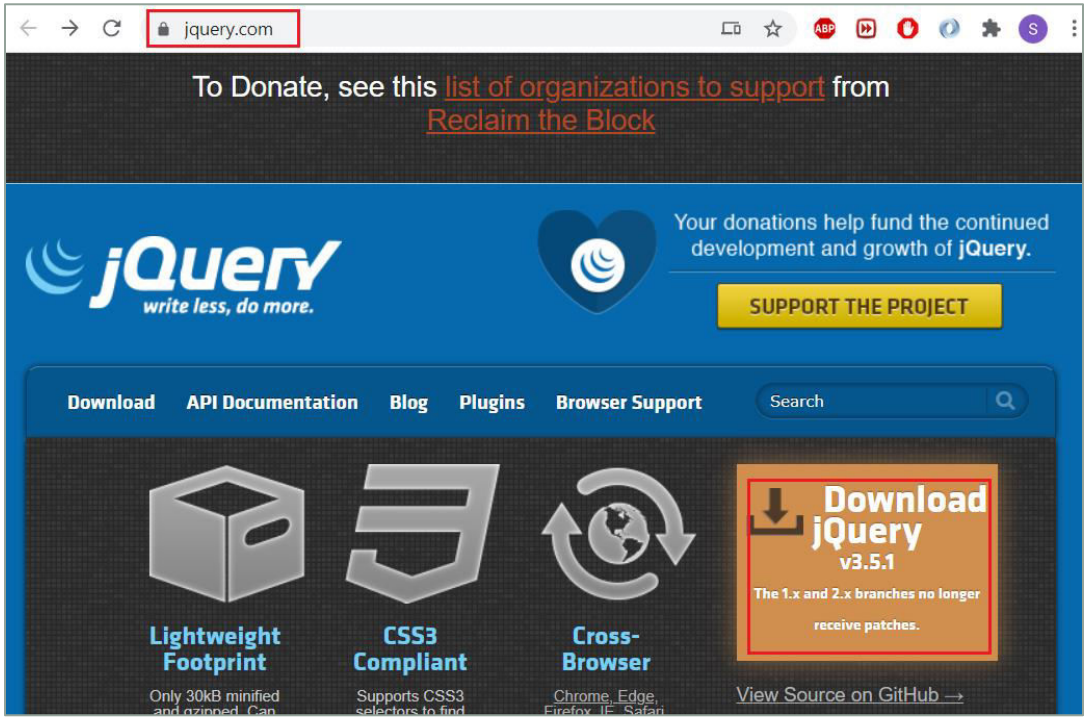
```
{ "name":"Kapil Verma",
  "desig":"ASE",
  "age":23,
  "sal":22000
}
```

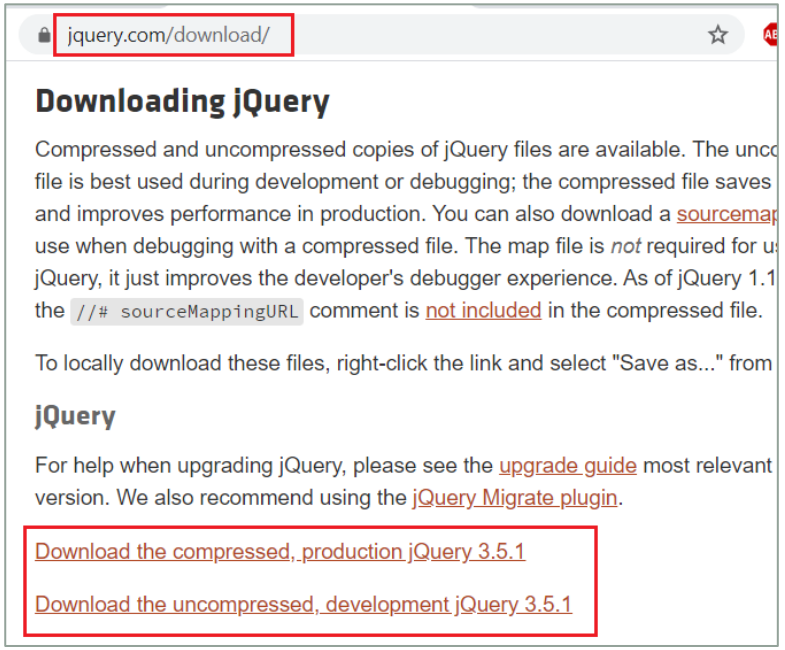
Emp name : Kapil Verma
Designation : ASE
Age : 23
Salary : 22000

GetJsonData

JQUERY

Software





- **Choosing a Text Editor**
  - Text editors that support jQuery include Brackets, Sublime Text, Kwrite, Gedit, Notepad++, PSPad, or TextMate.

jQuery Introduction

- jQuery is a lightweight, cross browser and feature-rich JavaScript library which is used to manipulate DOM
  - Originally created by John Resig in early 2006.
  - The jQuery project is currently run and maintained by a distributed group of developers as an open-source project.

- Why jQuery
  - JavaScript is great for a lot of things especially manipulating the DOM but it's pretty complex stuff. DOM manipulation is by no means straightforward at the base level, and that's where jQuery comes in. It abstracts away a lot of the complexity involved in dealing with the DOM, and makes creating effects super easy.
  - It can locate elements with a specific class
  - It can apply styles to multiple elements
  - It solves the cross browser issues
  - It supports method chaining
  - It makes the client side development very easy

Including jQuery in HTML Document

- jQuery library can be included in a document by linking to a local copy or to one of the versions available from public servers.
- Eg : include a local copy of the jQuery library

```
<html>
<head>
<title>Test jQuery</title>
<script src="../scripts/jquery-3.5.1.min.js"></script>
</head>
<body>
   <!-- bidy of HTML -->
</body>
</html>
```

- Eg : include the library from a publicly available repository
  - There are several well-known public repositories for jQuery; these repositories are also known as Content Delivery Networks (CDNs).

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.1.1.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
```

Using jQuery

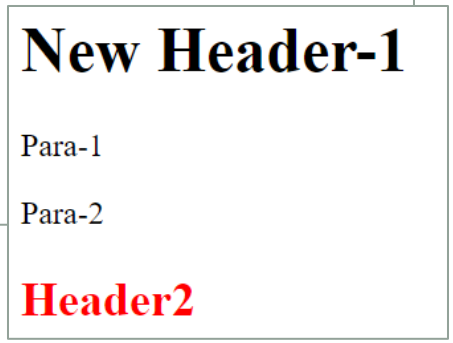
```
<html>
<head>
<title>Test jQuery</title>
<script type="text/javascript" src="jquery-3.5.1.js"></script>
<script type="text/javascript">
$(document).ready(function()  {
      alert('Hi');
 });
</script>
</head>
<body>
     Welcome to jQuery
</body>
</html>
```

```
$(function() {
    // jQuery code
});
```

Introduction to selectors

- jQuery uses same CSS selectors used to style html page to manipulate elements
  - CSS selectors select elements to add style to those elements where as jQuery selectors select elements to add behavior to those elements.
  - Selectors allow page elements (Single or Multiple) to be selected.
- Selector Syntax
  - $(selectorExpression)
  - jQuery(selectorExpression)
  - $(selector).action()

```
<html>
<head>
<title>Test jQuery</title>
<script src="../scripts/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
        $("h2").css("color","red");
        jQuery("h1").html("New Header-1");
    });
</script>
</head>
<body>
     <h1>jQuery Enabled</h1>
    <p>Para-1</p>
    <p>Para-2</p>
    <h2>Header2</h2>
</body>
</html>
```



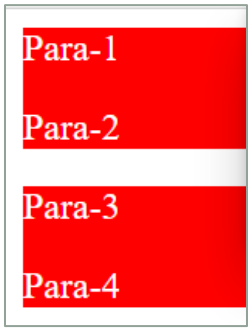
Selectors

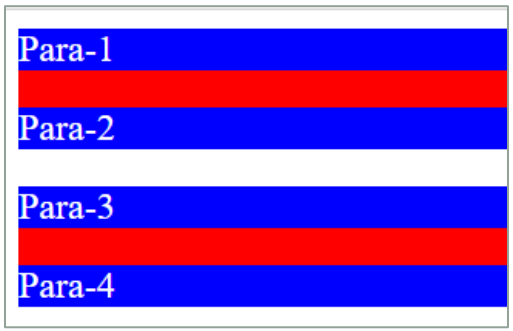
- **Selecting by Tag Name:**
- Selecting single tag takes the following syntax
  - $('p') – selects all <p> elements
  - $('a') – selects all <a> elements
- To reference multiple tags, use the (,) to separate the elements
  - $('p, a, span') - selects all paragraphs, anchors and span elements

- **Selecting Descendants**
- $('ancestor descendant') - selects all the descendants of the ansector
  - $('table tr') - Selects all tr elements that are the descendants of the table element
- Descendants can be children, grand children etc of the designated ancestor element.

Demo

```
<style type="text/css">
.redDiv{background-color:red; color:white;}
</style>
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function() {
    var paragraphs = $('p');
    alert(paragraphs.length);   //4
    paragraphs.css('background-color','blue');
    paragraphs.each(function(){
       alert($(this).html());
    });
    var collections = $('div,p');
    alert(collections.length);    //6
    var bodydivs = $('body div');
    alert(bodydivs.length);    //2
});
</script>
</head>
<body>
<div class="redDiv" >
<p>Para-1</p>
<p>Para-2</p>
</div>
<div  class="redDiv">
<p>Para-3</p>
<p>Para-4</p>
</div>
</body>
```





Selecting by Element ID

- It is used to locate the DOM element very fast.
- Use the # character to select elements by ID
  - $("#first") — selects the element with id="first"
  - $('#myID') – selects the element with id=" myID "

```
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function()
{
    alert($('#testDiv').html());
    $('#testDiv').html("Changed Text in Div!!");
    $("#p1").hide();
});
</script>
</head>
<body>
<div id="testDiv">Test Div</div>
<p id="p1"> Para-1</p>
<p>Para-2</p>
</body>
```

Test Div

Para-1

Para-2

Changed Text in Div!!

Para-2

Selecting Elements by Class Name

- Use the (.) character to select elements by class name
  - $(".intro") — selects all elements with class="intro"
- To reference multiple tags, use the (,) character to separate class name.
  - $('.blueDiv, .redDiv') - selects all elements containing class blueDiv and redDiv

```
<script>
$(document).ready(function(){
    $(".bold").css("font-weight", "bold");
});
</script>
</head>
<body>
<ul>
   <li class="bold">Test 1</li>
   <li>Test 2</li>
   <li class="bold">Test 3</li>
</ul>
</body>
```

- **Test 1**
- Test 2
- **Test 3**

```
<style type="text/css">
.blueDiv{background-color:lightblue; color:darkblue;}
.redDiv{background-color:orange; color:red;}
</style>
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function(){
    var collection = $('.blueDiv');
    collection.css('border','5px solid blue');
    collection.css('padding','10px');
});
</script>
</head>
<body>
<div class="blueDiv">
<p>para-1</p>
</div>
<div class="redDiv">
<p>para-2</p>
</div>
<div class="blueDiv">
<p>para-3</p>
</div>
</body>
```

para-1

para-2

para-3

Selecting by input elements

- To select all input elements
  - $(':input') - Selects all form elements (input, select, textarea, button).
  - $(":text") - All input elements with type="text"
  - $(":password") - All input elements with type="password"
  - $(":radio") - All input elements with type="radio"
  - $(":checkbox") - All input elements with type="checkbox"
  - $(":submit") - All input elements with type="submit"
  - $(":reset") - All input elements with type="reset"
  - $(":button") - All input elements with type="button"
  - $(":file") - All input elements with type="file"


```
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function()
{
    var inputs = $(':input');
    alert($(inputs[2]).val());  //Chennai
    $( ":text" ).css({ background: "yellow", border: "3px red solid" });
});
</script>
</head>
<body>
Name : <input id="txtName" type="text" value="Karthik"><br>
Age : <input id="txtAge" type="text" /><br>
City :
<select id="city">
<option value="Bangalore">Bangalore</option>
<option value="Chennai" selected="selected">Chennai</option>
<option value="Mumbai">Mumbai</option>
</select><br>
</body>
```

Filters

- The index-related selectors (:eq(), :lt(), :gt(), :even, :odd) filter the set of elements that have matched the expressions that precede them.
  - They narrow the set down based on the order of the elements within this matched set.
  - Eg, if elements are first selected with a class selector (.myclass) and four elements are returned, these elements are given indices 0 through 3
- eq() - Select the element at index n within the matched set.
  - Eg : $("p:eq(1)") - Select the second <p> element
  - Eg : $("element:eq(0)") is same as $('element:first-child')
- $('element:odd') and $('element:even') selects odd and even positions respectively. 0 based indexing
  - Odd returns (1,3,5…) and Even returns (0,2,4…)
- :gt() and lt() - Select all elements at an index > or < index within the matched set
  - Eg : $("tr:gt(3)") : Select all <tr> elements after the 4 first
  - Eg : $("tr:lt(4)") : Select the 4 first <tr> elements

```
<table border="1">
  <tr><td>TD #0</td><td>TD #1</td><td>TD #2</td></tr>
  <tr><td>TD #3</td><td>TD #4</td><td>TD #5</td></tr>
  <tr><td>TD #6</td><td>TD #7</td><td>TD #8</td></tr>
</table>
<script>
  $( "td:eq( 2 )" ).css( "color", "red" );
  //$( "tr:first" ).css( "font-style", "italic" );  //is same as below line
  $( "tr:eq(0)" ).css( "font-style", "italic" );
  $( "td:gt(4)" ).css( "backgroundColor", "yellow" );
</script>
```

TD #0	*TD #1*	*TD #2*
TD #3	TD #4	TD #5
TD #6	TD #7	TD #8

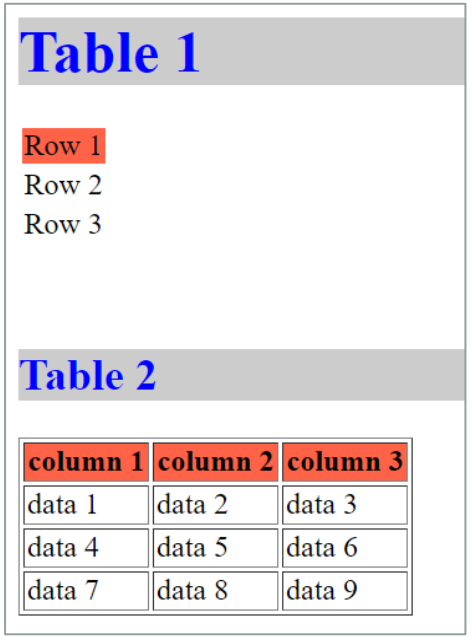
```
<script type="text/javascript">
$(document).ready(function() {
    $('tr:odd').css('background-color','tomato');
    $('tr:even').css('background-color','bisque');
});
</script>
<table border=1 cellspacing=5 cellpadding=5>
  <th>column 1<th>column 2<th>column 3
  <tr><td>data 1</td><td>data 2</td><td>data 3
  <tr><td>data 4<td>data 5<td>data 6
  <tr><td>data 7<td>data 8<td>data 9
  <tr><td>data 10<td>data 11<td>data 12
</table>
```

column 1	column 2	column 3
data 1	data 2	data 3
data 4	data 5	data 6
data 7	data 8	data 9
data 10	data 11	data 12

Filters

- :first , :last - Selects the first/last matched element.
  - :first is equivalent to :eq(0) and :lt(1). This matches only a single element, whereas, :first-child can match more than one; one for each parent.
- :header - Selects all elements that are headers, like h1, h2, etc

```
<script>
$(document).ready(function() {
    $( ":header" ).css({ background: "#ccc", color: "blue" });
    $('tr:first-child').css('background-color','tomato');
});
</script>
</head>
<body>
<h1>Table 1</h1>
<table>
  <tr><td>Row 1</td></tr>
  <tr><td>Row 2</td></tr>
  <tr><td>Row 3</td></tr>
</table>
<br/><br/>
<h2>Table 2</h2>
<table border=1>
<th>column 1<th>column 2<th>column 3
<tr><td>data 1</td><td>data 2</td><td>data 3
```



Filter

- $('element:first-child') and $('element:last-child') selects the first child & last child of its parent.
  - $('span:first-child') returns the span which is a first child for all the groups
- :nth-child() - Selects all elements that are the nth-child of their parent. 1-based indexing
  - Eg : $("p:nth-child(3)") - Select each <p> element that is the third child of its parent

- :contains() will select elements that match the contents.
  - $('div:contains("hello")') - selects div's which contains the text hello (match is case sensitive)

```
  <style>
  span {color: blue; }
  </style>
</head>
<body>
  <ul>
    <li>John</li>
    <li>Karl</li>
    <li>Brandon</li>
  </ul> <hr>
  <ul><li>Sam</li></ul> <hr>
  <ul>
    <li>Glen</li>
    <li>Tane</li>
  </ul>
 <script>
$( "ul li:nth-child(2)" ).append( "<span> - 2nd!</span>" );
</script>
```

- John
- Karl - 2nd!
- Brandon

- Sam

- Glen
- Tane - 2nd!

```
 <style>
   span {color: #008;}
   span.sogreen { color: green; font-weight: bolder;}
   span.solast {text-decoration: line-through;}
</style>
<script src="..\Scripts\jquery-3.5.1.min.js"></script>
</head>
<body>
<div> <span>John,</span><span>Karl,</span><span>Brandon</span></div>
<div> <span>Glen,</span> <span>Tane,</span><span>Ralph</span> </div>
<script>
$( "div span:first-child" ) .css( "text-decoration", "underline" )
     .hover(function() {
          $( this ).addClass( "sogreen" );
        }, function() {
            $( this ).removeClass( "sogreen" );
       });
$( "div span:last-child" ).css({ color:"red", fontSize:"80%" })
.hover(function() {
      $( this ).addClass( "solast" );
   }, function() {
      $( this ).removeClass( "solast" );
    });
$("div span:nth-child(2)").css("background-color","yellow");
</script>
```

John,Karl,Brandon
Glen, Tane,Ralph

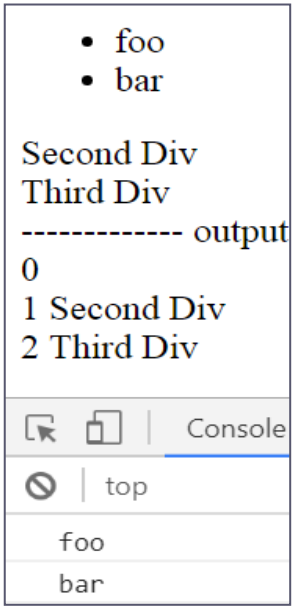
John,Karl,Brandon
Glen, Tane,Ralph

John,Karl,Brandon
Glen, Tane,~~Ralph~~

Iterating through Nodes

- .each(function(index,Element)) is used to iterate through jQuery objects.

```
<script>
$(document).ready(function(){
    var result = "";
    $('div.One,div.Two,div.Three').each(function(index) {
        result+=index+" "+$(this).text()+"<br/>";
    });
    $('#OutputDiv').html(result);
    //try with element
    $("li").each(function(){
       console.log($(this).text())
    });
});
</script>
</head>
<body>
<ul>
   <li>foo</li>
   <li>bar</li>
</ul>
<div class="One"></div>
<div class="Two">Second Div</div>
<div class="Three">Third Div</div>
------------- output from jQuery ---------------
<div id="OutputDiv" />
</body>
```



Working with Attributes

- Object attributes can be used using attr():
  - var val = $('#customDiv').attr('title'); - Retrieves the title attribute value

- .attr(attributeName,value) : accesses an object's attributes and modifies the values.
  - $('img').attr('title','Image title'); - changes the title attribute value to Image title.

- To modify multiple attributes, pass JSON object.

```
$("img").attr({
          "title":  "image title",
          "style" : "border:5px dotted red"

    });
```

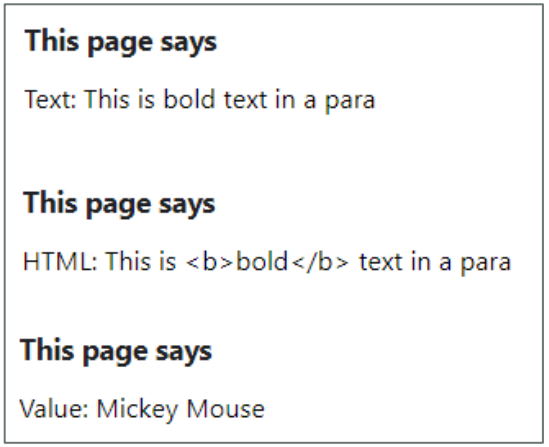
```
<img src="paintbrush.gif" title="image
title" style="border:5px dotted red">
```

- You can also remove attributes entirely using .removeAttr().

Getting Content

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

```
<script type="text/javascript">
    $(document).ready(function(){
        $("#btn1").click(function(){
            alert("Text: " + $("#test").text());
        });
        $("#btn2").click(function(){
            alert("HTML: " + $("#test").html());
        });
        $("#btn3").click(function(){
            alert("Value: " + $("#test1").val())
        });
});
</script>
</head>
<body>
<p id="test">This is <b>bold</b> text in a para</p>
<input id="test1" value="Mickey Mouse"><br>
<button id="btn1">Button1</button>
<button id="btn2">Button2</button>
<button id="btn3">Button3</button>
```



$.html() treats the string as HTML, $.text() treats the content as text

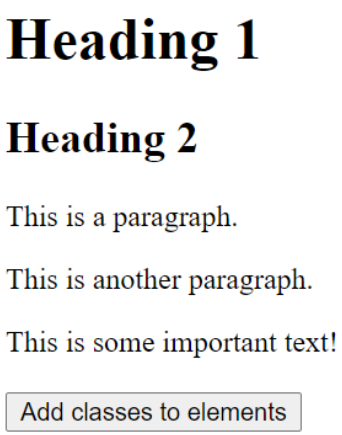
Adding and Removing Nodes

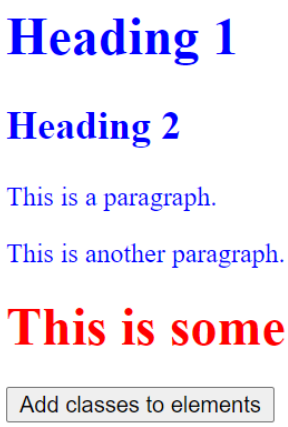
- In traditional approach adding and removing nodes is tedious.
- To insert nodes four methods available:
- Appending adds children at the end of the matching elements
  - .append() , .appendTo()
- Prepending adds children at the beginning of the matching elements
  - .prepend() , .prependTo()
- To wrap the elements use .wrap()
  - Eg: $(“p”).wrap(“<h1></h1>”) //wraps “p” in <h1> tags
- To remove nodes from an element use .remove()

```
<head>
<title>Simple Append</title>
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function() {
    $("button").click(function(){
        $("h2").append("<p>Surprise text!</p>");
    });
});
</script>
</head>
<body>
    <h2>This is a header.</h2>
    <button>Click me</button>
</body>
```

Working with Classes

- The four methods for working with css class attributes are
  - .addClass() : adds one or more classes to the class attribute of each element.
    - $('p').addClass('classOne');
    - $('p').addClass('classOne classTwo');
  - .hasClass() : returns true if the selected element has a matching class
    - if($('p').hasClass('classOne')) { //perform operation}
  - removeClass() remove one or more classes
    - $('p').removeClass('classOne classTwo');
  - To remove all class attributes for the matching selector
    - $('p').removeClass();
  - .toggleClass() : toggles adding/removing a class based on the current presence or absence of the class.
    - $('#targetDiv').toggleClass('highlight');






```
<style>
.important{
  font-weight:bold;
  font-size:xx-large;
  color:red;
}
.blue {color:blue;}
</style>
<script>
$(document).ready(function() {
    $("button").click(function(){
        $("h1,h2,p").addClass("blue");
        $("div").addClass("important");
  });
});
</script>
</head>
<body>
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<div>This is some important text!</div>
<button>Add classes to elements</button>
```

jQuery Event Model Benefits

- Events notify a program that a user performed some type of action
- jQuery Events
  - click()
  - blur()
  - focus()
  - dblclick()
  - mousedown()
  - mouseup()
  - mouseover()
  - keydown()
  - keypress()
  - hover() : hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

```
$("img").mouseover(function () {
        $(this).css("opacity", "0.3");
 });
 $("img").mouseout(function () {
         $(this).css("opacity", "1.0");
 });
```

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
  alert("Bye! You now leave p1!");
});
```

Handling Click Events

- .click(handler([eventObject])) is used to listen for a click event or trigger a click event on an element
  - $('#submitButton').click(function() { alert('Clicked') });

```
<script>
$(document).ready(function(){
    $('#SubmitButton').click(function(){
        var fName = $('#txtFirstName').val();
        var lName = $('#txtLastName').val();
        $('#tgtDiv').html("<b>"+fName+" "+lName+"</b>");
    });
});
</script>
</head>
<body>
FirstName : <input id="txtFirstName" type="text" /><br>
LastName : <input id="txtLastName" type="text" /><br>
<input id="SubmitButton" type="submit" value="Submit"/><br>
<div id="tgtDiv" />
</body>
```

FirstName : Anil

LastName : Patil

Submit

Anil Patil

```
<style>
  p {
    color: red;
    margin: 5px;
    cursor: pointer;
  }
  p:hover {background: yellow;}
  </style>
</head>
<body>
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p>Yet one more Paragraph</p>

<script>
 $( "p" ).click(function() {
   $( this ).slideUp();
});
</script>
</body>
```

First Paragraph

Second Paragraph

Yet one more Paragraph

```
<style type="text/css">
.highlight{background-color:yellow;};
</style>
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
        $(document).ready(function(){
            $('#targetDiv').mouseenter(function(){
                toggle(this);
            });
            $('#targetDiv').mouseleave(function(){
                toggle(this);
            });
            $('#targetDiv').mouseup(function(e){
                $(this).text('X : '+e.pageX+' Y :'+e.pageY );
            });
        });
        function toggle(element){
            $(element).toggleClass('highlight');
        }
</script>
</head>
<body>
<div id="targetDiv">Welcome to events</div>
</body>
```

Welcome to events

X : 123 Y :17

Using on() and off()

- The on() method attaches one or more event handlers for the selected elements.

```
$( "#dataTable tbody tr" ).on( "click", function() {
    console.log( $( this ).text() );
});
```

```
<script>
$(document).ready(function() {
  $("#div1").on("click",function(){
     $(this).css("background-color","pink");
  });
  $("h2").on("mouseover mouseout",function(){
     $(this).toggleClass("intro");
  });
  $("#div2").on({
    mouseover: function(){
      $(this).css("background-color", "lightgray");
    },
    mouseout: function(){
     $(this).css("background-color", "lightblue");
    },
    click: function(){
      $(this).hide();
    }
  });
});
</script>
```

Header-2

Text within h4 element

Text within para

This is Div-2

Header-2

Text within h4 element

Text within para

```
<body>
<h2>Header-2</h2>
<div id="div1">
 <h4>Text within h4 element</h4>
 <p>Text within para</p>
</div>
<div id="div2"> This is Div-2 </div>
</body>
```

Showing and Hiding Elements

- To set a duration and a callback function
  - show(duration, callback)
  - duration is the amount of time taken (in milliseconds), and callback is a callback function jQuery will call when the transition is complete.
- The corresponding version of hide( )
  - hide(duration, callback)
- To toggle an element from visible to invisible or the other way around with a specific speed and a callback function, use this form of toggle( )
  - toggle(duration, callback)

```
<style type="text/css">
.blueDiv{
  background-color:blue;
  color:white;font-size:30pt;
  display:none
  }
</style>
<script src="..\Scripts\jquery-3.5.1.js"></script>
<script>
$(document).ready(function(){
   $('#btnShow').click(function(){
      //fast(200),normal(400) and slow(600) can also be s
      $('.blueDiv').show(5000,function(){
         $('#results').text('Show Animation Completed');
      });
    });
    $('#btnHide').click(function(){
      $('.blueDiv').hide(5000,function(){
        $('#results').text('Hide Animation Completed');
       });
    });
    $('#btnSH').click(function(){
      $('.blueDiv').toggle(5000,function(){
        $('#results').text('Animation Completed');
      });
    });
});
</script>
```


```
<body>
<input id="btnShow" type="button" value="Show"/>
<input id="btnHide" type="button" value="Hide"/>
<input id="btnSH" type="button" value="Show/Hide"/>
<div class="blueDiv">
<p>Welcome to animations!!</p>
</div>
<div id="results"></div>
</body>
```

jQuery Sliding Effects

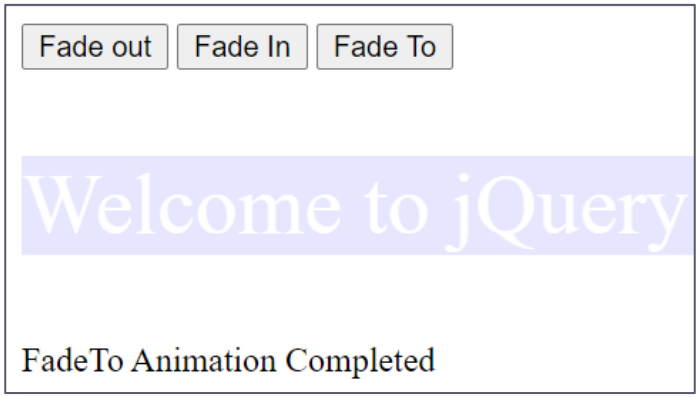
- The jQuery slide methods slide elements up and down.
- jQuery has the following slide methods:
  - $(selector).slideDown(speed,callback) : *Display the matched elements with a sliding motion*
  - $(selector).slideUp(speed,callback) : *Hide the matched elements with a sliding motion.*
  - $(selector).slideToggle(speed,callback)
- The speed parameter can take the following values: "slow", "fast", "normal", or milliseconds.
- The callback parameter is the name of a function to be executed after the function completes.

```
$("#flip").click(function(){
    $("#panel").slideDown();
});
```

jQuery Fading Effects

- The jQuery fade methods gradually change the opacity for selected elements.
- jQuery has the following fade methods:
  - $(selector).fadeIn(speed,callback)
  - $(selector).fadeOut(speed,callback)
  - $(selector).fadeTo(speed,opacity,callback)
- The speed parameter can take the following values: "slow", "fast", "normal", or milliseconds.
  - The opacity parameter in the fadeTo() method allows fading to a given opacity.
  - The callback parameter is the name of a function to be executed after the function completes.

```
$('.blueDiv').fadeTo(1000,0.1,function(){
        $('#results').text('FadeTo Animation Completed');
});
```



NODE.JS

Server Side Javascript

Node.js – an intro

- In 2009 Ryan Dahl created Node.js or Node, a framework primarily used to create highly scalable servers for web applications.
  - Node.js is an open source, cross-platform runtime environment for server-side JavaScript.
  - Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute code.
  - It is written in C++ and JavaScript.
  - Node.js is a development framework that is based on Google's V8 JavaScript engine that powers Google's Chrome web browser.
  - You write Node.js code in JavaScript, and then V8 compiles it into machine code to be executed.

  - It's a highly scalable system that uses asynchronous, non-blocking I/O model (input/output), rather than threads or separate processes
  - It is not a framework like jQuery nor a programming language like C# or JAVA; Its primarily a Javascript engine

Node.js is really two things: a runtime environment and a library

Traditional Programming vs Event-driven programming

- In traditional programming I/O is performed in the same way as it does local function calls. i.e. Processing cannot continue until the operation is completed.
  - When the operation like executing a query against database is being executed, the whole process/thread idles, waiting for the response. This is termed as “Blocking”
  - Due to this blocking behavior we cannot perform another I/O operation, the call stack becomes frozen waiting for the response.

```
result = query('SELECT * FROM posts WHERE id = 1');
do_something_with(result);
```

Typical blocking I/O programming

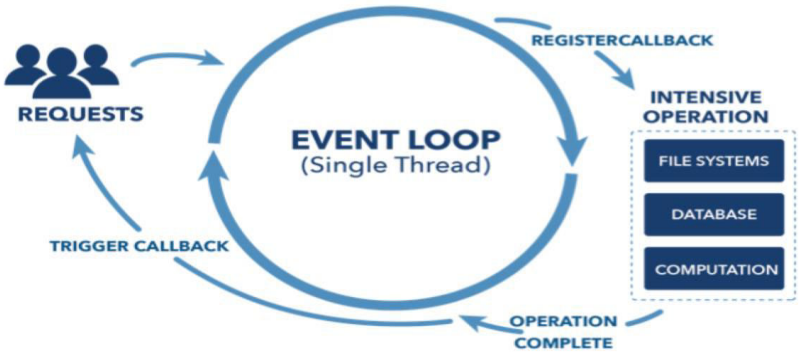
- Event-driven programming or Asynchronous programming is a programming style where the flow of execution is determined by events.
- Events are handled by event handlers or event callbacks
  - An event callback is a function that is invoked when something significant happens like when the user clicks on a button or when the result of a database query is available.
  - This style of programming — whereby instead of using a return value you define functions that are called by the system when interesting events occur — is called event-driven or asynchronous programming.

```
query_finished = function(result) {
    do_something_with(result);
}
query('SELECT * FROM posts WHERE id = 1', query_finished);
```

Event loop

- An event loop is a construct that mainly performs two functions in a continuous loop — **event detection and event handler triggering**.
  - In any run of the loop, it has to detect which events just happened.
  - Then, when an event happens, the event loop must determine the event callback and invoke it.
- This event loop is just one thread running inside one process, which means that, when an event happens, the event handler can run without interruption. This means the following:
  - There is at most one event handler running at any given time.
  - Any event handler will run to completion without being interrupted.

  - Node.js uses the “Single Threaded Event Loop” architecture to handle multiple concurrent clients.



Asynchronous and Event Driven

- All APIs of Node.js library are asynchronous that is, non-blocking.
  - It essentially means a Node.js based server never waits for an API to return data.
  - The server moves to the next API after calling it and a notification mechanism of Node.js helps the server to get a response from the previous API call.
  - It is non-blocking, so it doesn't make the program wait, but instead it registers a callback and lets the program continue.

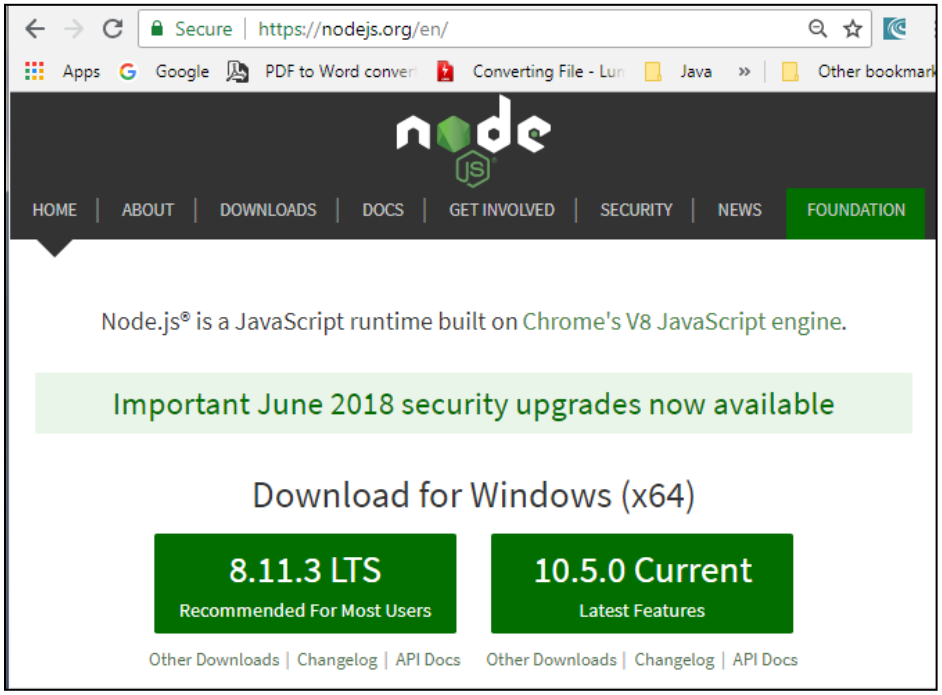
- Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.
- Node.js is great for data-intensive applications.
  - Using a single thread means that Node.js has an extremely low-memory footprint when used as a web server and can potentially serve a lot more requests.
  - Eg, a data intensive application that serves a dataset from a database to clients via HTTP

- **What Node is NOT!**

Node is **not** a webserver. By itself it doesn't do anything. It doesn't work like Apache. There is no config file where you point it to you HTML files. If you want it to be a HTTP server, you have to write an HTTP server (with the help of its built-in libraries). Node.js is just another way to execute code on your computer.
It is simply a JavaScript runtime.

Setting up Node

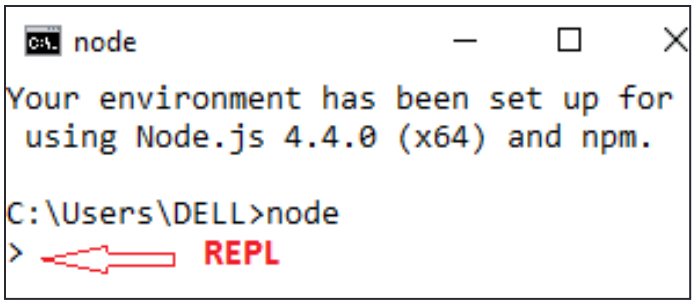
- To install and setup an environment for Node.js :
  - Download the latest version of Node.js installable archive file from https://nodejs.org/en/
  - Double click to run the msi file
  - Verify if the installation was successful : node –v in command window.

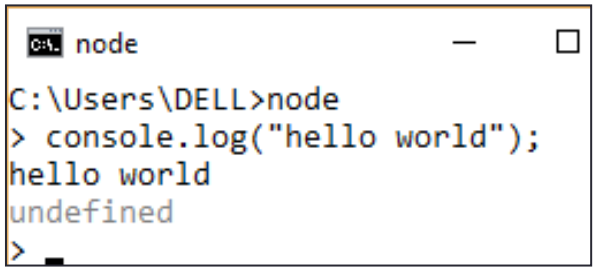


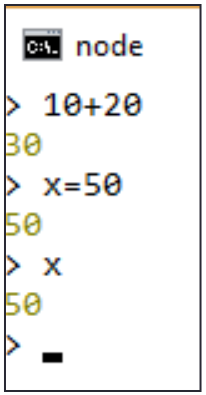

```
C:\Users\Administrator>node -v
v8.11.3
```

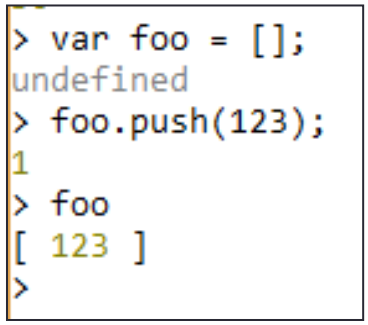
Using the Node CLI : REPL (Read-Eval-Print-Loop)

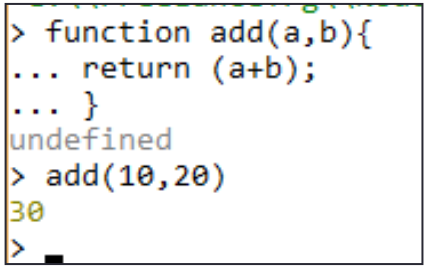
- There are two primary ways to use Node.js on your machines: by using the Node Shell or by saving JavaScript to files and running those.
  - Node shell is also called the Node REPL; a great way to quickly test things in Node.
  - When you run “node” without any command line arguments, it puts you in REPL

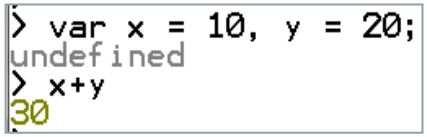












- You can also create a js file and type in some javascript.

```
C:\Users\DELL>node helloworld.js
Hello World!
```

```
//helloworld.js
console.log("Hello World!");
```

Node js Modules

- A module in Node.js is a logical encapsulation of code in a single unit.
  - Since each module is an independent entity with its own encapsulated functionality, it can be managed as a separate unit of work.

- Consider modules to be the same as JavaScript libraries.
  - A set of functions you want to include in your application.
  - Module in Node.js is a simple or complex functionality organized in JavaScript files which can be reused throughout a Node.js application.
  - A module is a discrete program, contained in a single file in Node.js. Modules are therefore tied to files, with one module per file.

- Node.js has a set of built-in modules which you can use without any further installation.
  - Built-in modules provide a core set of features we can build upon.
  - Also, each module can be placed in a separate .js file under a separate folder.
  - To include a module, use the **require()** function with the name of the module.

- In Node, modules are referenced either by file path or by name
  - For example, we can require some native modules:

```
var http = require('http');
var dns = require('dns');
```

```
var myFile = require('./myFile'); // loads myFile.js
```

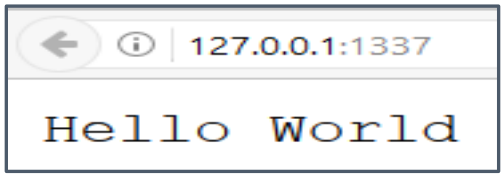
Node.js Web App

```
//RunServer.js
var http = require("http");

function process_request(req, res) {
   var body = 'Hello World\n';
   var content_length = body.length ;
   res.writeHead(200, {
            'Content-Length': content_length,
            'Content-Type': 'text/plain'   });

    res.end(body);
}

var srv = http.createServer(process_request);
srv.listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

Import required module using **require**; load http module and store returned HTTP instance into http variable

createServer() : turns your computer into an HTTP server
Creates an HTTP server which listens for request over 1337 port on local machine

```
G:\FreeLanceTrg\Node.js\Demo\Intro>node runserver.js
Server running at http://127.0.0.1:1337/
```




```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
   })
  .listen(1337, '127.0.0.1');
```

Node.js Module

- Node.js includes three types of modules:
  - Core Modules
  - Local Modules
  - Third Party Modules

- Loading a core module
  - Node has several modules compiled into its binary distribution called core modules.
  - It is referred solely by the module name, not by the path and are preferentially loaded even if a third-party module exists with the same name.
  - var http = require('http');
- Some of the important core modules in Node.js

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods &events to work with file I/O.
util	util module includes utility functions useful for programmers.

Node.js Local Module

- The local modules are custom modules that are created locally by developer in the app
  - These modules can include various functionalities bundled into distinct files and folders
  - You can also package it and distribute it via NPM, so that Node.js community can use it.
  - For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

```
//Module1.js : exporting variable
exports.answer=50;
```

```
//UseModule1.js
var ans=require("./module1");
console.log(ans.answer);
```

exports object is a special ***object*** created by the Node module system which is returned as the value of the require function when you include that module.

```
E:\FreelanceTrg\Node.js\Demo\Modules>node module1.js

E:\FreelanceTrg\Node.js\Demo\Modules>node UseModule1.js
50
```

module2.js > ...

```
1   //exporting function
2   exports.sayHelloInEnglish = function(){
3   return "Hello";
4   };
5   exports.sayHelloInSpanish = function(){
6   return "Hola";
7   };
```

UseModule2.js > ...

```
1   var greetings=require("./module2");
2   console.log(greetings.sayHelloInSpanish());
3   console.log(greetings.sayHelloInEnglish());
4   /*
5   This is equivalent to:
6   var greetings=require("./module2").sayHelloInSpanish();
7   console.log(greetings);
8   */
```

Create Your Own Modules

module1a.js > ...

```
exports.bname = 'Core Node.js';
exports.read = function() {
    console.log('I am reading ' + exports.bname);
}
```

UseModule1a.js > ...

```
var book = require('./module1a.js');
console.log('Book name: ' + book.bname);
book.read();
```

```
Book name: Core Node.js
I am reading Core Node.js
```

- Exports is just module.exports's little helper. Your module returns module.exports to the caller ultimately, not exports. All exports does is collect properties and attach them to module.exports

module2a.js > ...

```
exports.func1 = function() {
    console.log('in function func1()');
};

module.exports.func2 = function() {
    console.log('in function func2()');
};
```

UseModule2a.js > ...

```
const f1 = require('./module2a');
console.log(f1);

f1.func1();
f1.func2();
```

```
{ func1: [Function], func2: [Function] }
in function func1()
in function func2()
```

NPM (Node Package Manager)

- Loading a module(Third party) installed via NPM
  - To use the modules written by other people in the Node community and published on the Internet (**npmjs.com**).
  - We can install those third party modules using the **Node Package Manager** which is installed by default with the node installation.
  - Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application.
  - It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository.
  - NPM is a command line tool that installs, updates or uninstalls Node.js packages in your application.
  - After you install Node.js, verify NPM installation : **npm –v**

```
C:\Users\Shrilata>node -v
v14.16.0

C:\Users\Shrilata>npm -v
6.14.11
```

- Installing Packages
  - In order to use a module, you must install it on your machine.
  - To install a package, type npm install, followed by the package name

NPM (Node Package Manager)

- There are two ways to install a package using npm: globally and locally.
- **Globally** − This method is generally used to install development tools and CLI based packages. To install a package globally, use the following code.
  - **npm install -g <package-name>**
  - Eg to install Typescript : npm install –g typescript
  - Eg to install Angular : npm install -g @angular/cli

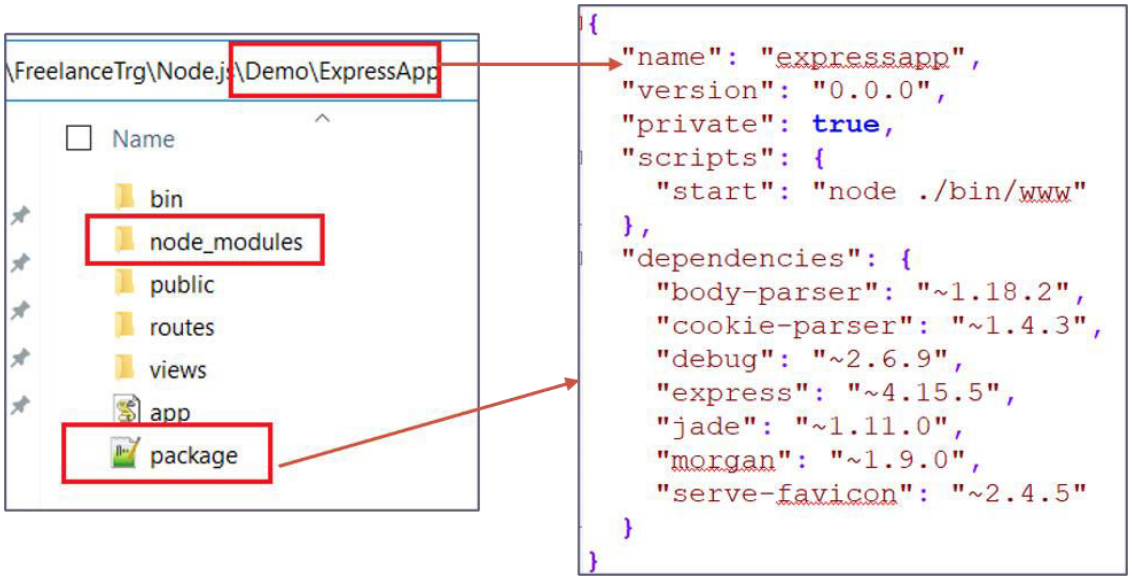
- **Locally** − This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed.
  - To install a package locally, use the same command as above without the -g flag.
  - **npm install <package-name>**
  - Eg to install expressJS : npm install express
  - Eg : To install cookie parser in Express : npm install --save cookie-parser
  - Eg: to install bootstrap : npm install bootstrap@4.1.1

- When packages are installed, they are saved on local machine
- npm installs module packages to the node_modules folder.
  - **Installing a package using NPM :** *$ npm install [g] <Package Unique Name>*
  - **To remove an installed package :** *npm uninstall [g] < Package Unique Name>*
  - **To update a package to its latest version :** *npm update [g] < Package Unique Name>*

Loading a third party module

- Lets say I want to create a ExpressJS application. I will install ExpressJs locally.
  - Step-1) choose a empty folder
  - Step-2) run npm init to create a package.json file
  - Step-3) install express : npm install express –save (to update package.json)
  - Step-4) check the updated json file to see new dependencies

  - See how package.json is installed in root folder along with node_modules folder
  - My Express app is dependent on a number of other modules
  - All these dependencies will have an entry in package.json



Loading a third party module : package.json

- The package.json file in Node.js is the heart of the entire application.
  - It is basically the manifest file that contains the metadata of the project.
  - package.json is a configuration file from where the npm can recognize dependencies between packages and installs modules accordingly.
  - It must be located in project's root directory.
  - It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number and a list of dependencies required by the application.
  - Your project also must include a package.json before any packages can be installed from NPM.
  - Eg : a minimal package.json:

```
{
  "name" : "barebones",
  "version" : "0.0.0",
}
```

    - The name field should explain itself: this is the name of your project. The version field is used by npm to make sure the right version of the package is being installed.

Events Module

- Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.
  - To include the built-in Events module use the require() method.
  - Event module includes EventEmitter class which can be used to raise and handle custom events.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
```

- This object exposes, among many others, the on and emit methods.
  - emit is used to trigger an event
  - on is used to add a callback function that's going to be executed when the event is triggered (bind an event handler with an event)

```
var events = require('events');       // Import events module
var eventEmitter = new events.EventEmitter();   //Create an eventEmitter object

// Bind event to event  handler
eventEmitter.on('eventName', eventHandler);

//fire an event programmatically
eventEmitter.emit('eventName');       // Fire an event
```

SimpleEventEmitterEg1.js > ...

```
1   // Import events module
2   var events = require('events');
3
4   // Create an eventEmitter object
5   var eventEmitter = new events.EventEmitter();
6
7   // Create an event handler as follows
8   var connectHandler = function connected() {
9      console.log('connection succesful.');
0
1      // Fire the data_received event
2      eventEmitter.emit('data_received');
3   }
4
5   // Bind the connection event with the handler
6   eventEmitter.on('connection', connectHandler);
7
8   // Bind the data_received event with the anonymous function
9   eventEmitter.on('data_received', function(){
0      console.log('data received succesfully.');
1   });
2
3   // Fire the connection event
4   eventEmitter.emit('connection');
5
6   console.log("Program Ended.");
```

EVENT NAMES Events are simply keys and can have any string value: data, join, or some crazy long event name except for a one special event, called error

```
connection succesful.
data received succesfully.
Program Ended.
```

Example

example of an event emitter that emits a tick event every second:

```
var EventEmitter = require('events').EventEmitter;
var emitter = new EventEmitter();
var count = 0;
setInterval(function() {
  emitter.emit('tick', count);
  count ++;
  }, 1000);
emitter.on('tick', function(count) {
  console.log('tick:', count);
});
```

```
tick: 0
tick: 1
tick: 2
tick: 3
tick: 4
tick: 5
```

leEventEmitterEg3.js > ...

```
var emitter = require('events').EventEmitter;
function iterateProcessor(num) {
  var emt = new emitter();
  setTimeout(function () {
        for (var i = 1; i <= num; i++) {
          emt.emit('BeforeProcess', i);
          console.log('Processing Iteration:' + i);
          emt.emit('AfterProcess', i);
        }
      }
      , 5000)
  return emt;
}
var it = iterateProcessor(5);

it.on('BeforeProcess', function (info) {
  console.log('Starting the process for ' + info);
});

it.on('AfterProcess', function (info) {
  console.log('Finishing processing for ' + info);
});
```

```
Starting the process for 1
Processing Iteration:1
Finishing processing for 1
Starting the process for 2
Processing Iteration:2
Finishing processing for 2
Starting the process for 3
Processing Iteration:3
Finishing processing for 3
Starting the process for 4
Processing Iteration:4
Finishing processing for 4
Starting the process for 5
Processing Iteration:5
Finishing processing for 5
```

Node.js fs (File System) Module

- The fs module provides a lot of very useful functionality to access and interact with the file system.
  - There is no need to install it. Being part of the Node.js core, it can be used by simply requiring it:
  - const fs = require('fs')
- This module provides a wrapper for the standard file I/O operations.
- All the methods in this module has asynchronous and synchronous forms.
  - synchronous methods in this module ends with 'Sync'. For instance renameSync() is the synchronous method for rename() synchronous method.
  - The asynchronous form always take a completion callback as its last argument.
  - The arguments passed to the completion callback depend on the method, but the first argument is always reserved for an exception. If the operation was completed successfully, then the first argument will be null or undefined.
  - When using the synchronous form any exceptions are immediately thrown. You can use try/catch to handle exceptions or allow them to bubble up.

Asynchronous method is preferred over synchronous method because it never blocks the program execution where as the synchronous method blocks.

Node.js File System

- Node fs module provides an API to interact with FileSystem and to perform some IO operations like create a file, read a File, delete a File etc..
  - fs module is responsible for all the async or synchronous file I/O operations.

```
var fs = require('fs');
// write
fs.writeFileSync('test.txt', 'Hello fs!');
// read
console.log(fs.readFileSync('test.txt')); //<Buffer 48 65 6c 6c 6f 20 66 73 21>
console.log(fs.readFileSync('test.txt').toString());   //Hello fs!

console.log(fs.readFileSync('test.txt','utf8'));  //Hello fs!
```

```
var fs = require("fs");

// Asynchronous read
fs.readFile('test.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('test.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

```
Synchronous read: Hello fs!
Program Ended
Asynchronous read: Hello fs!
```

```
var fs = require('fs');
fs.writeFile('test.txt', 'Hello World!', function (err) {
        if (err)
           console.log(err);
        else
           console.log('Write operation complete.');
});
```

Node.js File System

- fs.readFile(fileName [,options], callback) : read the physical file asynchronously.
- fs.writeFile(filename, data [, options], callback) : writes data to a file
- fs.appendFile(): appends the content to an existing file
- fs.unlink(path, callback); delete an existing file

```
var fs = require("fs")
fs.unlink("test1.txt", function(err){
    if(err) console.log("Err : " , err);
    console.log("delete successful")
 })
```

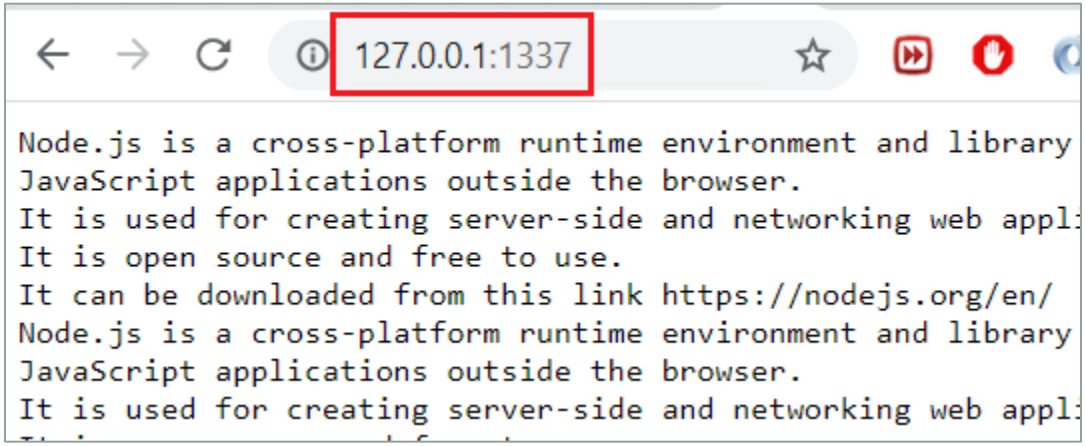
- fs.close(fd, callback(err));
- fs.rename(oldPath, newPath, callback): rename a file or folder

```
fs.rename("src.json", "tgt.json", err => {
    if (err) {
      return console.error(err)
    }
    console.log('Rename operation complete.');
  });
```

Example

- Reading a file at server end and serving to browser!

```
var http = require('http');
var fs = require('fs');
function process_request(req, res) {
   fs.readFile('bigsample.txt', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.write(data);
      res.end();
   });
}
var s = http.createServer(process_request);
s.listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```



Web development with Node : http.ServerRequest

- When listening for request events, the callback gets an http.ServerRequest object as the first argument (function(req,res))
- This object contains some properties:
  - req.url: This property contains the requested URL as a string
    - It does not contain the schema, hostname, or port, but it contains everything after that.
    - Eg : if URL is :http://localhost:3000/about?a=20 then req.url will return /about?a=20
  - req.method: This contains the HTTP method used on the request. It can be, for example, GET, POST, DELETE, or HEAD.
  - req.headers: This contains an object with a property for every HTTP header on the request.

  - Eg : Serving file on request : Reading a file at server end and serving to browser!

Routing

- Routing refers to the mechanism for serving the client the content it has asked

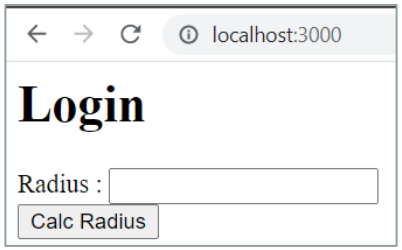
```
var http = require('http');
var server = http.createServer(function(req,res){
    //console.log(req.url);
var path = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
switch(path) {
    case '' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>Home Page</h1>');
        break;
    case '/about' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>About us</h1>');
        break;
    case '/admin' :
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<h1>Admin page</h1>');
        break;
    default:
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Not Found');
        break;
    }
});
server.listen(3000);
```

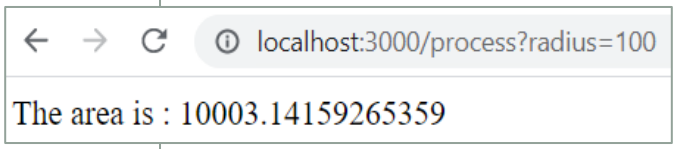
```
var http = require("http");
http.createServer(function(request, response) {
  if (request.url === "/" && request.method === "GET") {
       response.writeHead(200, { "Content-Type": "text/html" });
       response.end("Hello <strong>home page</strong>");
  }
   else if (request.url === "/foo" && request.method === "GET") {
       response.writeHead(200, { "Content-Type": "text/html" });
       response.end("Hello <strong>foo</strong>");
  }
  else if (request.url === "/bar" && request.method === "GET") {
       response.writeHead(200, { "Content-Type": "text/html" });
       response.end("Hello <strong>bar</strong>");
  }
  else {
       response.writeHead(404, { "Content-Type": "text/html" });
       response.end("404 Not Found");
}
}).listen(8000);
```

```
var http = require('http');
var url = require('url');
var fs = require('fs');

function process_req(req, res) {
  if (req.method == 'GET' && req.url == '/') {
    fs.readFile('radius.html', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
   });
  }
  else if(req.method == 'GET' && req.url.substring(0,8) == '/process'){
     var q = url.parse(req.url, true);
     var qdata = q.query;
     var r = qdata.radius;

     var rad = Math.PI + r * r;
     res.writeHead(200, {'Content-Type': 'text/html'});
     res.write("The area is : " + rad);
     res.end();
   }
   else
     res.end("not found");
}
var server = http.createServer(process_req)
server.listen(3000);
console.log('server listening on localhost:3000');
```

{ radius: '100' }

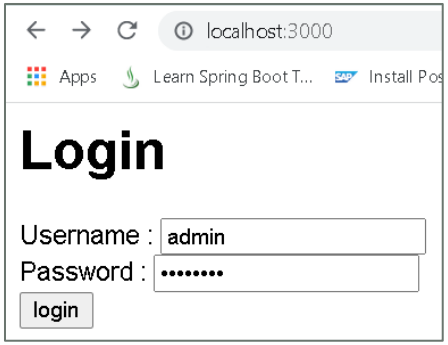
```
<html>
<body>
  <h1>Login</h1>
   <form action="process">
    Radius : <input name="radius">
   <input type="submit"
          value="Calc Radius">
   </form>
</body></html>
```

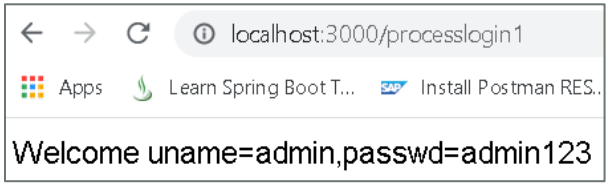




```
var http = require('http');
var fs = require('fs');

function process_req(req, res) {
  if (req.method == 'GET' && req.url == '/') {
     fs.readFile('loginPost.html', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
   });
  }
  else if(req.method == 'POST'){
     var body = "";
     req.on("data",function(data){
        body += data;
        res.writeHead(200, {'Content-Type': 'text/html'});
        var arr = body.split("&");
        res.write("Welcome " + arr);
        res.end();
      })
  }
}
var server = http.createServer(process_req)
server.listen(3000);
console.log('server listening on localhost:3000');
```

```
<body>   //loginpost.html
  <h1>Login</h1>
   <form action="processlogin"
         method="post">
    Username :
      <input name="uname"><br>
    Password :
      <input type="password"
             name="passwd"><br>
      <input type="submit"
             value="login">
   </form>
</body>
```





Util module

- **util** module provides "utility" functions that are helpful to a developer but don't really belong anywhere else.
  - It provides useful functions such as string formatting, debugging, logging, and some type comparers for JS objects such as Array or Error, which returns just 'object' when using the typeof keyword.

```
var txt = 'Congratulate %s on his %dth birthday!';
var result = util.format(txt, 'Arnav', 18);
console.log(result); /Congratulate Arnav on his 18th birthday!
```

% - Returns a single percent sign.

d - Treated as Number (both integer and float).

s - Treated as string and display as a string.

j - Represent JSON data.

```
var util = require('util');
empName = 'Anita'
emailId = "anita@gmail.com"
age = 37
project = { bu: 'FSBU', client: 'Bank of scotland'};
var str1 = util.format('Emp name is %s with email id %s',empName, emailId);
var str2 = util.format('Emp age %d,',age);
var str3 = util.format('Project details %j,',project);

console.log(str1);
console.log(str2);
console.log(str3);
```

```
Emp name is Anita with email id anita@gmail.com
Emp age 37
Project details {"bu":"FSBU","client":"Bank of scotland"}
```

```
console.log(util.isArray([]));          //true
console.log(util.isArray(new Array)); //true
console.log(util.isArray({}))           //false
```

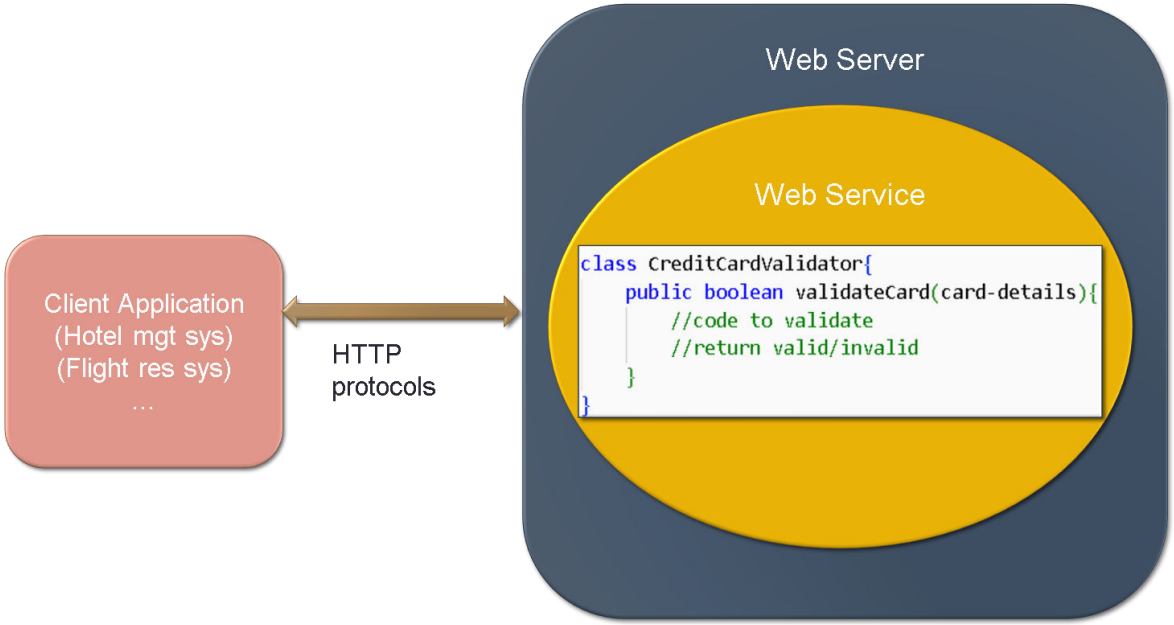
```
util.isDate(new Date()) //true
util.isDate(Date())     //false
```

OS module

- os module provides a few basic operating-system related utility functions

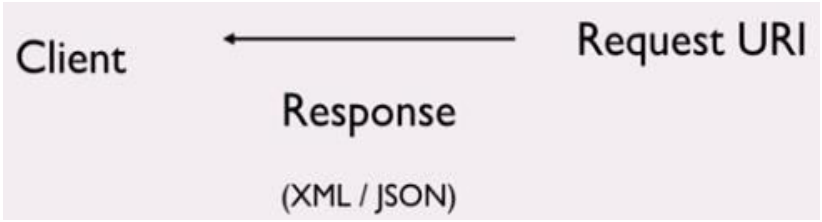
```
var os = require('os');
console.log("Platform: " + os.platform());   //win32
console.log("Architecture: " + os.arch());  //x64
console.log("Free memory (in bytes): " + os.freemem());  //1030639616
console.log("Host name: " + os.hostname());  //LAPTOP-CG2KSI6E
console.log("Total memory (in bytes): " + os.totalmem());  //8471760896
console.log("Name of OS: " + os.type());  //Windows_NT
console.log("Current user: " + os.userInfo());  //x64
```

REST – REpresentational State Transfer

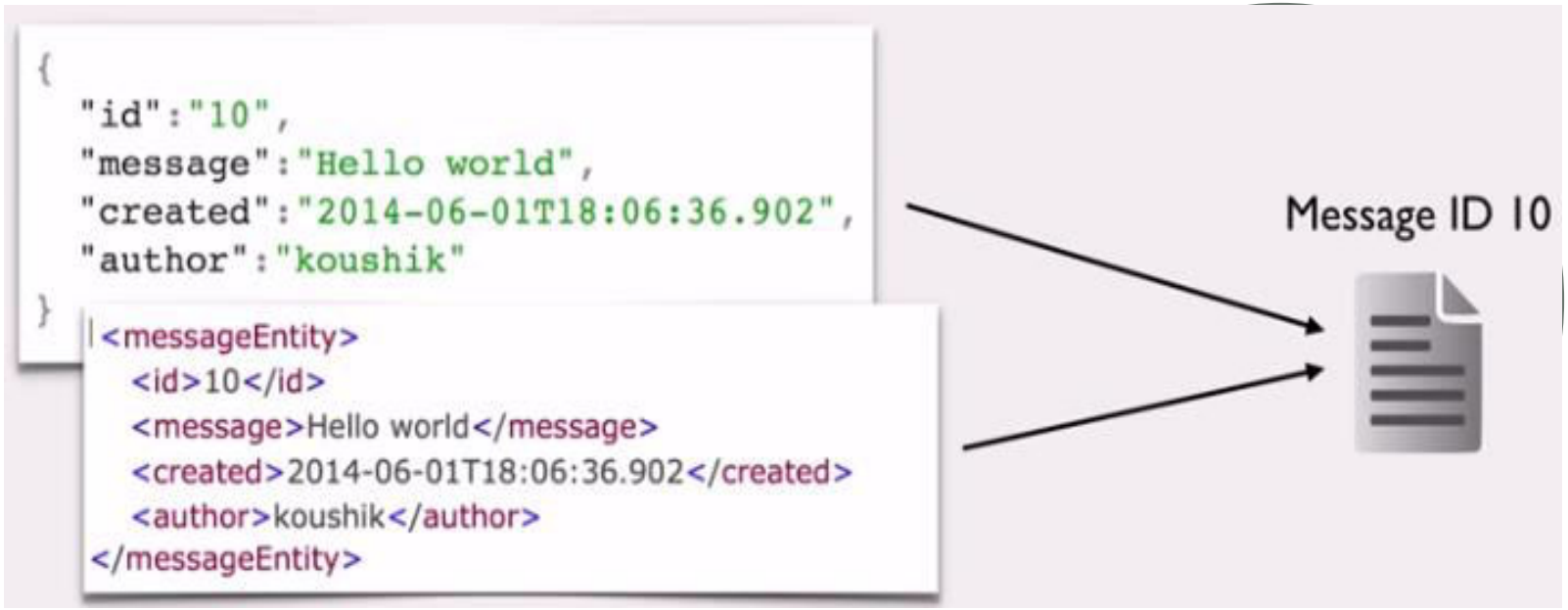


“a service made available over the web”

REST Response




```
public class MessageEntity {
    private long id;
    private String message;
    private Date created;
    private String author;
...
}
```



EXPRESS.JS

Node.js web application framework

Introduction

- If you write serious apps using only core Node.js modules you most likely find yourself reinventing the wheel by writing the same code continually for similar tasks, such as the following:
  - Parsing of HTTP request bodies and Parsing of cookies
  - Managing sessions
  - Organizing routes with a chain of if conditions based on URL paths and HTTP methods of the requests
  - Determining proper response headers based on data types

Express 4.17.1
Fast, unopinionated, minimalist web framework for Node.js

- Express is a web application framework for Node
  - Its built on top of Node.js.
  - It provides various features that make web application development fast and easy compared to Node.js.

Express.js Installation

- Create a new folder: eg E:\Express-app
- E:\Express-app>npm init //for creating package.json
- E:\Express-app>npm install express --save

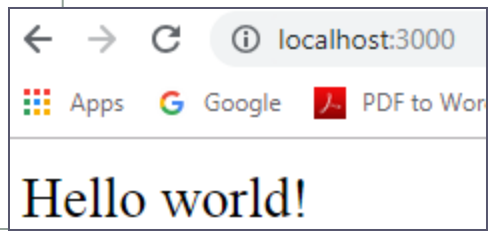
```
{
  "name": "express-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

First app

- An Express app is created by calling the express() function
  - express() : Creates an Express application; is a top-level function exported by the express module
  - The app object conventionally denotes the Express application.
  - This object, which is traditionally named app, has methods for routing HTTP requests, configuring middleware, rendering HTML views, registering a template engine, and modifying application settings that control how the application behaves

```
let express = require("express")  // import the express module

let app = express()  // create an Express application

app.get("/",function(req,resp){
    resp.send("Hello world")
})

app.listen(3000, function(){
    console.log("app running on port 3000")
})
```

The *app.listen()* method returns an http.Server object



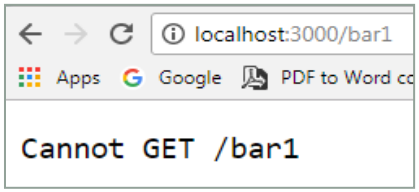
Express Routes

- HTTP verb and URL combinations are referred to as routes, and Express has efficient syntax for handling them.

```
var express = require("express");
var http = require("http");
var app = express();

app.get("/", function(req, res, next) {
      res.send("Hello <strong>home page</strong>");
});

app.get("/foo", function(req, res, next) {
      res.send("Hello <strong>foo</strong>");
});

app.get("/bar", function(req, res, next) {
     res.send("Hello <strong>bar</strong>");
});
//app is passed to the http.createServer() method

http.createServer(app).listen(8000);
```

Open browser and type:
http://localhost:3000
http://localhost:3000/foo
http://localhost:3000/bar
http://localhost:3000/admin - gives error

Express Routes



- If none of your routes match the request, you'll get a ***"Cannot GET <your-request-route>"*** message as response.
- This message can be replaced by a 404 not found page using this simple route

```
app.get('*', function(req, res){
    res.send('Sorry, this is an invalid URL.');
});
```

```
app.get('/course/:id', function(req, res){
    var course = //code to retrieve course
    If(!course){
        res.status(404).send("course not found");
});
```

```
app.get("/", function(req, res, next) {
    res.send("Hello <strong>home page</strong>");
});
```

- The get() method defines routes for handling GET requests.
- Express also defines similar methods for the other HTTP verbs (put(), post(), delete(), and so on).
  - app.method(path, handler) : This METHOD can be applied to any one of the HTTP verbs – get, post, put, delete.
  - The path is a string or regular expression representing the URL that the route responds to. Note that the query string is not considered part of the route's URL.
- Also notice that we haven't defined a 404 route, as this is the default behavior of Express when a request does not match any defined routes.

Routing basics

```
app.get("/", function(req, res, next) {
      res.send("Hello <strong>home page</strong>");
});
```

- Express also augments the request and response objects with additional methods. Example response.send() .
  - send() is used to send a response status code and/or body back to the client.
  - If the first argument to send() is a number, then it is treated as the status code. If a status code is not provided, Express will send back a 200.
  - The response body can be specified in the first or second argument, and can be a string, Buffer, array, or object.

- send() also sets the Content-Type header unless you do so explicitly.
  - If the body is a string, Express will set the Content-Type header to text/html.
  - If the body is an array or object, then Express will send back JSON.
  - If the response body is a Buffer, the Content-Type header is also set to application/octet-stream

Routing basics

```
var express = require("express");
var app = express();
var path = require('path');

app.get("/buff", function(req, res, next) {
    var buff = Buffer.from("Hello World");
    res.send(buff.toString());
});
app.get("/string", function(req, res, next) {
    res.send("Hello <strong>String response</strong>");
});
app.get("/json", function(req, res, next) {
     res.send({name:'Soha',age:23});
});
app.get("/array", function(req, res, next) {
     res.send(['NodeJS','Angular','ExpressJS']);
});
app.get("/file", function(req, res, next) {
    res.sendFile(path.join(__dirname + '/summer.html'));
});
app.listen(3000);
```

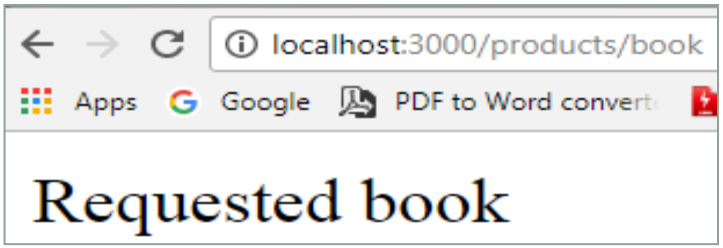
Route Parameters

- Route can be parameterized using a regular expression

```
var express = require("express");
var http = require("http");
var app = express();
app.get(/\/products\/([^\/]+)\/?$/, function(req, res, next) {
    res.send("Requested " + req.params[0]);
});
http.createServer(app).listen(8000);
```

Ignore multiple slash

/products?productId=sweater
/products/sweater



The above regular expression matches anything but /
/products/books
/products/books:aaa
/products/books/
/products/books?aaa
/products/books=aaa

Doesn't match:
/products/books/aaa
/products/books//
/products//

Route Parameters

- One of the most powerful features of routing is the ability to use placeholders to extract named values from the requested route, marked by the colon (:) character.
  - When the route is parsed, express puts the matched placeholders on the req.params object for you.

```
app.get('/user/:id', function(req, res) {
      res.send('user ' + req.params.id);
});

app.get('/product/:prodname', function(req, res) {
      res.send('Product :  ' + req.params.prodname);
});
```

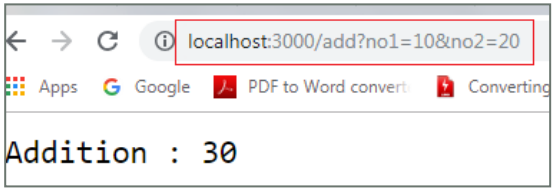
Placeholders match any sequence of characters except for forward slashes.

Working with parameters using get

- **req.query**
  - Express parses query string parameters by default, and puts them into the req.query property.
  - If the request is GET /search?username=Shrilata
    req.query.username returns “Shrilata"

  - Lets say the incoming url is : http://localhost:3000/add?no1=10&no2=20
  - Use req.query to query the request parameters

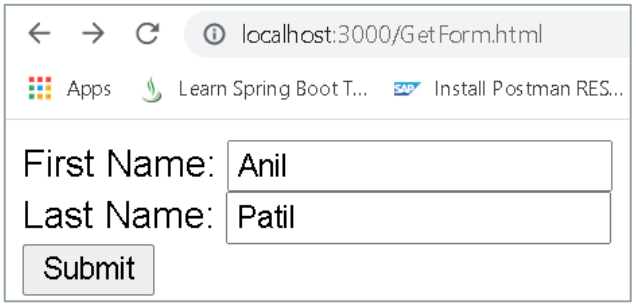
```
app.get("/add", function (req, res) {
    n1 = parseInt(req.query.no1);
    n2 = parseInt(req.query.no2);
    res.end("Addition : " +   (n1 + n2) );
});
```

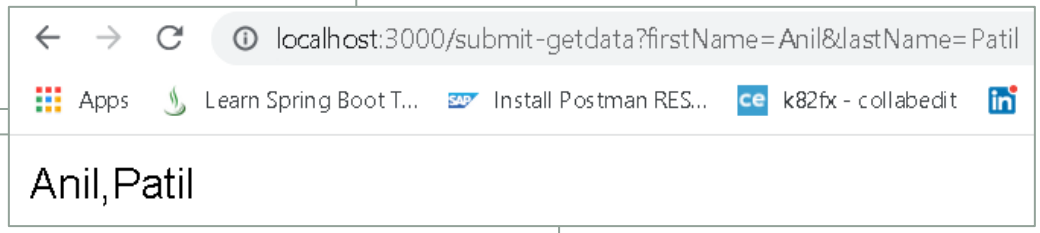


Handle GET Request

GetForm.html > ...

```
<!DOCTYPE html>
<html>
<body>
    <form action="/submit-getdata" method="get">
        First Name: <input name="firstName" type="text" /
        Last Name: <input name="lastName" type="text" />
        <input type="submit" />
    </form>
</body>
</html>
```






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

app.get('/GetForm.html', function (req, res) {
    res.sendFile('public/GetForm.html' , { root : __dirname});
});

app.get('/submit-getdata', function (req, res) {
    console.log(req.query.firstName);
    console.log(req.query.lastName);
    res.send(req.query.firstName+","+req.query.lastName);
});
app.listen(3000);
```

Multiple different methods

- We can also have multiple different methods at the same route.

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

app.get('/hello', function(req, res){
   res.send("Hello World!");
});

app.post('/hello', function(req, res){
   res.send("You just called the post method at '/hello'!\n");
});

app.delete('/hello', function(req, res){
   res.send("You just called the delete method at '/hello'!\n");
});

app.listen(3000);
```

Handle POST Request

- To handle HTTP POST request in Express.js version 4 and above, you need to install a middleware module called body-parser.
  - This is used to parse the body of requests which have payloads attached to them.
  - Install it using : **npm install** --save body-parser
  - Mount it by including the following lines in your js

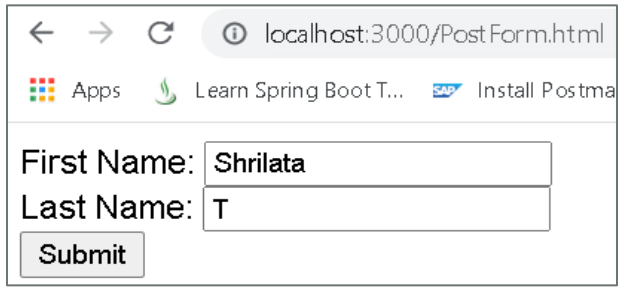
```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
```

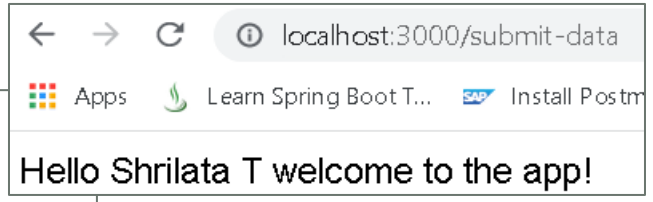
  - This body-parser module parses the JSON, buffer, string and url encoded data submitted using HTTP POST request.
  - Eg :To parse json data: app.use(bodyParser.json())

bodyParser.urlencoded(): Parses the text as URL encoded data (which is how browsers tend to send form data from regular forms set to POST) and exposes the resulting object (containing the keys and values) on **req.body**

Handle POST Request

```html
<!DOCTYPE html>
<html>  <!-- PostForm.html -->
<body>
    <form action="/submit-data" method="post">
        First Name: <input name="firstName" type="text" />
        Last Name: <input name="lastName" type="text" />
        <input type="submit" />
    </form>
</body>
</html>
```





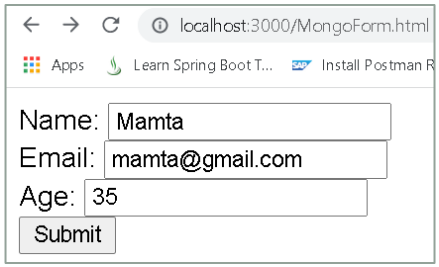

```node
var express = require('express');
var app = express();
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/PostForm.html', function (req, res) {
    //res.sendFile('E:\\Node.js\\Demo\\expressPrj\\PostForm.html');
    res.sendFile('public/PostForm.html' , { root : __dirname});
});

app.post('/submit-data', function (req, res) {
    var name = req.body.firstName + ' ' + req.body.lastName;
    res.send("Hello " + name + ' welcome to the app!');
});
app.listen(3000);
```

Mongo

- In order to access MongoDB database, we need to install MongoDB drivers. To install native mongodb drivers using NPM :
- Install MongoDB : npm install mongodb –save
- Import mongodb module (native drivers) and get the reference of MongoClient object : var MongoClient = require('mongodb').MongoClient;
- Connect to MongoDB through the MongoClient's connect() method

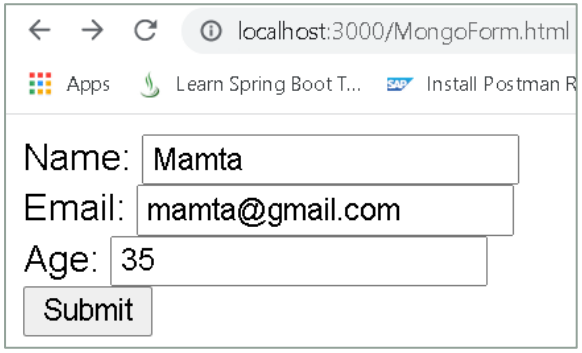
```
var express = require('express');
var app = express();

var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/trgdb';
var db;
MongoClient.connect(url, function(err, dbase) {
      if (err) return console.log(err)
      db = dbase;
});

app.get('/MongoForm.html', function (req, res) {
     res.sendFile('public/MongoForm.html' , { root : __dirname});
});
```

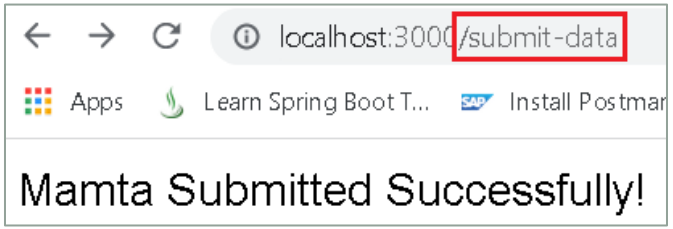
```
<!DOCTYPE html>
<html>
<body>
    <form action="/submit-data" method="post">
        Name: <input name="name" type="text" />
        Email: <input name="email" type="email"
        Age: <input name="age" type="number" />
        <input type="submit" />
    </form>
</body>
```



Mongo

```
app.post('/submit-data', function (req, res) {
    res.send(req.body.name + ' Submitted Successfully!');
    db.collection('userdata').save(req.body, function(err, result){
        if (err) return console.log(err);
        console.log('saved to database');
  });
});

app.get('/GetDetails', function (req, res) {
   db.collection('userdata').find().toArray((err, results) => {
     res.send(results);  //result is a json object
   });
});
app.listen(3000);
```



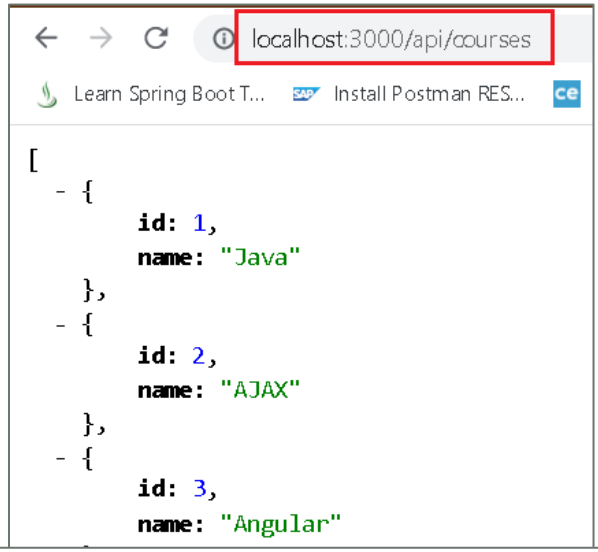


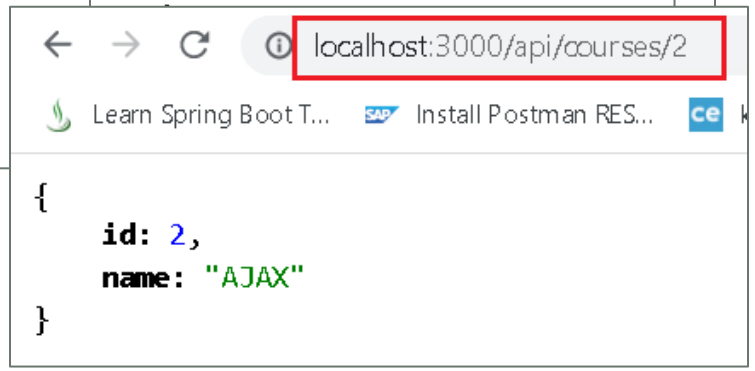
REST with Express

- npm install body-parser –save
  - body-parser middleware helps us decode the body from an HTTP request:It parses the body of the request and lets us react to it
- npm install cors –save
  - cross-origin resource sharing : Since we are calling the API from different locations by hitting endpoints in the browser. We also have to install the CORS middleware.

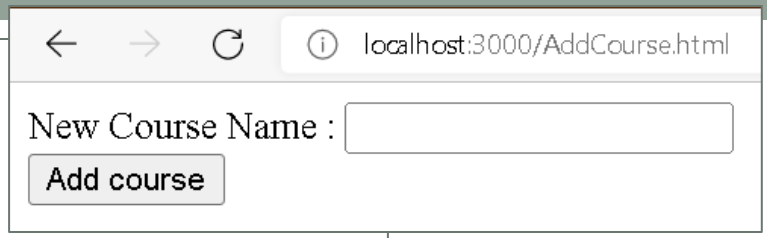
```
var express = require('express');
var app = express();
const bodyParser = require('body-parser');  //parses the body of the request and
lets us react to it
const cors = require('cors');   //cross-origin resource sharing

// Configuring body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

let courses = [{id:1,name:'Java'},
               {id:2,name:'AJAX'},
               {id:3,name:'Angular’}
              ];
```

```
app.get('/', function(req, res){
   let str = "<h1>Hello World </h1>"
   str += "Do you want to <a href='AddCourse.html'>Add new course</a>"
   res.send(str);
});

app.get('/AddCourse.html', function(req, res){
   res.sendFile('AddCourse.html' , { root : __dirname});
});

app.get('/api/courses', function(req, res){
   res.send(courses);
});

//logic to look for  a course with given id
app.get('/api/courses/:id', function(req, res){
   var course;
   for(i in courses){
      if(courses[i].id == parseInt(req.params.id)){
         console.log("found ");
         course = courses[i];
      }
   }
   res.send(course);
});
```





```
app.post('/api/courses',function(req,res){
      let obj = req.body
      obj.id = courses.length+1
      courses.push(obj);
      res.send("Added course : " + JSON.stringify(obj));
});

app.delete('/api/courses/:id', function(req, res){
   var course, index=0;
   for(c in courses){
      if(courses[c].id == parseInt(req.params.id)){
         console.log("course found for delete");
         course = courses[c];
         index++;
      }
   }
   courses.splice(index-1,1)

   res.send("course with id " + req.params.id + "  deleted");
});
app.listen(3000);
```




```
<body>
   <form method="post" action="api/courses">
       New Course Name : <input name="name"><br>
       <input type="submit" value="Add course">
   </form></body>
```

Method	URL endpoint	what it does
get	localhost:3000	fetch home page
get	localhost:3000/AddCourse.html	fetch AddCourse.html
get	localhost:3000/api/courses	fetch all courses
get	localhost:3000/api/courses/2	fetch course with given ID
post	localhost:3000/api/courses	create course
delete	localhost:3000/api/courses/2	delete course with given Id

express-validator

- Express Validator is an Express middleware library that you can incorporate in your apps for server-side data validation.
- We must provide server-side validation when building applications - especially client-facing applications.
  - User's inputs sometimes contain unintentional mistake / malicious data.
  - The client-side validation can be deciphered, or data can be manipulated by just turning off the JavaScript in the browser.
  - Client-side validation is a great way to sift through most of the input, but you still need to perform server-side validation as well.
  - There are many ways to validate data in Node.js. Eg express-validator library, Joi validation library, hapi etc

express-validator

- Step-1:

```
npm install express-validator      //Install express-validator:
npm install body-parser       //to parse body of a POST request
```

- Step-2: Import check, validationResult class from the express-validator module

```
const { check, validationResult } = require('express-validator');
```

  - check() and validationResult() are methods available via require('express-validator')
  - check() : creates a validation chain for one or more fields.
  - validationResult() : returns an object containing all validation errors; makes them available in a Result object.

- Step-3 : Now, to validate your form input, you need to pass an array in which you specify the fields that you want to validate as a second argument to your route handler for /users POST requests

```
app.post("/someurl",[<fields to validate>],(req, resp) => {...})
```

```
app.post("/form",[
            check('name').isLength({ min: 3 }),
            check('email').isEmail(),
            check('age').isNumeric()
           ],
          (req, resp) => {})
```

express-validator

https://github.com/validatorjs/validator.js#validators

```
app.post("/form",[
        check('name')
            .isLength({ min: 3 })
            .withMessage('Name must be greater than 3 characters'),
        check('password')
            .isLength({ min: 8, max: 10 })
            .withMessage('Password length should be 8 to 10 characters')
            .matches('[0-9]').withMessage('Password Must Contain a Number')
            .matches('[A-Z]').withMessage('Password Must Contain an Uppercase Letter'),
        check('email').isEmail(),
        check('age').isNumeric()
    ],
    (req, resp) => {
     const errors = validationResult(req)
     if (!errors.isEmpty()) {
       return resp.status(422).json({ errors: errors.array() })
     }

     const name  = req.body.name
     const password = req.body.password
     const email = req.body.email
     const age   = req.body.age
    })
```

We pass an array of check() calls as the second argument of the post() call.

Every check() call accepts the parameter name as argument.

Then we call validationResult() to verify there were no validation errors; If there are any, we tell them to the client

TYPESCRIPT

TypeScript

- Typescript : is a free and open-source programming language developed by Microsoft
  - Is a **typed** superset of JavaScript
  - Transpilation compiles TypeScript to JavaScript
  - Is object oriented with classes, interfaces and statically typed
  - “TypeScript is JavaScript for application-scale development.”
  - Provides data types and strong typing
  - It is portable as it runs on any browser, any host and device

Getting Started

- To write, compile and run typescript code.
  - Install NodeJS, followed by TypeScript into local systems
  - Install TypeScript as follows:
  - npm install –g typescript (-g installs typescript so that it is accessible globally across all applications on this comp)
  - Node version 4.6.x or greater, npm 3.x.x or greater
  - To check version of node and npm installed : node –v and npm –v and tsc –v

```
C:\Users\Shrilata>node -v
v12.18.4

C:\Users\Shrilata>npm -v
6.14.6

C:\Users\Shrilata>tsc -v
Version 3.9.7
```

  - Create a .ts file for first TS application:
  - Compile: tsc helloworld.ts
  - Run : node helloworld.js

```
//helloworld.ts
var message:string = "Hello World"
console.log(message)
```

```
E:\FreeLanceTrg\Angular2>tsc helloworld.ts

E:\FreeLanceTrg\Angular2>node helloworld.js
Hello World
```

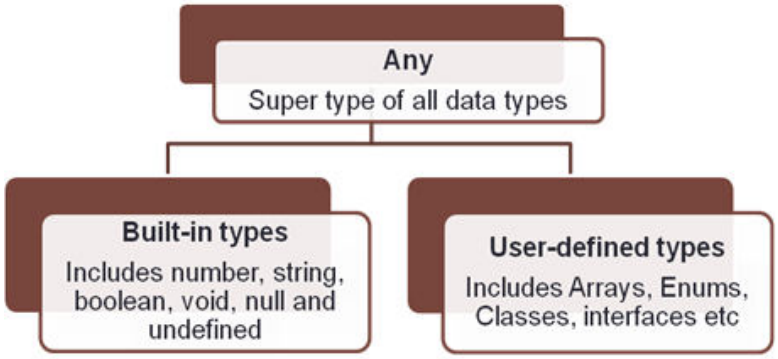
TypeScript Basic Syntax

- Identifiers are names given to elements in a program like variables, functions etc. The rules for identifiers are:
  - Can include both, characters and digits. However, cannot begin with a digit.
  - Cannot include special symbols except for underscore (_) or a dollar sign ($).
  - Cannot be keywords.
  - Must be unique and cannot contain spaces
  - Are case-sensitive.
  - **Valid :** firstName, first_name, num1, $result
  - **Invalid** : var, first name, first-name , 1number
- Semicolons are optional:
- Comments
  - **Single-line comments (//)** − Any text between a // and the end of a line
  - **Multi-line comments (/* */)** − These comments may span multiple lines.

```
//this is single line comment
/* This is a
   Multi-line comment   */
```

TypeScript Types

- Any type : is the super type of all types in TypeScript; denotes a dynamic type.
  - Using the any type is equivalent to opting out of type checking for a variable



Data type	Description
number	Double precision 64-bit floating point values. It can be used to represent both, integers and fractions.
string	Represents a sequence of Unicode characters
boolean	Represents logical values, true and false
void	Used on function return types to represent non-returning functions
null	Represents an intentional absence of an object value.
undefined	Denotes value given to all uninitialized variables

- Declare a variable by using the **var** keyword:
- var identifier : [type-annotation] = value ;
- When you declare a variable, you have four options −

1. Declare its type and value in one statement. var name:string = ”mary”
2. Declare its type but no value. In this case, the variable will be set to *undefined*.
   var name:string;
3. Declare its value but no type.
   var name = ”mary” //The type is inferred from the value. Here, type string
4. Declare neither value not type. In this case, the data type of the variable will be **any** and will be initialized to *undefined*. var name;

TypeScript Types

- Array: Egs
  - var jobs: Array<string> = ['IBM', 'Microsoft', 'Google'];
  - var jobs: string[] = ['Apple', 'Dell', 'HP'];
  - We specify the type of the items in the array with either the Array<type> or type[ ] notations

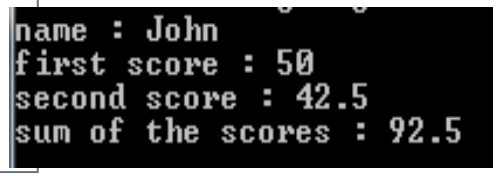
- Any : is the default type if we omit typing for a given variable.
  - Having a variable of type any allows it to receive any kind of value

```
var something: any = 'as string';
something = 1;
something = [1, 2, 3];
```

Examples

- TypeScript will try to infer as much of the type information as it can in order to give you type safety with minimal cost of productivity during code development.
- Eg : var foo = 123;
- foo = '456'; // Error: cannot assign `string` to `number`

```
var pname:string = "John";
var score1:number = 50;
var score2:number = 42.50
var sum = score1 + score2
console.log("name : "+ pname)
console.log("first score: "+ score1)
console.log("second score: "+ score2)
console.log("sum of the scores: "+ sum)
```




```
var any1;    // any value. same as not having a static type
var num1: number; // number type
num1 = 1;  // set after the fact.
var num2: number = 2;                  // initialized and typed
var num3 = 3;              //typed as a number via type inference
var str1 = num1 + 'some string';
console.log(typeof(str1))                //string
```

TypeScript Operators

- Arithmetic operators (+,-,*,/,%)
- Assignment operators (=,+=, -=, /=, *=, %=)
- Comparison operators (==, !=, < <=, > >=)
- Boolean operators (&&, ||, !)
- Bitwise operators (&, |, !, ^, >>, >>>)
- String operators (=, +, +=)
- Ternary/conditional operator (Test ? expr1 : expr2).
  - Eg var result = num > 0 ?"positive":"negative"
- Type Operator (typeof)

```
var num = 12
console.log(typeof num);   //output: number
```

- Instanceof : used to test if an object is of a specified type or not (more later)

Language constructs

```
if(boolean_expression) {
   // statements
}

if(boolean_expression) {
   // statements
} else {
   // statements
}
```

```
//example
var num:number = 12;
if (num % 2==0) {
   console.log("Even");
} else {
   console.log("Odd");
}
```

```
switch(variable_expression) {
   case constant_expr1: {
      //statements;  break;
   }
   case constant_expr2: {
      //statements;  break;
   }
   default: {
      //statements;  break;
   }
}
```

```
//example
var grade:string = "A";
switch(grade) {
   case "A": {
      console.log("Excellent");
      break;
   }
   case "B": {
      console.log("Good");
      break;
   }
default: {
      console.log("Invalid ");
      break;
   } }
```

Language constructs

```
for (initialvalue; condition; step) {
   //statements
}
```

```
// for...in loop
for (var val in array/tuple/collection) {
   //statements
}
```

```
while(condition) {
   // statements
}
```

```
do {
   //statements
} while(condition)
```

```
var num:number = 5;
var i:number;
var factorial = 1;

for(i = num;i>=1;i--) {
    factorial *= i;
}
console.log(factorial)   //120
```

```
var j;
var nums = [1001,1002,1003]
for(j in nums) {
    console.log(j)
}

for(j of nums) {
    console.log(j)
}
```

```
0
1
2
1001
1002
1003
```

```
var subjects = ["Java", "TypeScript", "Angular"];
for (var sub of subjects) // Use iterator
        console.log(sub);
console.log("Top Element : " + subjects.pop());
```

```
Java
TypeScript
Angular
Top Element : Angular
```

TypeScript Functions

```
function fname()[:return_type] {
      //statements
      [return value;]
}
```

```
function fname( param1 [:datatype],
                param2 [:datatype]) {
}
```

```
function greetText(name: string): string {
    return "Hello " + name;
}
console.log("Shrilata");
```

```
Hello shrilata
```

```
function greet():string {  //function returns a string
   return "Hello World"
}

function caller() {   //func with no args no return
     var msg = greet()    //function greet() invoked
     console.log(msg)
}
caller()    //invoke function
```

```
//parameterized functions
function test(n1:number, s1:string) {
   console.log(n1)
   console.log(s1)
}

test(123,"a string")
```

```
123
a string
```

```
//optional parameters
function disp(id:number,   name:string,   email?:string) {
   console.log("ID:", id);
   console.log("Name",name);

   if(email!=undefined)
   console.log("Email Id",email);
}
disp(123,"John");
disp(111,"mary","mary@xyz.com");
```

```
ID: 123
Name John
ID: 111
Name mary
Email Id mary@xyz.com
```

```
//Default Parameters
 function calc(price:number, rate:number = 0.50) {
   var discount = price * rate;
   console.log("Discount : ",discount);
}
calc(1000)
calc(1000,0.30)
```

```
Discount :  500
Discount :  300
```

Rest parameters

- Rest parameters (...argumentName for the last argument) allow you to quickly accept multiple arguments in function and get them as an array.

```
function iTakeItAll(first, second, ...allOthers) {
    console.log(allOthers);
}
iTakeItAll('foo', 'bar'); // []
iTakeItAll('foo', 'bar', 'bas', 'qux'); // ['bas','qux']
```

```
function addNumbers(...nums:number[]) {
   var i;
   var sum:number = 0;

   for(i = 0;i<nums.length;i++) {
        sum = sum + nums[i];
   }
   console.log("sum of the numbers",sum)
}
addNumbers(1,2,3)
addNumbers(10,10,10,10,10)
```

Arrays : Ways of creating array:

- var array_name[:datatype]; //declaration
- array_name = [val1,val2,valn..] //initialization
- var array_name[:data type] = [val1,val2…valn] //declaration+initialization
- var arr_name:data_type[] = new Array(size) //using the Array object
- var arr_name:data_type[] = new Array(val1, val2…) //comma separated values

```
var arr1:string[];
arr1= ["1","2","3","4"]
console.log(arr1[0]);  //1
```

```
var nums:number[] = [1,2,3,3]
console.log(nums[0]);  //1
```

```
var names:string[] = new
Array(“Joy",“Roy",“Leo”)
for(var i = 0;i<names.length;i++) {
   console.log(names[i])
}
```

```
var arr2:number[] = new Array(4)
for(var i = 0;i<arr2.length;i++) {
   arr2[i] = i * 2 ;
   console.log(arr2[i])
}
```

```
var arr:Array<number> = [1,2,3,4]; // using generics
```

- You can pass to the function a pointer to an array by specifying the array's name without an index. Allow a function to return an array.

```
function disp():string[] {
   return new Array("Mary","Tom","Jack","Jill")
}
var nums:string[] = disp()
for(var i in nums) {
   console.log(nums[i])
}
```

```
var names:string[] = new Array("Mary","Tom","Jack","Jill")
function disp(arr_names:string[]) {
   for(var i = 0;i<arr_names.length;i++) {
      console.log(names[i])
   }
}
disp(names)
```

Typescript class

```
class Person {
    fname: string;
    lname: string;

    constructor(fname: string, lname:string) {
        this.fname = fname;
        this.lname = lname;
    }

    greet() {
        console.log("Hello", this.fname);
    }
}

var p: Person = new Person('Joy', 'Ray');
p.greet();
```

```
Hello Joy
41
```

```
class Model {
    user;
    items;
    contact;
    constructor() {
        this.user = "Shrilata";
        this.contact = {email:"shrilata@gmail.com",
                        phone:9977886600
                       };
        this.items = [new TodoItem("Buy Flowers", false),
                      new TodoItem("Get Shoes", false),
                      new TodoItem("Collect Tickets", false),
                      new TodoItem("Call Joe", false)]
    }
}
class TodoItem {
    action;
    done;
    constructor(action, done) {
        this.action = action;
        this.done = done;
    }
}
```

Inheritance

```
class Person1 {
    fname: string;
    lname: string;

     constructor(fname: string, lname:string) {
         this.fname = fname;
         this.lname = lname;
     }
  }

class Employee extends Person1 {
    empCode: number;

    constructor(empcode: number, fname:string, lname:string) {
        super(fname, lname);
        this.empCode = empcode;
    }

    displayName():void {
        console.log("Name = " + this.fname +  ", Employee Code = " + this.empCode);
    }
}

let emp = new Employee(100, "Bill","Gates");
emp.displayName(); // Name = Bill, Employee Code = 100
```