SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SITA3004 USER INTERFACE DESIGN & IMPLEMENTATION

		L	T	P	Credits	Total Marks
SITA3004	USER INTERFACE DESIGN AND IMPLEMENTATION	3	0	0	3	100

COURSE OBJECTIVES

- To explain hypertext and style sheet languages.
- To apply Responsive Web Design Framework and features.
- To illustrate the basics of JavaScript, jQuerry and allied scripting languages for Web design.
- To build Server side JS frameworks.
- To design client side JS frameworks.

UNIT 1 UI DESIGN 9 Hrs.
HTML5: What is HTML5 - Features of HTML5 – Semantic Tags – New Input Elements and tags - Media tags (audio andvideo tags) – Designing Graphics using Canvas API - Drag and Drop features – Geolocation API - Web storage (Sessionand local storage).CSS3: What is CSS3 – Features of CSS3 – Implementation of border radius, box shadow, image border, custom web font, backgrounds - Advanced text effects(shadow) - 2D and 3D Transformations - Transitions to elements -Animations to text and elements.

UNIT 2 RESPONSIVE WEB DESIGN (RWD) 9 Hrs.
Responsive Design: What is RWD – Introduction to RWD Techniques – Fluid Layout, Fluid Images and Media queries - Introduction to RWD Frame work.Twitter Bootstrap – Bootstrap Background and Features - Getting Started with Bootstrap - Demystifying Grids – Off Canvas - Bootstrap Components - JS Plugins – Customization.

UNIT 3 INTRODUCTION TO JAVASCRIPT AND JQUERY 9 Hrs.
Introduction - Core features - Data types and Variables - Operators, Expressions and Statements - Functions & Scope - Objects - Array, Date and Math related Objects - Document Object Model - Event Handling – Browser Object Model - Windows and Documents - Form handling and validations. Object-Oriented Techniques in JavaScript - Classes – Constructors and Prototyping (Sub classes and Super classes) – JSON – Introduction to AJAX. Introduction – jQuery Selectors – jQuery HTML - Animations – Effects – Event Handling – DOM – jQuery DOM Traversing, DOM Manipulation – jQuery AJAX.

UNIT 4 INTRODUCTION TO SERVER-SIDE JS FRAMEWORK – NODE.JS 9 Hrs.
Introduction - What is Node JS – Architecture – Feature of Node JS - Installation and Setup - Creating web servers with HTTP (Request & Response) – Event Handling - GET & POST implementation - Connect to SQL Database using Node JS – Implementation of CRUD operations.

UNIT 5 INTRODUCTION TO CLIENT-SIDE JS FRAMEWORK 9 Hrs.

Introduction to Angular 4.0 - Needs & Evolution – Features – Setup and Configuration – Components and Modules – Templates – Change Detection – Directives – Data Binding - Pipes – Nested Components. Template - Model Driven Forms or Reactive Forms - Custom Valuators. Introduction to ReactJS - React Components- Build a simple React component- React internals -Component inter communication- Component composition- Component styling.

Max. 45 Hrs.

COURSE OUTCOMES

On Completion of the course, student will be able to
CO1 - Develop web pages and style sheets using HTML and CSS3 respectively.
CO2 - Design responsive websites with RWD techniques.
CO3 - Apply JavaScript and allied scripting languages for implementing object models and functions.
CO4 - Demonstrate Serve Side JS Framework for application development.
CO5 - Use Client Side JS Framework for redefining the application development.
CO6 - Develop User Interface Designs using the frameworks

TEXT / REFERENCE BOOKS

1. Harvey and Paul Deitel & Associates, Harvey Deitel and Abbey Deitel, "Internet and World Wide Web - How to Program", 5th Edition, Pearson Education, 2011.
2. Achyut S Godbole and Atul Kahate, "Web Technologies", 2nd Edition, Tata McGraw Hill, 2012.
3. Thomas A Powell, Fritz Schneider, "JavaScript: The Complete Reference", 3rd Edition, Tata McGraw Hill, 2013.
4. David Flanagan, "JavaScript: The Definitive Guide, 6th Edition", O'Reilly Media, 2011.
5. Bear Bibeault and Yehuda Katz, "jQuery in Action", January 2008.
6. Web link for Responsive Web Design - https://bradfrost.github.io/this-is-responsive/.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

UNIT – I - UI Design – SITA3004

I UI DESIGN

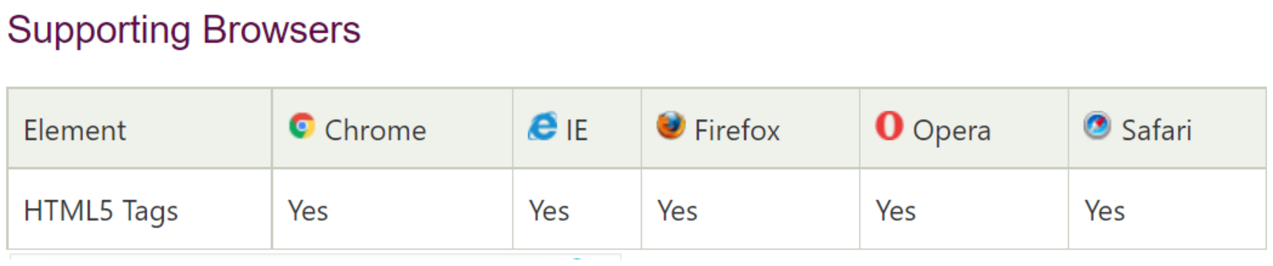
HTML5: What is HTML5 - Features of HTML5 – Semantic Tags – New Input Elements and tags - Media tags (audio and video tags) – Designing Graphics using Canvas API - Drag and Drop features – Geolocation API - Web storage (Session and local storage).CSS3: What is CSS3 – Features of CSS3 – Implementation of border radius, box shadow, image border, custom web font, backgrounds - Advanced text effects(shadow) - 2D and 3D Transformations - Transitions to elements - Animations to text and elements.

HTML5

What is HTML 5?

With HTML you can create your own Web site.HTML stands for **Hyper Text Markup Language**. HTML5 is the latest and most enhanced version of HTML. Technically, HTML5 is not a programming language, but rather a markup language. A markup language is a set of **markup tags** it is used to describe web pages, it is **not case sensitive** language, and documents **contain HTML tags** and plain text. HTML5 is the next major revision of the HTML standard like HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for **structuring and presenting content** on the World Wide Web. HTML5 is a cooperation between the **World Wide Web Consortium** (W3C) and the **Web Hypertext Application Technology Working Group** (WHATWG). The new standard incorporates **features** like **video playback and drag-and-drop** that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

Supporting Browsers



Supporting Browsers

Element	Chrome	IE	Firefox	Opera	Safari
HTML5 Tags	Yes	Yes	Yes	Yes	Yes

Fig 1.1 HTML5 Supporting Browsers

Features

HTML5 introduces a number of new elements and attributes that can help you in building modern websites.

New Semantic Elements − These are like <header>, <footer>, and <section>.
Forms 2.0 − Improvements to HTML web forms where new attributes have been introduced for <input> tag.
Persistent Local Storage − To achieve without resorting to third-party plugins.
WebSocket − A next-generation bidirectional communication technology for web applications.
Server-Sent Events − HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
Canvas − This supports a two-dimensional drawing surface that you can program with JavaScript.
Audio & Video − You can embed audio or video on your web pages without resorting to third-party plug-in.

Geolocation − Now visitors can choose to share their physical location with your web application.
Microdata − This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
Drag and drop − Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers. It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

Syntax

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4.
HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements.
HTML5 comes with a lot of flexibility and it supports the following features
Uppercase tag names.
Quotes are optional for attributes.
Attribute values are optional.
Closing empty elements are optional.

The **DOCTYPE**
DOCTYPEs in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD. HTML 5 authors would use simple syntax to specify DOCTYPE as follows

<!DOCTYPE html>

The above syntax is case-insensitive.

Character Encoding
HTML 5 authors can use simple syntax to specify Character Encoding as follows –

<meta charset = “UTF-8”>

The above syntax is case-insensitive.

The **<script>** tag
It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows –

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

HTML 5 removes extra information required and you can use simply following syntax –

```
<script src = "scriptfile.js"></script>
```

The **<link>** tag
So far you were writing <link> as follows –

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

HTML 5 removes extra information required and you can simply use the following syntax −

```
<link rel = "stylesheet" href = "stylefile.css">
```

HTML 5 Tags

There is a list of newly included tags in HTML 5. These HTML 5 tags (elements) provide a better document structure. This list shows all HTML 5 tags in alphabetical order with description.

Tag	Description
<article>	This element is used to define an independent piece of content in a document, that may be a blog, a magazine or a newspaper article.
<aside>	It specifies that article is slightly related to the rest of the whole page.
<audio>	It is used to play audio file in HTML.
<bdi>	The bdi stands for bi-directional isolation. It isolates a part of text that is formatted in other direction from the outside text document.
<canvas>	It is used to draw canvas.
<data>	It provides machine readable version of its data.
<datalist>	It provides auto complete feature for textfield.
<details>	It specifies the additional information or controls required by user.
<dialog>	It defines a window or a dialog box.
<figcaption>	It is used to define a caption for a <figure> element.
<figure>	It defines a self-contained content like photos, diagrams etc.
<footer>	It defines a footer for a section.
<header>	It defines a header for a section.
<main>	It defines the main content of a document.

<mark>	It specifies the marked or highlighted content.
<menuitem>	It defines a command that the user can invoke from a popup menu.
<meter>	It is used to measure the scalar value within a given range.
<nav>	It is used to define the navigation link in the document.
<progress>	It specifies the progress of the task.
<rp>	It defines what to show in browser that don't support ruby annotation.
<rt>	It defines an explanation/pronunciation of characters.
<ruby>	It defines ruby annotation along with <rp> and <rt>.
<section>	It defines a section in the document.
<summary>	It specifies a visible heading for <detailed> element.
<svg>	It is used to display shapes.
<time>	It is used to define a date/time.
<video>	It is used to play video file in HTML.
<wbr>	It defines a possible line break.

Fig 1.2. List of HTML5 Tags

HTML Semantic Elements

What are Semantic Elements?
A semantic element clearly describes its meaning to both the browser and the developer.
Examples of non-semantic elements: <div> and <span> - Tells nothing about its content.
Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

HTML <section> Element

The <section> element defines a section in a document.
According to W3C's HTML documentation: "A section is a thematic grouping of content, typically with a heading."

Examples of where a <section> element can be used:

- Chapters
- Introduction

- News items
- Contact information

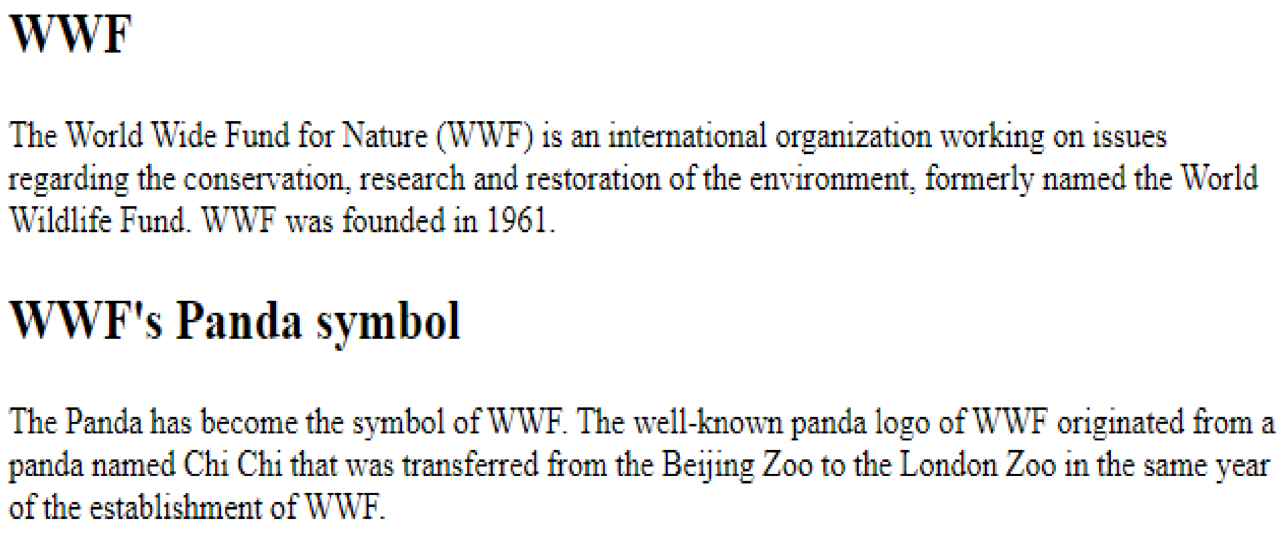
A web page could normally be split into sections for introduction, content, and contact information.

Example

Two sections in a document:

```
<section>
<h1>WWF</h1>
<p>The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.</p>
</section>
<section>
<h1>WWF's Panda symbol</h1>
<p>The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.</p>
</section>
```

Output



WWF

The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.

WWF's Panda symbol

The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.

Fig 1.3 Section output

HTML <article> Element

The <article> element specifies independent, self-contained content.
An article should make sense on its own, and it should be possible to distribute it independently from the rest of the web site.
Examples of where the <article> element can be used:

- Forum posts
- Blog posts
- User comments
- Product cards

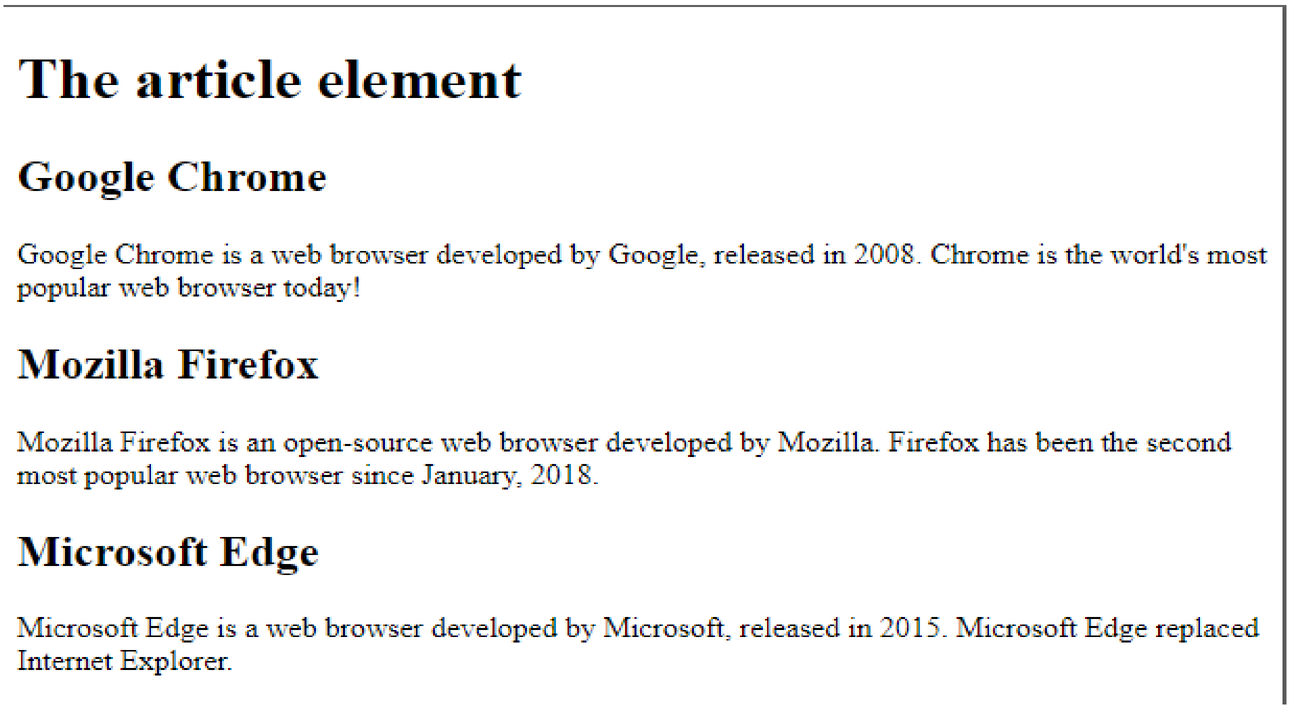
- Newspaper articles

Example

Three articles with independent, self-contained content:

```
<article>
<h2>Google Chrome</h2>
<p>Google Chrome is a web browser developed by Google, released in 2008. Chrome is the world's most popular web browser today!</p>
</article>
<article>
<h2>Mozilla Firefox</h2>
<p>Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the second most popular web browser since January, 2018.</p>
</article>
<article>
<h2>Microsoft Edge</h2>
<p>Microsoft Edge is a web browser developed by Microsoft, released in 2015. Microsoft Edge replaced Internet Explorer.</p>
</article>
```

Output



The article element

Google Chrome

Google Chrome is a web browser developed by Google, released in 2008. Chrome is the world's most popular web browser today!

Mozilla Firefox

Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the second most popular web browser since January, 2018.

Microsoft Edge

Microsoft Edge is a web browser developed by Microsoft, released in 2015. Microsoft Edge replaced Internet Explorer.

Fig 1.4 Article output

Nesting <article> in <section> or Vice Versa?

The <article> element specifies independent, self-contained content. The <section> element defines section in a document. Can we use the definitions to decide how to nest those elements? No, we cannot! So, you will find HTML pages with <section> elements containing <article> elements, and <article> elements containing <section> elements.

HTML <header> Element

The <header> element represents a container for introductory content or a set of navigational links.

A <header> element typically contains:

one or more heading elements (<h1> - <h6>)

logo or icon

authorship information

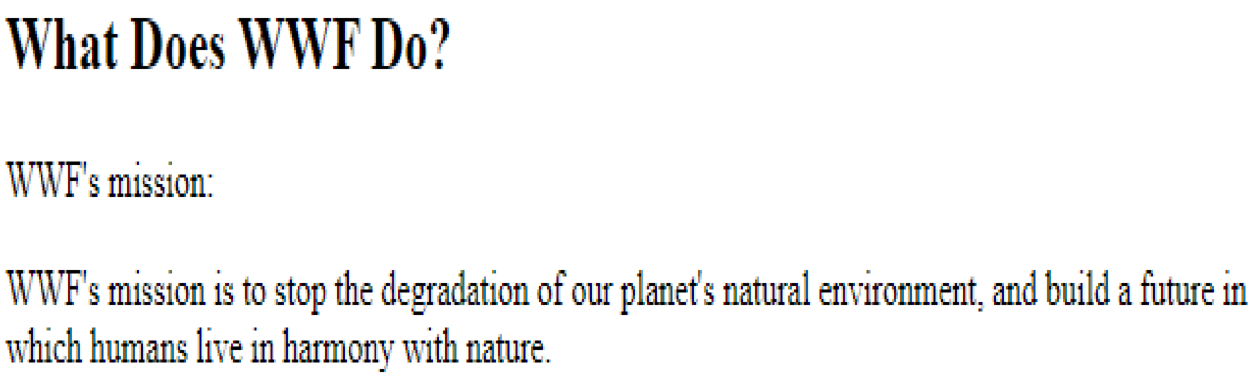
Note: You can have several <header> elements in one HTML document. However, <header> cannot be placed within a <footer>, <address> or another <header> element.

Example

A header for an <article>:

```
<article>
  <header>
    <h1>What Does WWF Do?</h1>
    <p>WWF's mission:</p>
  </header>
  <p>WWF's mission is to stop the degradation of our planet's natural environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

Output



What Does WWF Do?

WWF's mission:

WWF's mission is to stop the degradation of our planet's natural environment, and build a future in which humans live in harmony with nature.

Fig 1.5 Header for an Article output

HTML <footer> Element

The <footer> element defines a footer for a document or section.

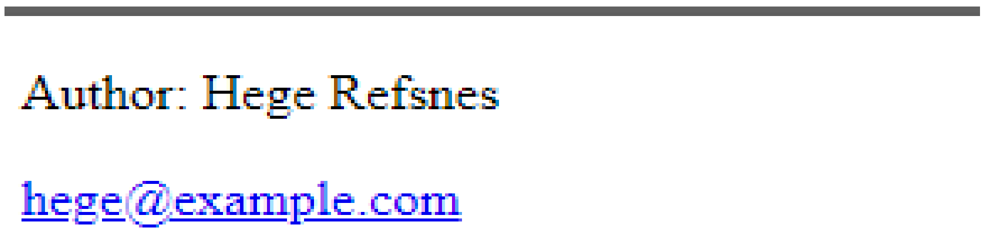
A <footer> element typically contains:

- authorship information
- copyright information
- contact information
- sitemap
- back to top links

- related documents

You can have several <footer> elements in one document.

Output


Author: Hege Refsnes

hege@example.com

Fig 1.6 Footer Output

HTML <nav> Element

- The <nav> element defines a set of navigation links.
- Notice that NOT all links of a document should be inside a <nav> element.
- The <nav> element is intended only for major block of navigation links.

Browsers, such as screen readers for disabled users, can use this element to determine whether to omit the initial rendering of this content.

Output

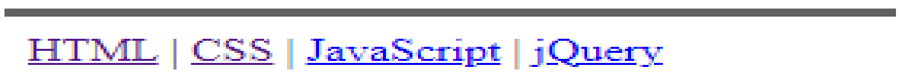

HTML | CSS | JavaScript | jQuery

Fig 1.7. Navigation Output

HTML <aside> Element

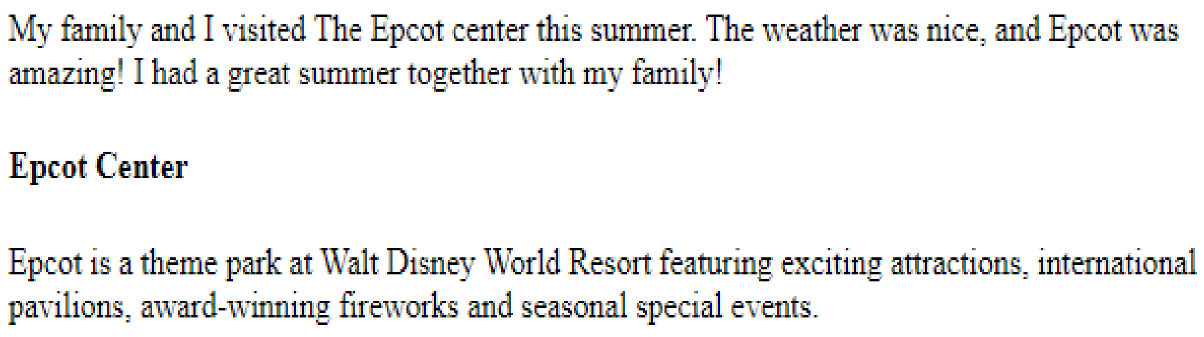
The <aside> element defines some content aside from the content it is placed in (like a sidebar). The <aside> content should be indirectly related to the surrounding content.

Example

Display some content aside from the content it is placed in:

<p>My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!</p>
<aside>
<h4>Epcot Center</h4>
<p>Epcot is a theme park at Walt Disney World Resort featuring exciting attractions, international pavilions, award-winning fireworks and seasonal special events.</p>
</aside>

Output



My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!

Epcot Center

Epcot is a theme park at Walt Disney World Resort featuring exciting attractions, international pavilions, award-winning fireworks and seasonal special events.

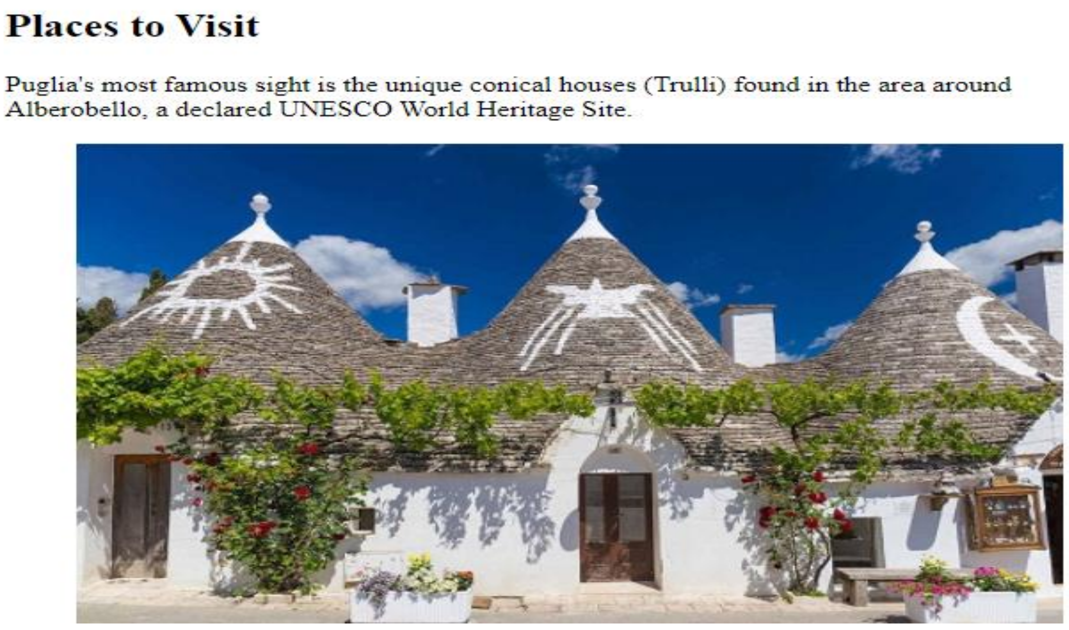
Fig 1.8 Aside output

HTML <figure> and <figcaption> Elements

The <figure> tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
The <figcaption> tag defines a caption for a <figure> element. The <figcaption> element can be placed as the first or as the last child of a <figure> element.
The <img> element defines the actual image/illustration.

Example
```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

Output



Places to Visit

Puglia's most famous sight is the unique conical houses (Trulli) found in the area around Alberobello, a declared UNESCO World Heritage Site.

Fig 1.9 Figure Output

HTML Input Types

Here are the different input types you can use in HTML:

```
<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">
```

The default value of the type attribute is "text".

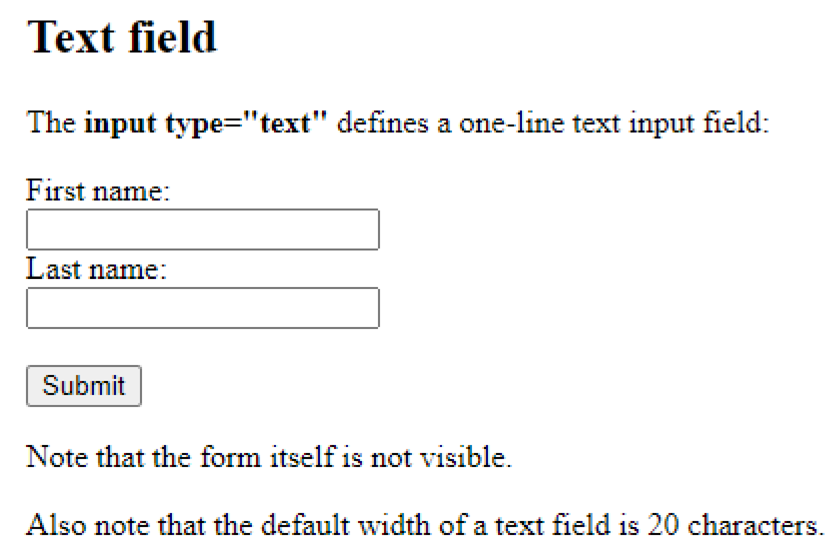
Input Type Text

<input type="text"> defines a single-line text input field:

Example

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

Output


Text field

The **input type="text"** defines a one-line text input field:

First name:

Last name:

Submit

Note that the form itself is not visible.

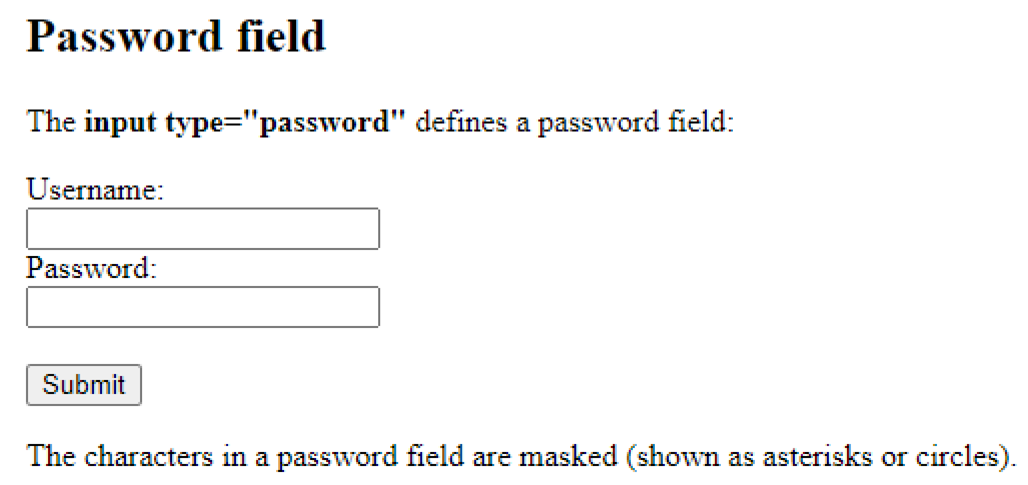
Also note that the default width of a text field is 20 characters.

Fig 1.9 Text Field

Input Type Password

```
<input type="password"> defines a password field:
Example
<form>
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd">
</form>
```

Output


Password field

The **input type="password"** defines a password field:

Username:

Password:

Submit

The characters in a password field are masked (shown as asterisks or circles).

Fig 1.10 Password output

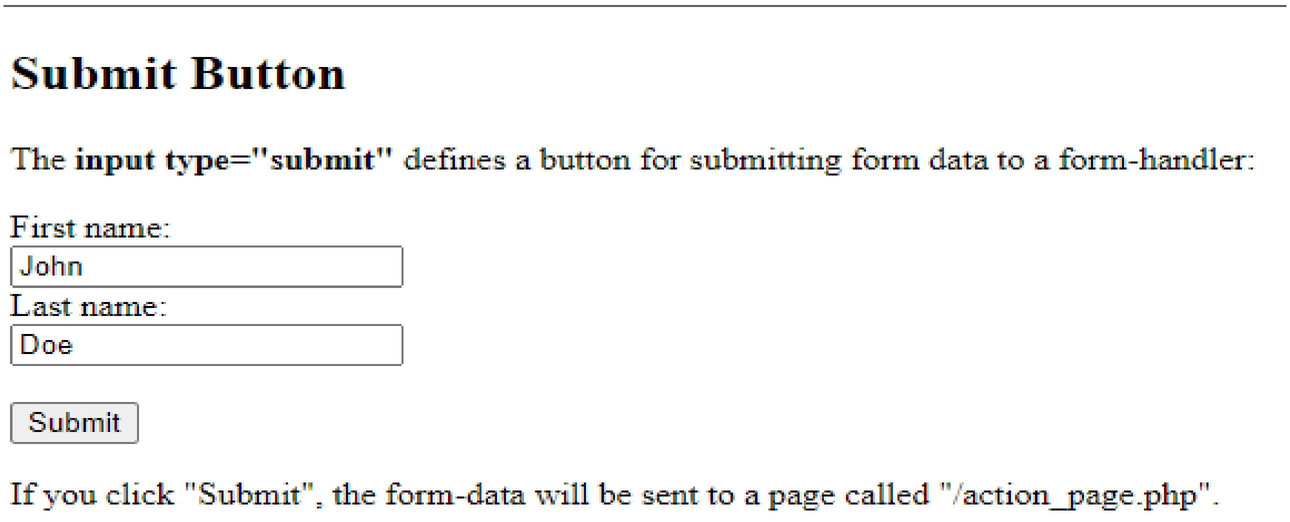
Input Type Submit

<input type="submit"> defines a button for submitting form data to a form-handler.
The form-handler is typically a server page with a script for processing input data.
The form-handler is specified in the form's action attribute:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Output



Submit Button

The **input type="submit"** defines a button for submitting form data to a form-handler:

First name:
John
Last name:
Doe

If you click "Submit", the form-data will be sent to a page called "/action_page.php".

Fig 1.11 Submit output

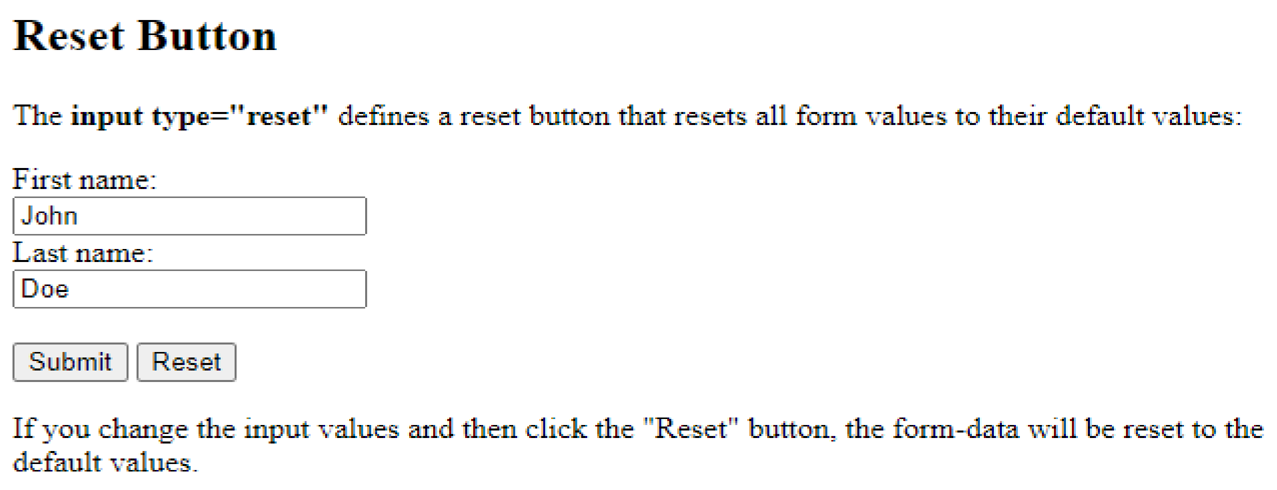
Input Type Reset

<input type="reset"> defines a reset button that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

Output



Reset Button

The **input type="reset"** defines a reset button that resets all form values to their default values:

First name:
John
Last name:
Doe

If you change the input values and then click the "Reset" button, the form-data will be reset to the default values.

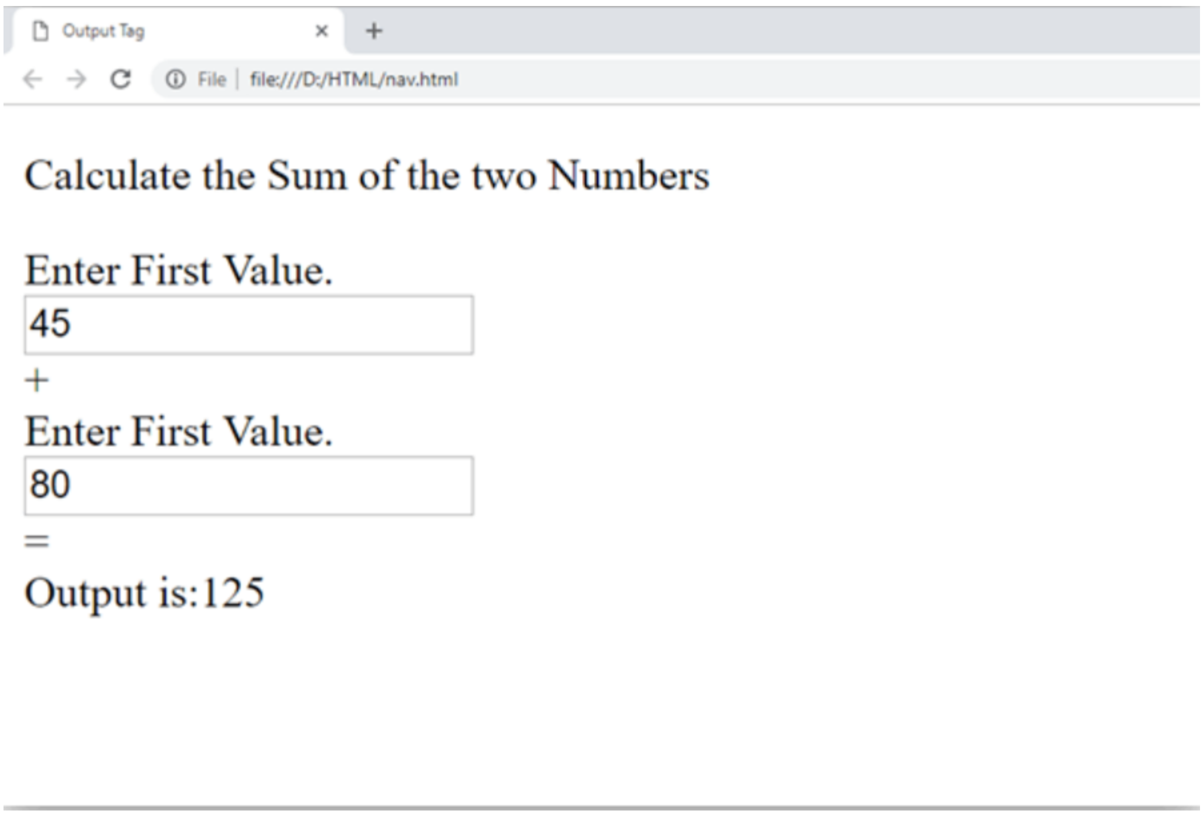
Fig 1.12 Reset output

<output> element in HTML5

HTML5 introduced a new element <output> which is used to represent the result of different types of output, such as output written by a script. You can use the **for** attribute to specify a relationship between the output element and other elements in the document that affected the calculation (for example, as inputs or parameters). The value of the for attribute is a space-separated list of IDs of other elements.

```
<!DOCTYPE html>
<html>
<head>
<title>Output Tag</title>
</head>
<body>
 <p>Calculate the Sum of the two Numbers</p>
 <form oninput="res.value=parseInt(a.value)+parseInt(b.value);">
    <label>Enter First Value.</label><br>
    <input type="number" name="a" value=""/><br>
    +<br/>
    <label>Enter First Value.</label><br>
    <input type="number" name="b" value=""><br>
    =<br>
    Output is:<output name="res"></output>
 </form>
</body>
</html>
```

Output



Calculate the Sum of the two Numbers

Enter First Value.
45
+
Enter First Value.
80
=
Output is:125

Fig 1.13 Output tag

The placeholder attribute

- HTML5 introduced a new attribute called placeholder.
- This attribute on <input> and <textarea> elements provide a hint to the user of what can be entered in the field.
- The placeholder text must not contain carriage returns or line-feeds.

Here is the simple syntax for placeholder attribute −
<input type = "text" name = "search" placeholder = "search the web"/>
This attribute is supported by latest versions of Mozilla, Safari and Chrome browsers only.

```
<!DOCTYPE HTML>
<html>
  <body>
    <form action = "" method = "get">
      Enter email : <input type = "email" name = "newinput"
        placeholder = "email@example.com"/>
      <input type = "submit" value = "submit" />
    </form>
  </body>
</html>
```

Output

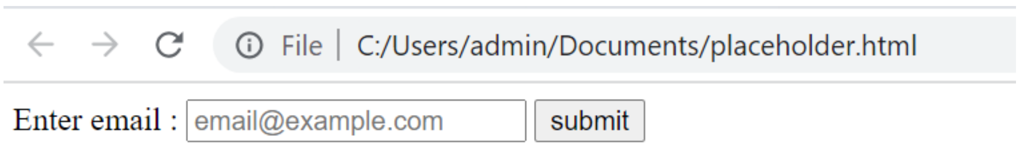


Fig 1.14 Placeholder output

The autofocus attribute

This is a simple one-step pattern, easily programmed in JavaScript at the time of document load, automatically focus one particular form field. HTML5 introduced a new attribute called autofocus which would be used as follows –

```
<input type = "text" name = "search" autofocus/>
```

This attribute is supported by latest versions of Mozilla, Safari and Chrome browsers only.

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

    <form action = "" method = "get">
      Enter email : <input type = "text" name = "newinput" autofocus/>
      <p>Try to submit using Submit button</p>
      <input type = "submit" value = "submit" />
    </form>

  </body>
</html>
```

Output

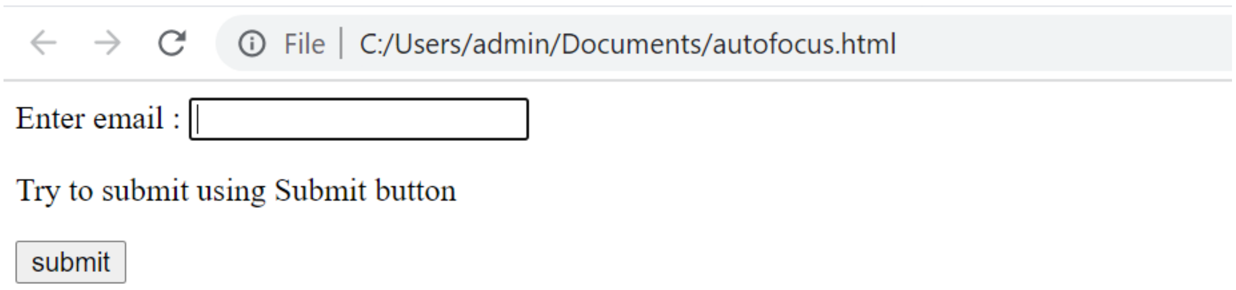

File
C:/Users/admin/Documents/autofocus.html
Enter email :
Try to submit using Submit button
submit

Fig 1.15 Autofocus output

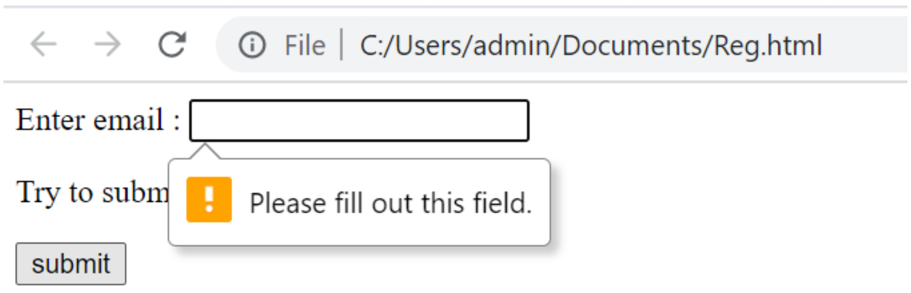
The required attribute

Now you do not need to have JavaScript for client-side validations like empty text box would never be submitted because HTML5 introduced a new attribute called required which would be used as follows and would insist to have a value –

```
<input type = "text" name = "search" required/>

<!DOCTYPE HTML>
```

```
<html>
  <body>

    <form action = "" method = "get">
      Enter email : <input type = "text" name = "newinput" required/>
      <p>Try to submit using Submit button</p>
      <input type = "submit" value = "submit" />
    </form>

  </body>
</html>
```

Output



C:/Users/admin/Documents/Reg.html

Enter email :

Try to subm

Please fill out this field.

Fig 1.16 Required output

Audio and Video Tags

HTML5 features include native audio and video support without the need for Flash.
The HTML5 <audio> and <video> tags make it simple to add media to a website.
You need to set src attribute to identify the media source and include a controls attribute so the user can play and pause the media.

Video Tags

Embedding Video
Here is the simplest form of embedding a video file in your webpage –

```
<video src = "foo.mp4"  width = "300" height = "200" controls>
 Your browser does not support the <video> element.
</video>
```

- The current HTML5 draft specification does not specify which video formats browsers should support in the video tag.
- But most commonly used video formats are −
- ➢ **Ogg** − Ogg files with Thedora video codec and Vorbis audio codec.
- ➢ **mpeg4** − MPEG4 files with H.264 video codec and AAC audio codec.
- You can use <source> tag to specify media along with media type and many other attributes.
- A video element allows multiple source elements and browser will use the first recognized format.

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

    <video  width = "300" height = "200" controls autoplay>
      <source src = "/html5/foo.ogg" type ="video/ogg" />
      <source src = "/html5/foo.mp4" type = "video/mp4" />
      Your browser does not support the <video> element.
    </video>

  </body>
</html>
```

Output

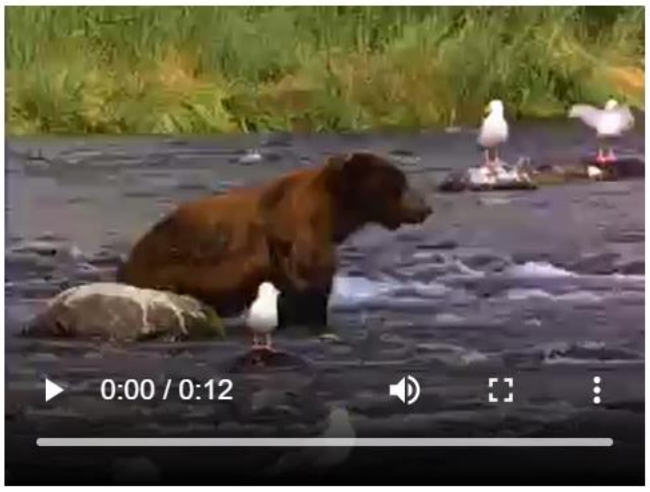


Fig 1.17 Video tag

Embedding Audio

- HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

```
<audio src = "foo.wav" controls autoplay>
  Your browser does not support the <audio> element.
</audio>
```

- The current HTML5 draft specification does not specify which audio formats browsers should support in the audio tag.
- But most commonly used audio formats are **ogg, mp3** and **wav**.
- You can use <source> tag to specify media along with media type and many other attributes.
- An audio element allows multiple source elements and browser will use the first recognized format −

```
<!DOCTYPE HTML>
<html>
  <body>
       <audio controls autoplay>
      <source src = "/html5/audio.ogg" type = "audio/ogg" />
      <source src = "/html5/audio.wav" type = "audio/wav" />
      Your browser does not support the <audio> element.
    </audio>

  </body>
</html>
```

Output

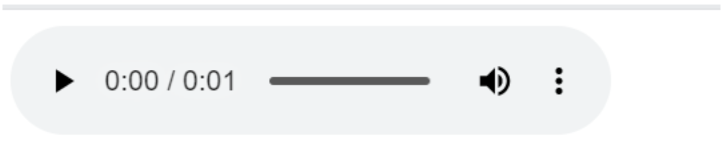


Fig 1.18 Audio output

Configuring Servers for Media Type
Most of the servers don't have Ogg or mp4 media with the correct MIME types, so you'll likely need to add the appropriate configuration for this.

- AddType audio/ogg .oga
- AddType audio/wav .wav
- AddType video/ogg .ogv or . ogg
- AddType video/mp4 .mp4

```
<!DOCTYPE HTML>
<html>
  <head>
     <script type = "text/javascript">
      function PlayVideo() {
        var v = document.getElementsByTagName("video")[0];
        v.play();
      }
    </script>
  </head>
  <body>
    <form>
      <video width = "300" height = "200" src = "/html5/foo.mp4">
      Your browser does not support the video element.
      </video>
      <br />
      <input type = "button" onclick = "PlayVideo();" value = "Play"/>
    </form>
  </body>
</html>
```

Output

Fig 1.19 Media output

Canvas tag

HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations. Here is a simple <canvas> element which has only two specific attributes width and height plus all the core HTML5 attributes like id, name and class, etc.

<canvas id = "mycanvas" width = "100" height = "100"></canvas>

You can easily find that <canvas> element in the DOM using *getElementById()* method as follows −

```
var canvas = document.getElementById("mycanvas");
```

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

    <style>
      #mycanvas{border:1px solid red;}
    </style>
  </head>
    <body>
    <canvas id = "mycanvas" width = "100" height = "100"></canvas>
  </body>
</html>
```

Output

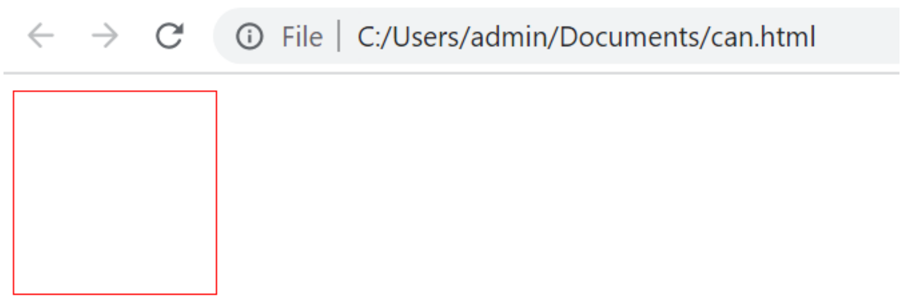


Fig 1.20 Canvas Output

The Rendering Context

The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it. The canvas element has a DOM method called **getContext**, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context **2d**.

Following is the code to get required context along with a check if your browser supports <canvas> element −

```
var canvas = document.getElementById("mycanvas");
if (canvas.getContext)
{
var ctx = canvas.getContext('2d'); // drawing code here
}
else
{ // canvas-unsupported code here
 }
```

Browser Support

The latest versions of Firefox, Safari, Chrome and Opera all support for HTML5 Canvas but IE8 does not support canvas natively. You can use ExplorerCanvas to have canvas support through Internet Explorer. You just need to include this JavaScript as follows –

```
<!--[if IE]><script src = "excanvas.js"></script><![endif]-->
```

Make an Element Draggable

- To make an element draggable, set the draggable attribute to true:

```
<img draggable="true">
```

- What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged. In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged. The dataTransfer.setData() method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);}
```

In this case, the data type is "text" and the value is the id of the draggable element ("drag1").

Where to Drop – ondragover. The ondragover event specifies where the dragged data can be dropped. By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element. This is done by calling the event.preventDefault() method for the ondragover event:

```
event.preventDefault()
```

Do the Drop – ondrop
When the dragged data is dropped, a drop event occurs. In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

Code explaination:
Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop). Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method. The dragged data is the id of the dragged element ("drag1").Append the dragged element into the drop element

What is HTML Web Storage?

With web storage, web applications can store data locally within the user's browser. Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server. Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data. HTML web storage provides two objects for storing data on the client:

- window.localStorage - stores data with no expiration date
- window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)

Before using web storage, check browser support for localStorage and sessionStorage:

```
if (typeof(Storage) !== "undefined")
{
  // Code for localStorage/sessionStorage.
}
else
{
  // Sorry! No Web Storage support..
}
```

The localStorage Object
The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML= localStorage.getItem("lastname");
```

Create a localStorage name/value pair with name="lastname" and value="Smith"
Retrieve the value of "lastname" and insert it into the element with id="result"
The example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML = localStorage.lastname;
```

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

- **Note:** Name/value pairs are always stored as strings.
- Remember to convert them to another format when needed!

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:
Example

```
if (localStorage.clickcount){
 localStorage.clickcount =Number(localStorage.clickcount)+ 1;
} else {
 localStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the button " +
localStorage.clickcount + " time(s).";
```

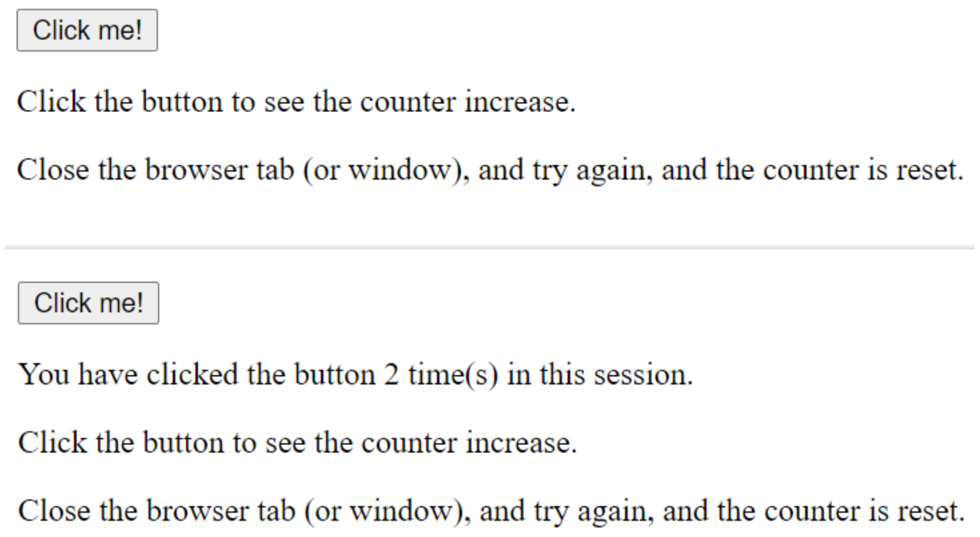
The sessionStorage Object

- The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session.
- The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {
  sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
} else {
  sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
```

Output


Click me!

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.

Click me!

You have clicked the button 2 time(s) in this session.

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.

Fig 1.21 Storage output

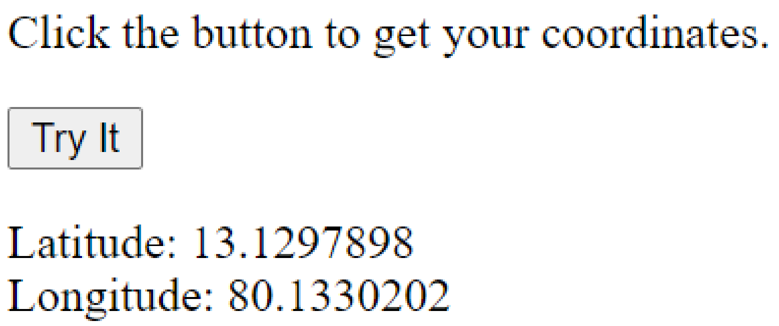
HTML5 Geolocation
The HTML Geolocation API is used to get the geographical position of a user. Since this can compromise privacy, the position is not available unless the user approves it. The getCurrentPosition() method is used to return the user's position. The example below returns the latitude and longitude of the user's position:

Location-specific Information
This page has demonstrated how to show a user's position on a map.
Geolocation is also very useful for location-specific information, like:
- Up-to-date local information
- Showing Points-of-interest near the user

Turn-by-turn navigation (GPS)

```
<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>
```

Output


Click the button to get your coordinates.

```
Latitude: 13.1297898
Longitude: 80.1330202
```

Fig 1.22 Geolocation output

Check if Geolocation is supported. If supported, run the getCurrentPosition() method. If not, display a message to the user.If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition). The showPosition() function outputs the Latitude and Longitude. The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

- The second parameter of the getCurrentPosition() method is used to handle errors.
- It specifies a function to run if it fails to get the user's location

```
function showError(error){
  switch(error.code){
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
      x.innerHTML = "An unknown error occurred."
      break;
  }
}
```

Location-specific Information

This page has demonstrated how to show a user's position on a map. Geolocation is also very useful for location-specific information, like:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

- The getCurrentPosition() method returns an object on success.
- The latitude, longitude and accuracy properties are always returned.

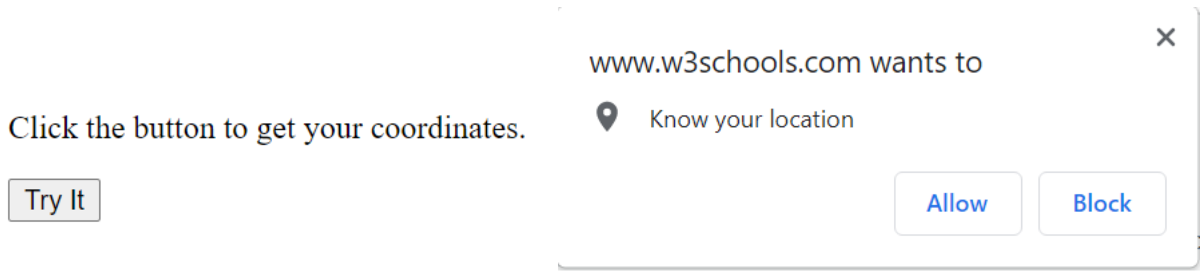
Geolocation Object - Other interesting Methods
The Geolocation object also has other interesting methods:

- watchPosition() - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- clearWatch() - Stops the watchPosition() method.

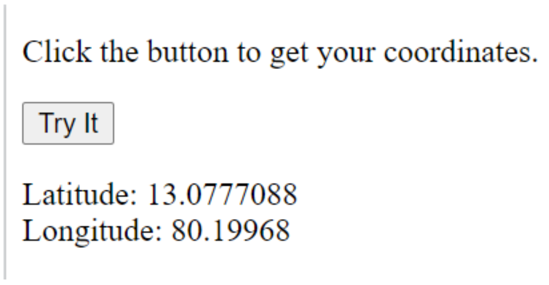
The example below shows the watchPosition() method.
You need an accurate GPS device to test this (like smartphone):

```
<script>
var x=document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.watchPosition(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>
```

Output


Click the button to get your coordinates.

www.w3schools.com wants to
Know your location


Click the button to get your coordinates.

```
Latitude: 13.0777088
Longitude: 80.19968
```

Fig 1.23 Geolocation output

What is CSS 3?

CSS stands for **Cascading Style Sheets**. It is a style sheet language which is used to **describe the look and formatting** of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL. CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

Why use CSS?
These are the three major benefits of CSS:
1) Solves a big problem
- Before CSS, tags like font, color, background style, element alignments, border and size had to be repeated on every web page.
- For example: If you are developing a large website where fonts and color information are added on every single page,
- it will be become a long and expensive process.

2) Saves a lot of time
CSS style definitions are saved in external CSS files so it is possible to change the entire website by changing just one file.
3) Provide more attributes
CSS provides more detailed attributes than plain HTML to define the look and feel of the website.

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

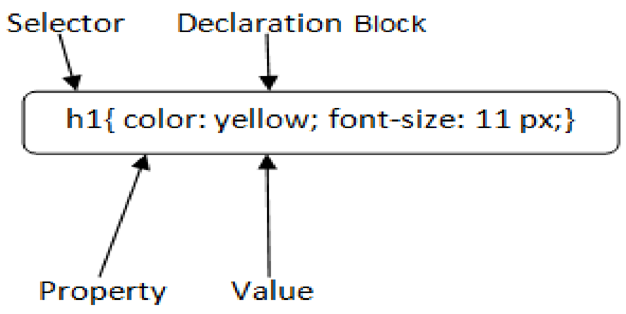


Fig 1.24 CSS Structure

Selector: Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.
Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

color: yellow;
font-size: 11 px;

Each declaration contains a property name and value, separated by a colon.
Property: A Property is a type of attribute of HTML element. It could be color, border etc.
Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.
CSS Selector
- **CSS selectors** are used ***to select the content you want to style***.

- Selectors are the part of CSS rule set.
- CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

- CSS Element Selector
- CSS Id Selector
- CSS Class Selector
- CSS Universal Selector
- CSS Group Selector

CSS Element Selector

The element selector selects the HTML element by name.

```
<!DOCTYPE html>

<html>

<head>

<style>

p{

   text-align: center;

   color: blue;

}

</style>

</head>

<body>

<p>This style will be applied on every paragraph.</p>

<p id="para1">Me too!</p>

<p>And me!</p>

</body>

</html>
```

The CSS id Selector

- The id selector uses the id attribute of an HTML element to select a specific element.
- The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
<!DOCTYPE html>

<html>

<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>
</html>
```

The CSS class Selector

- The class selector selects HTML elements with a specific class attribute.
- To select elements with a specific class, write a period (.) character, followed by the class name.

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  text-align: center;
  color: red;
}
</style>
</head>
<body>
<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>
</body>
</html>
```

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

```
<!DOCTYPE html>
```

```
<html>
<head>
<style>
* {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>
<h1>Hello world!</h1>
<p>Every element on the page will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p></body>
</html>
```

The CSS Grouping Selector

- The grouping selector selects all the HTML elements with the same style definitions.
- It will be better to group the selectors, to minimize the code.
- To group selectors, separate each selector with a comma.

```
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>
```

```
<h1>Hello World!</h1>

<h2>Smaller heading!</h2>

<p>This is a paragraph.</p>

</body>

</html>
```

Types of style sheets

- There are three types of style sheets. They are
  - ❖ **Inline CSS**
  - ❖ **Internal CSS**
  - ❖ **External CSS**

Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

```
<!DOCTYPE html>

<html>

<body>

<h1 style="color:blue;text-align:center;">

This is a heading</h1>

<p style="color:red;">This is a paragraph.</p>

</body>

</html>
```

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style. The internal style is defined inside the <style> element, inside the head section.

```
<!DOCTYPE html>

<html>

<head>

<style>

body {

  background-color: linen;

}

h1 {
  color: maroon;

  margin-left: 40px;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>
```

External CSS

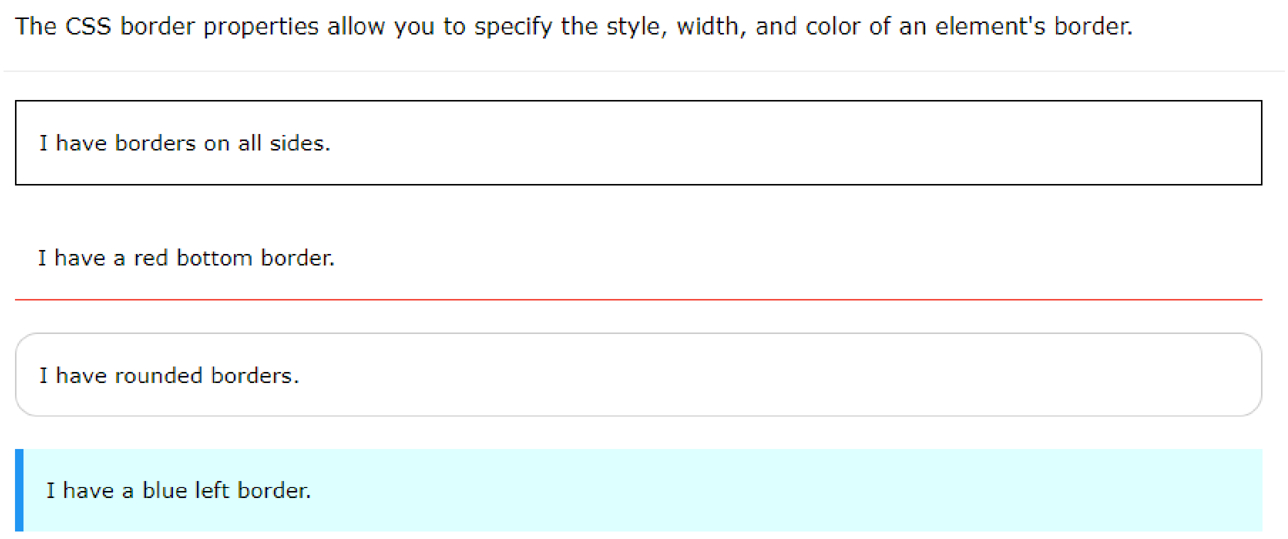
With an external style sheet, you can change the look of an entire website by changing just one file. Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

```
<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="mystyle.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>
```

mystyle.css

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;

  margin-left: 20px;
}
```

CSS Borders


The CSS border properties allow you to specify the style, width, and color of an element's border.

I have borders on all sides.

I have a red bottom border.

I have rounded borders.

I have a blue left border.

Fig 1.25 CSS Border Style

CSS Border Style

The border-style property specifies what kind of border to display.

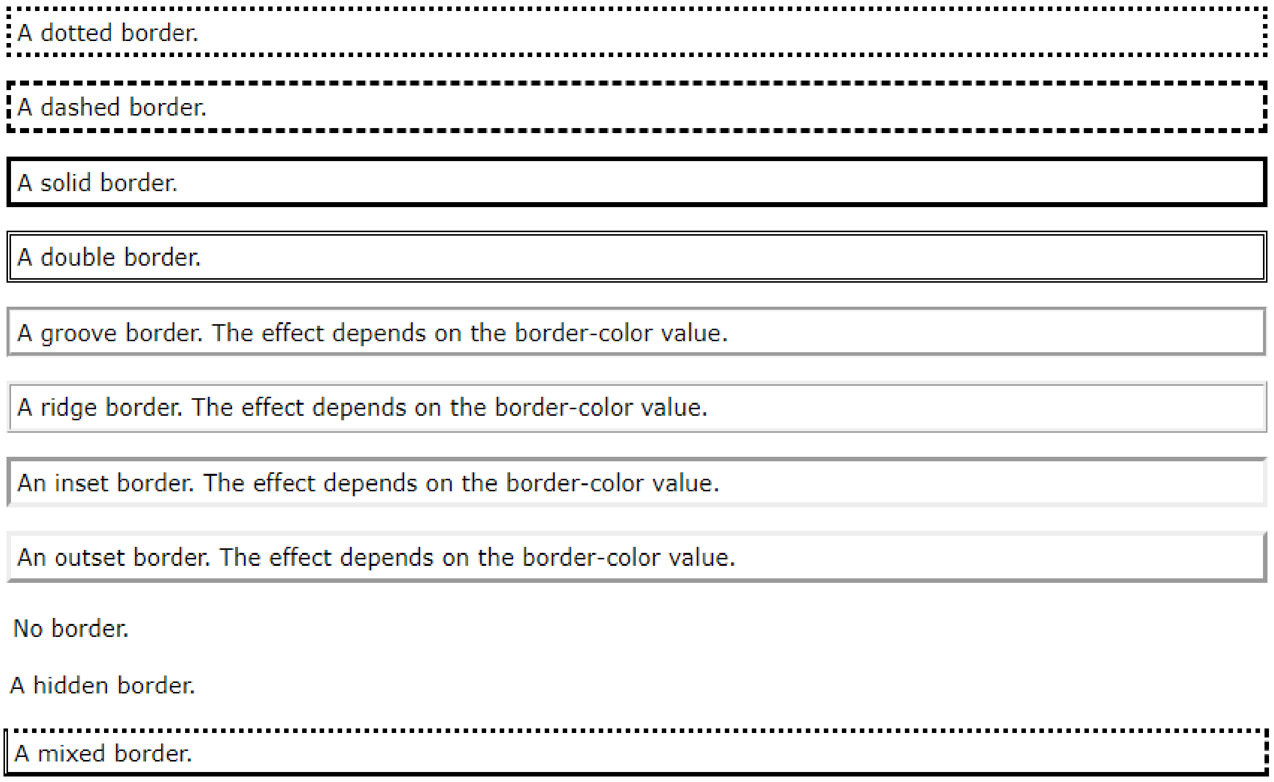
The following values are allowed:

- dotted - Defines a dotted border
- dashed - Defines a dashed border
- solid - Defines a solid border
- double - Defines a double border
- groove - Defines a 3D grooved border. The effect depends on the border-color value
- ridge - Defines a 3D ridged border. The effect depends on the border-color value
- inset - Defines a 3D inset border. The effect depends on the border-color value
- outset - Defines a 3D outset border. The effect depends on the border-color value
- none - Defines no border
- hidden - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

```
<!DOCTYPE html>
<html>
<head>
<style>
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
```

```
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
</style>
</head>
<body>
<h2>The border-style Property</h2>
<p>This property specifies what kind of border to display:</p>
<p class="dotted">A dotted border.</p>
<p class="dashed">A dashed border.</p>
<p class="solid">A solid border.</p>
<p class="double">A double border.</p>
<p class="groove">A groove border.</p>
<p class="ridge">A ridge border.</p>
<p class="inset">An inset border.</p>
<p class="outset">An outset border.</p>
<p class="none">No border.</p>
<p class="hidden">A hidden border.</p>
<p class="mix">A mixed border.</p>
</body>
</html>
```



A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

Fig 1.26 CSS Border Style

CSS Border Width

The border-width property specifies the width of the four borders. The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick:

Example

Demonstration of the different border widths:

```
p.one {
  border-style: solid;
  border-width: 5px;
}
p.two {
  border-style: solid;
  border-width: medium;
}
p.three {
  border-style: dotted;
  border-width: 2px;
}
p.four {
  border-style: dotted;
  border-width: thick;
}
```

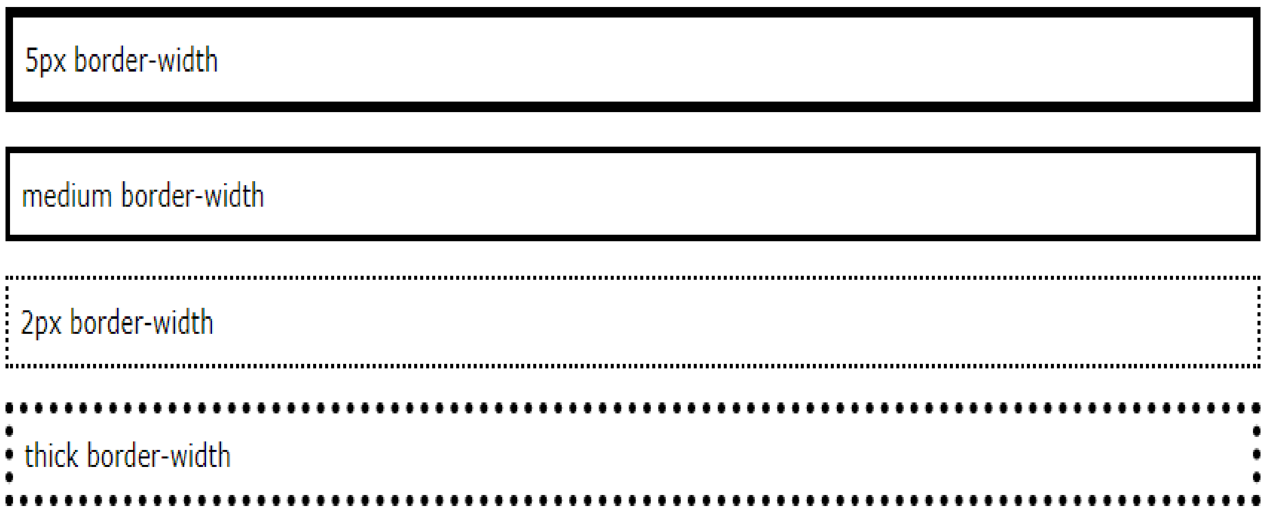
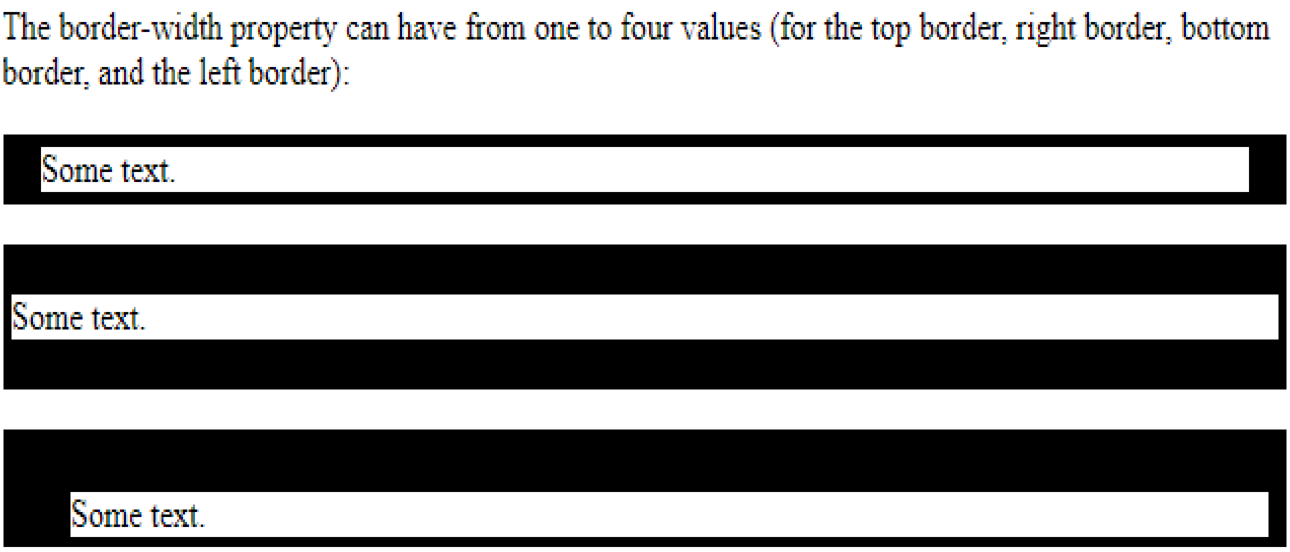


Fig 1.27 CSS Border width

Specific Side Widths

The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border):

```
p.one {
  border-style: solid;
  border-width: 5px 20px; /* 5px top and bottom, 20px on the sides */
}
p.two {
  border-style: solid;
  border-width: 20px 5px; /* 20px top and bottom, 5px on the sides */
}
p.three {
  border-style: solid;
  border-width: 25px 10px 4px 35px; /* 25px top, 10px right, 4px bottom and 35px left */
}
```


The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border):

Some text.

Some text.

Some text.

Fig 1.28 CSS Border width property

CSS Border Color

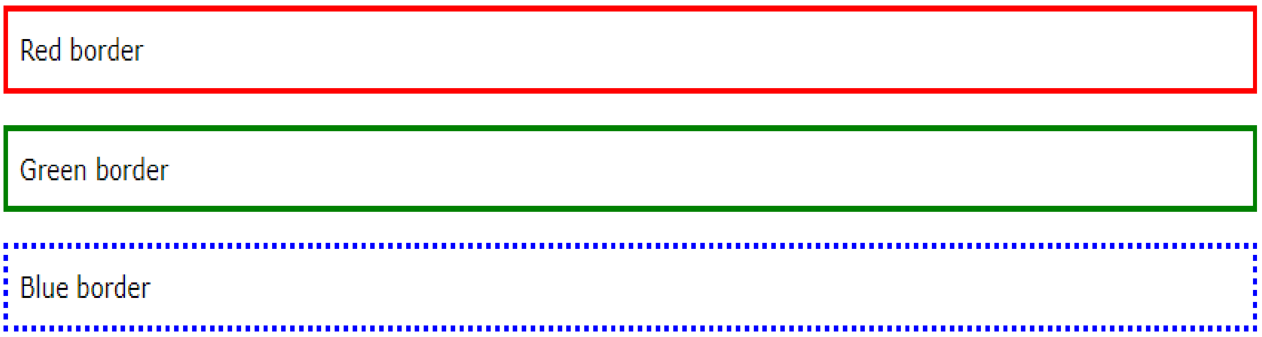
The border-color property is used to set the color of the four borders.

The color can be set by:

- name - specify a color name, like "red“
- HEX - specify a HEX value, like "#ff0000“
- RGB - specify a RGB value, like "rgb(255,0,0)“
- HSL - specify a HSL value, like "hsl(0, 100%, 50%)“ (hue, saturation and lightness) transparent

Note: If border-color is not set, it inherits the color of the element.

```
p.one {
  border-style: solid;
  border-color: red;
}

p.two {
  border-style: solid;
  border-color: green;
}

p.three {
  border-style: dotted;
  border-color: blue;
}
```



Red border

Green border

Blue border

Fig 1.29 CSS Border Color

Specific Side Colors

The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border).

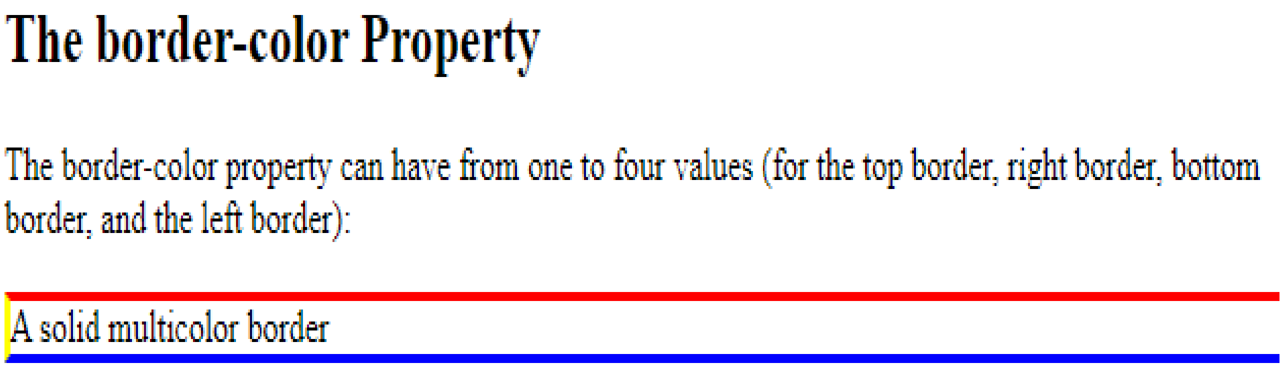
p.one {

  border-style: solid;

  border-color: red green blue yellow; /* red top, green right, blue bottom and yellow left */

}

Output



The border-color Property

The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border):

A solid multicolor border

Fig 1.30 CSS Border Color Property

HEX Values

The color of the border can also be specified using a hexadecimal value (HEX):

Example

p.one {

  border-style: solid;

  border-color: #ff0000; /* red */

}

RGB Values

Example

```
p.one {
  border-style: solid;
  border-color: rgb(255, 0, 0); /* red */
}
```

HSL Values

You can also use HSL values:

Example

```
p.one {
  border-style: solid;
  border-color: hsl(0, 100%, 50%); /* red */
}
```

CSS Border - Individual Sides

From the examples on the previous pages, you have seen that it is possible to specify a different border for each side.

In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):

Example

```
p {
  border-top-style: dotted;
  border-right-style: solid;
  border-bottom-style: dotted;
  border-left-style: solid;
```

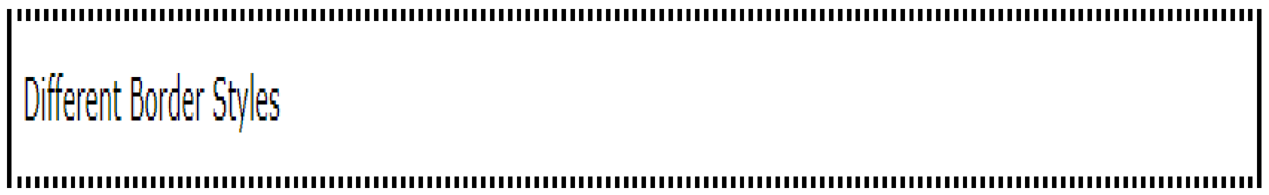


Fig 1.31 Individual Sides

If the border-style property has four values:

border-style: dotted solid double dashed

top border is dotted

right border is solid

bottom border is double

left border is dashed

If the border-style property has three values:

border-style: dotted solid double

top border is dotted

right and left borders are solid

bottom border is double

If the border-style property has two values:

border-style: dotted solid

top and bottom borders are dotted

right and left borders are solid

If the border-style property has one value:

border-style: dotted

all four borders are dotted

```
p {
  border-style: dotted solid double dashed;
}
/* Three values */
p {
  border-style: dotted solid double;
}
/* Two values */
p {
  border-style: dotted solid;
}
/* One value */
p {
  border-style: dotted;
}
```

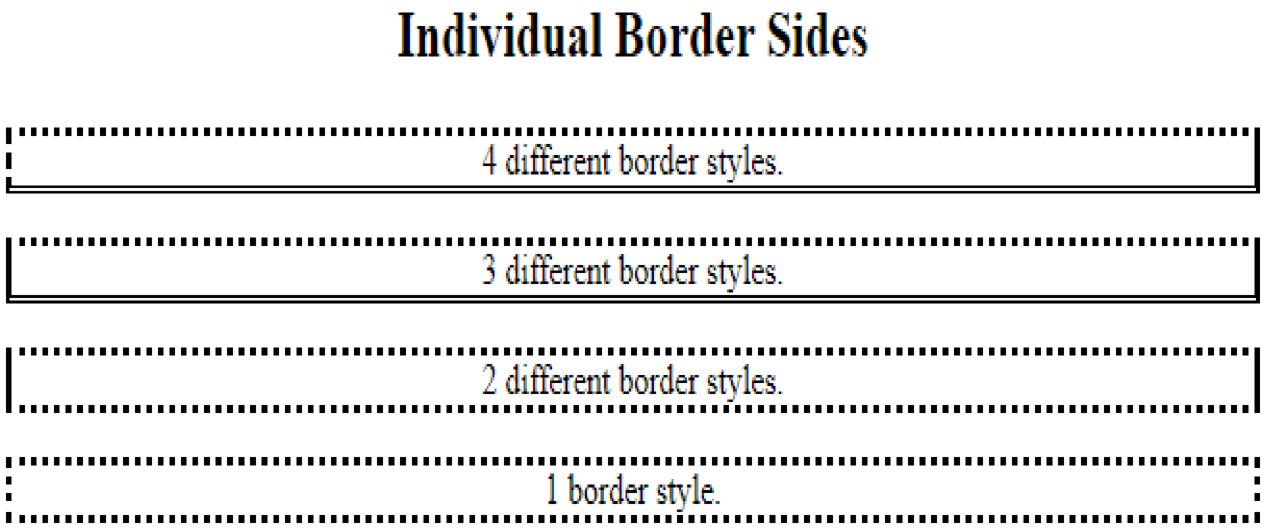

Individual Border Sides
4 different border styles.
3 different border styles.
2 different border styles.
1 border style.

Fig 1.32 CSS Border Sides

CSS Border Images

CSS border-image Property

The CSS border-image property allows you to specify an image to be used instead of the normal border around an element.

The property has three parts:

- The image to use as the border
- Where to slice the image
- Define whether the middle sections should be repeated or stretched

We will use the following image (called "border.png"):

- The border-image property takes the image and slices it into nine sections, like a tic-tac-toe board.
- It then places the corners at the corners, and the middle sections are repeated or stretched as you specify.

Note: For border-image to work, the element also needs the border property set!

Here, the middle sections of the image are repeated to create the border:

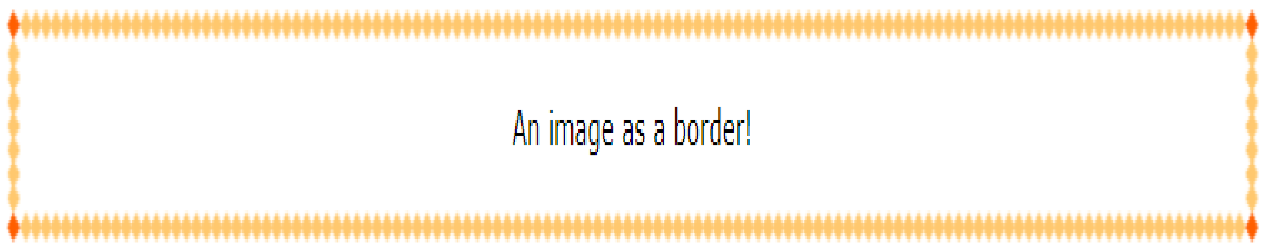
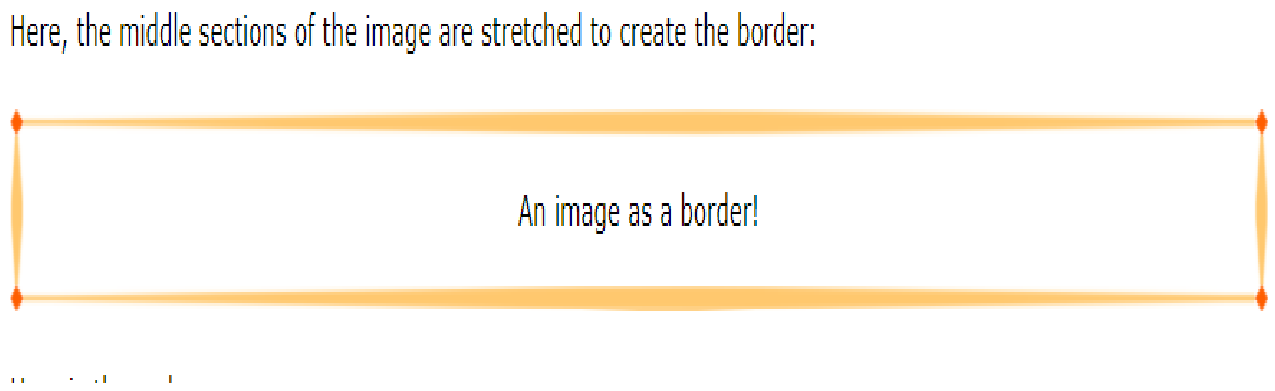


Fig 1.33 CSS Border Image

```
#borderimg {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30 round;
}
```


Here, the middle sections of the image are stretched to create the border:

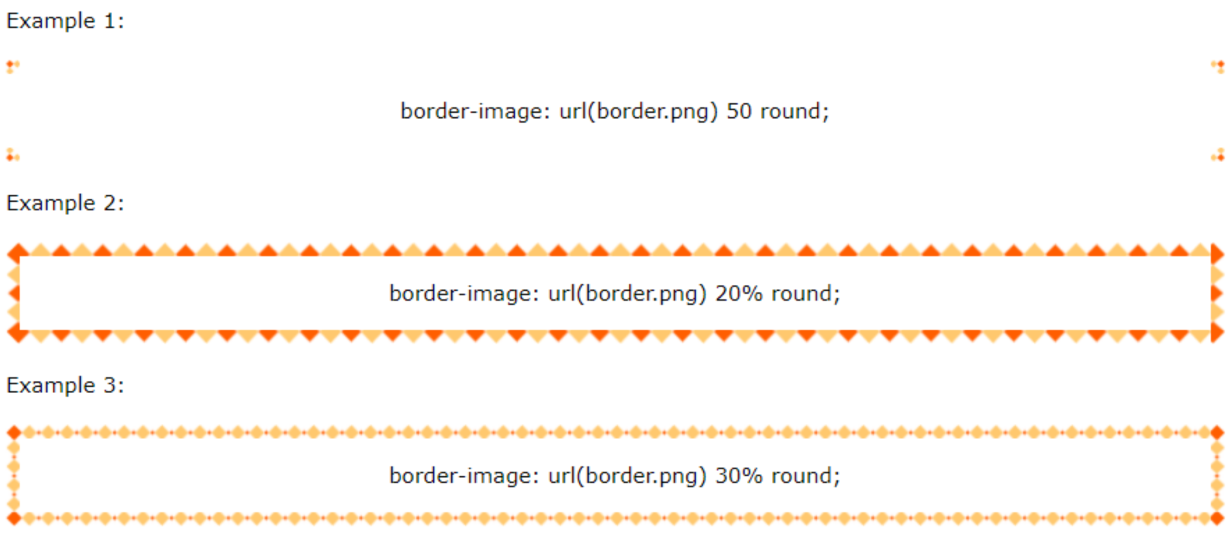
An image as a border!

Fig 1.34 CSS Border Image

```
<!DOCTYPE html>
<html>
<head>
<style>
#borderimg {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30 stretch;}
</style>
</head>
<body>
<h1>The border-image Property</h1>
<p>Here, the middle sections of the image are stretched to create the border:</p>
<p id="borderimg">border-image: url(border.png) 30 stretch;</p>
<p>Here is the original image:</p><img src="border.png">
<p><strong>Note:</strong> Internet Explorer 10, and earlier versions, do not support the
border-image property.</p></body></html>
```

CSS border-image - Different Slice Values

Different slice values completely changes the look of the border:


Example 1:

border-image: url(border.png) 50 round;

Example 2:

border-image: url(border.png) 20% round;

Example 3:

border-image: url(border.png) 30% round;

Fig 1.35 CSS Border Image –Different Slice

```
#borderimg1 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 50 round;
}
#borderimg2 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 20% round;
}
#borderimg3 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30% round;
}
```

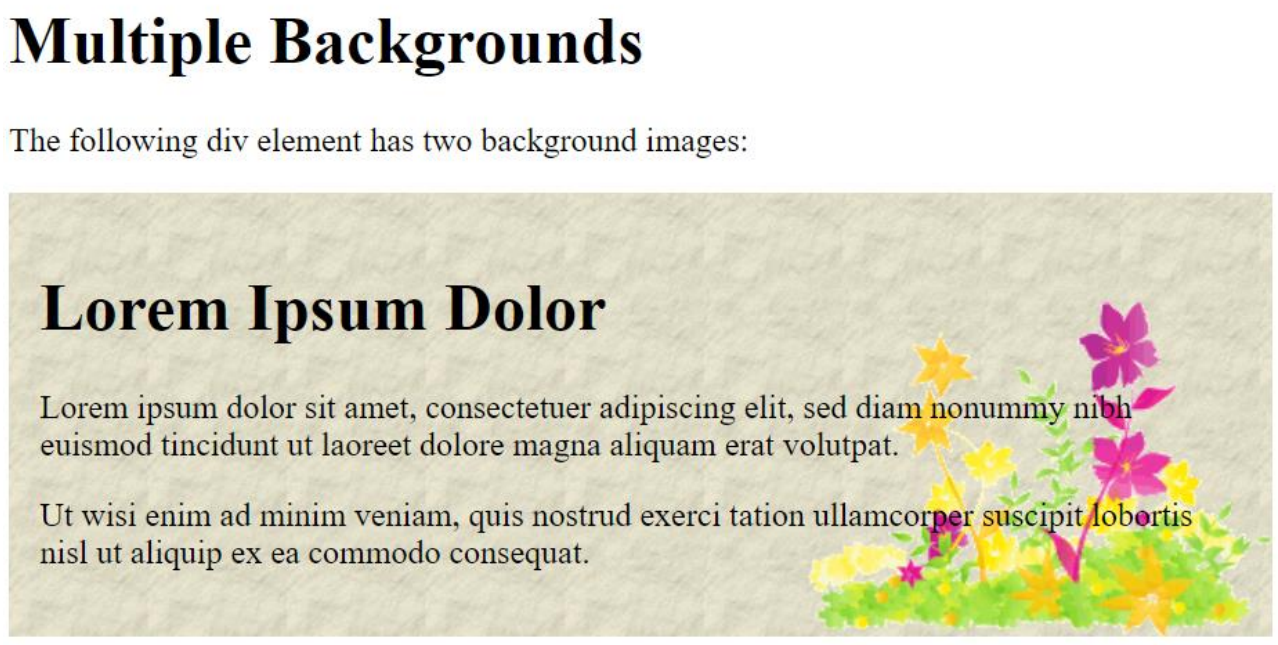
CSS Multiple Backgrounds

CSS allows you to add multiple background images for an element, through the background-image property. The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer.

The following example has two background images, the first image is a flower (aligned to the bottom and right) and the second image is a paper background (aligned to the top-left corner):

Example

```
#example1 {
  background-image: url(img_flwr.gif), url(paper.gif);
  background-position: right bottom, left top;
  background-repeat: no-repeat, repeat;
}
<!DOCTYPE html>
<html>
<head>
<style>
#example1 {
  background-image: url(img_flwr.gif), url(paper.gif);
  background-position: right bottom, left top;
  background-repeat: no-repeat, repeat;
  padding: 15px;
}
</style>
</head>
<body>
<h1>Multiple Backgrounds</h1>
<p>The following div element has two background images:</p>
<div id="example1">
  <h1>Lorem Ipsum Dolor</h1>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl
ut aliquip ex ea commodo consequat.</p>
</div>
</body>
</html>
```



Multiple Backgrounds

The following div element has two background images:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

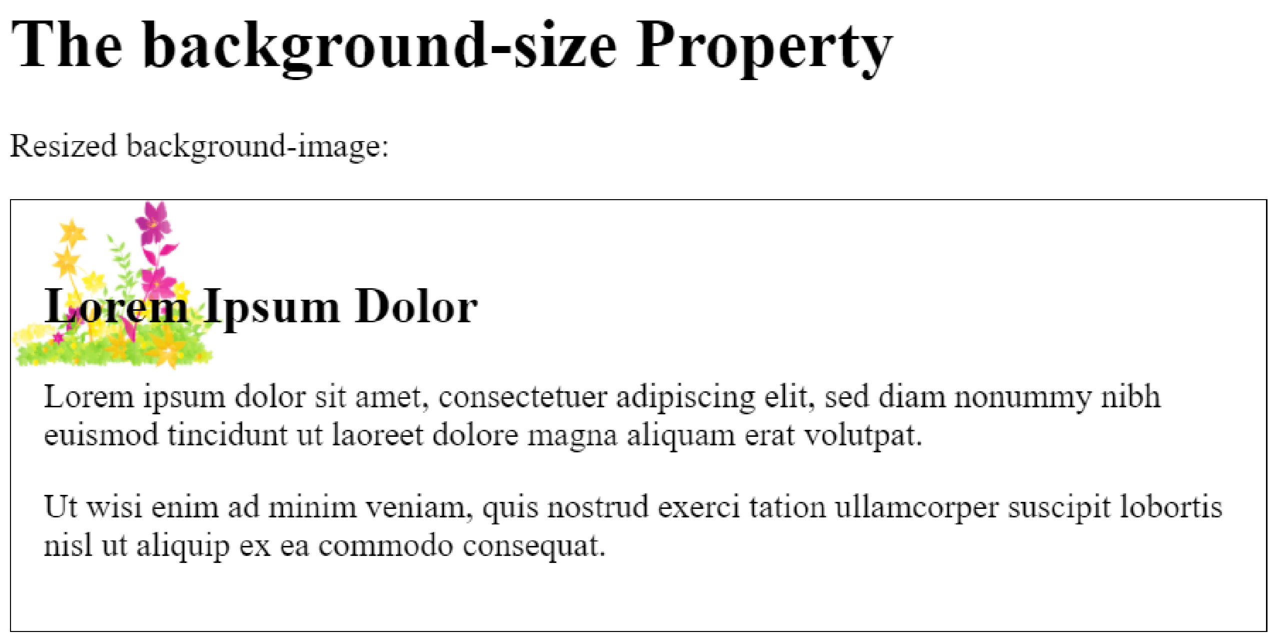
Fig 1.36 CSS Multiple Background

CSS Background Size

- The CSS background-size property allows you to specify the size of background images.
- The size can be specified in lengths, percentages, or by using one of the two keywords: contain or cover.

The following example resizes a background image to much smaller than the original image (using pixels):

```
<!DOCTYPE html>
<html>
<head>
<style>
#example1 {
  border: 1px solid black;
  background: url(img_flwr.gif);
  background-size: 100px 80px;
  background-repeat: no-repeat;
  padding: 15px;
}
#example2 {
  border: 1px solid black;
  background: url(img_flwr.gif);
  background-repeat: no-repeat;
  padding: 15px;
}
</style>
</head>
<body>
<h1>The background-size Property</h1>
<p>Resized background-image:</p>
<div id="example1">
  <h2>Lorem Ipsum Dolor</h2>
  <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl
ut aliquip ex ea commodo consequat.</p>
</div>
```



```
The background-size Property

Resized background-image:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
nisl ut aliquip ex ea commodo consequat.
```

Fig 1.37 CSS Background Size

CSS Shadow Effects

With CSS you can add shadow to text and to elements.

CSS Text Shadow

The CSS text-shadow property applies shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

```
h1 {
  text-shadow: 2px 2px;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  text-shadow: 2px 2px;
}
</style>
</head>
<body>
<h1>Text-shadow effect!</h1>
</body>
</html>
```

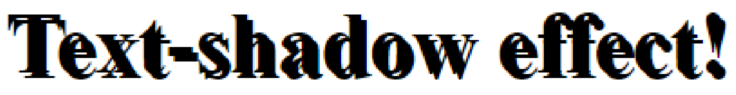


Fig 1.38 CSS Text Shadow Effect

Then, add a blur effect to the shadow:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  text-shadow: 2px 2px 5px red;
}
</style>
```

```
</head>
<body>
<h1>Text-shadow effect!</h1>
</body>
</html>
```

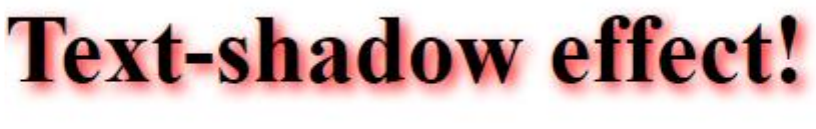


Fig 1.39 CSS Text Shadow Effect

The following example shows a white text with black shadow:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  color: white;
  text-shadow: 2px 2px 4px #000000;
}
</style>
</head>
<body>
<h1>Text-shadow effect!</h1>
</body>
</html>
```

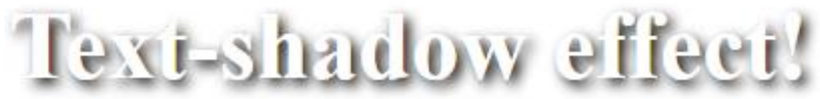


Fig 1.40 CSS Text Shadow Effect

```
<!DOCTYPE html>
```

```
<html>
<head>
<style>
h1 {
  color: white;
  text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
}
</style>
</head>
<body>
<h1>Text-shadow effect!</h1>
</body>
</html>
```

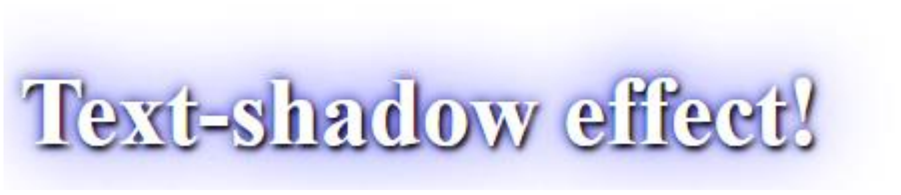


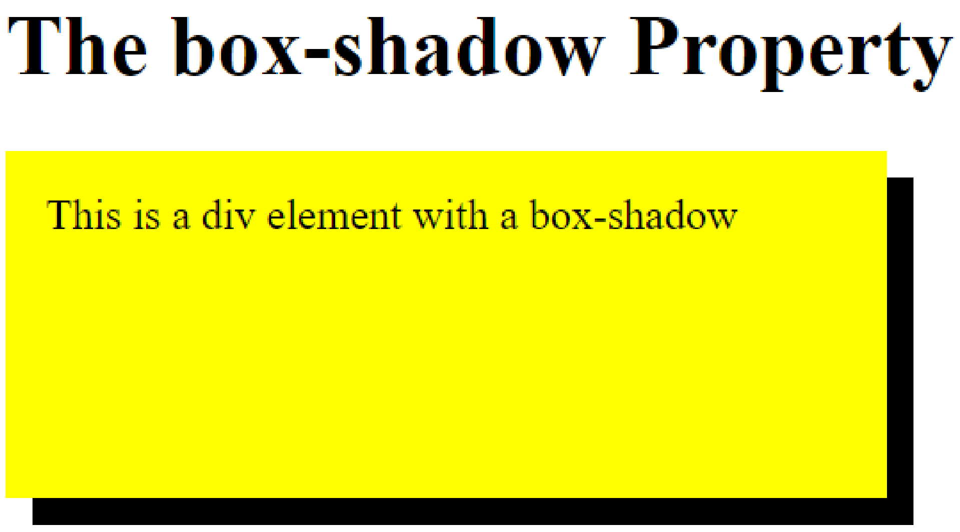
Fig 1.41 CSS Text Shadow Effect

CSS Box Shadow

CSS box-shadow Property

- The CSS box-shadow property is used to apply one or more shadows to an element.
- Specify a Horizontal and a Vertical Shadow
- In its simplest use, you only specify a horizontal and a vertical shadow.
- The default color of the shadow is the current text-color.

```
div {
  box-shadow: 10px 10px;
}
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  padding: 15px;
  background-color: yellow;
  box-shadow: 10px 10px;
}
</style>
</head>
<body>
<h1>The box-shadow Property</h1>
<div>This is a div element with a box-shadow</div>
</body>
</html>
```



The box-shadow Property

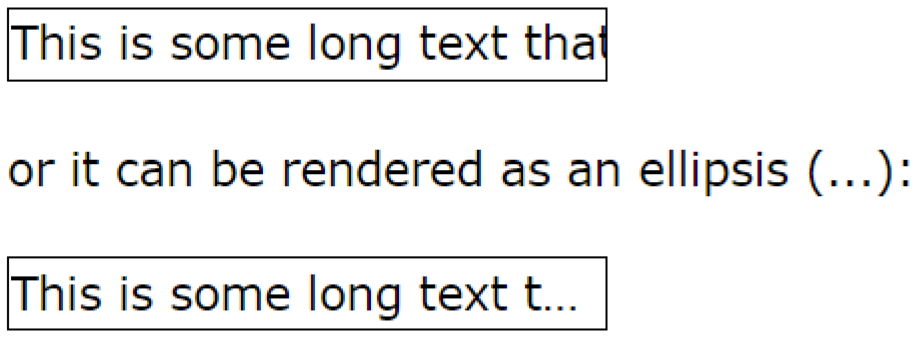
This is a div element with a box-shadow

Fig 1.42 CSS Box-Shadow

CSS Text Overflow

The CSS text-overflow property specifies how overflowed content that is not displayed should be signaled to the user.

It can be clipped:



This is some long text that

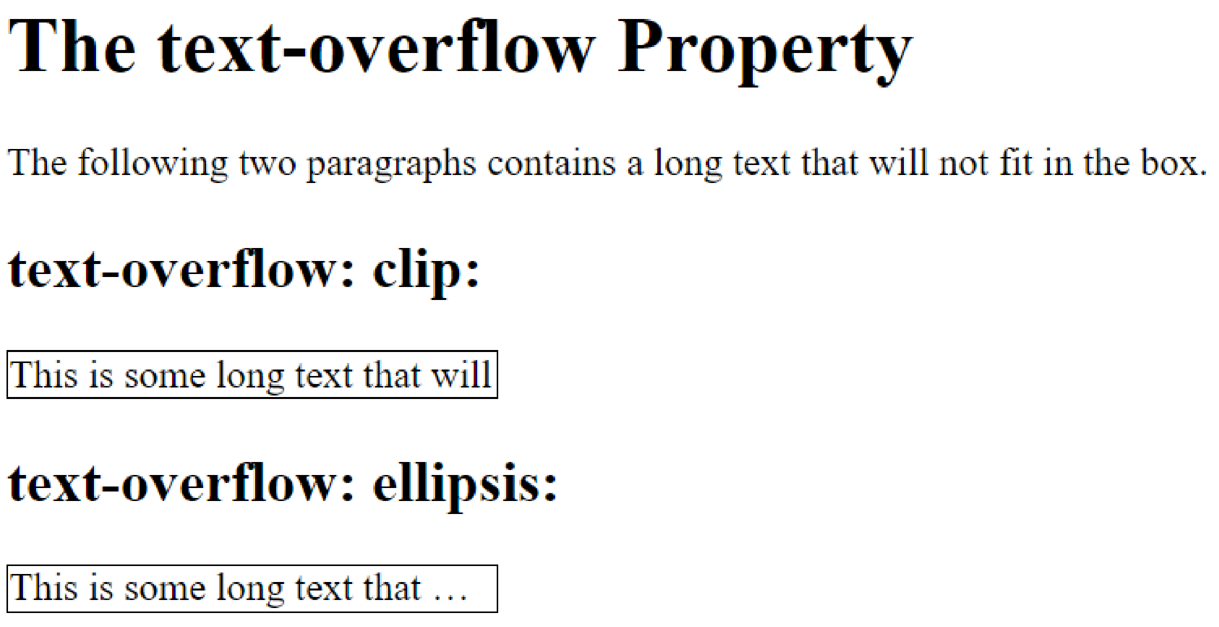
or it can be rendered as an ellipsis (...):

This is some long text t...

Fig 1.43 CSS Text Overflow

```
<!DOCTYPE html>
<html>
<head>
<style>
p.test1 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: clip;
}
p.test2 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: ellipsis;
}
</style>
</head>
<body>
<h1>The text-overflow Property</h1>
<p>The following two paragraphs contains a long text that will not fit in the box.</p>
<h2>text-overflow: clip:</h2>
```

```
<p class="test1">This is some long text that will not fit in the box</p>
<h2>text-overflow: ellipsis:</h2>
<p class="test2">This is some long text that will not fit in the box</p>
</body>
</html>
```



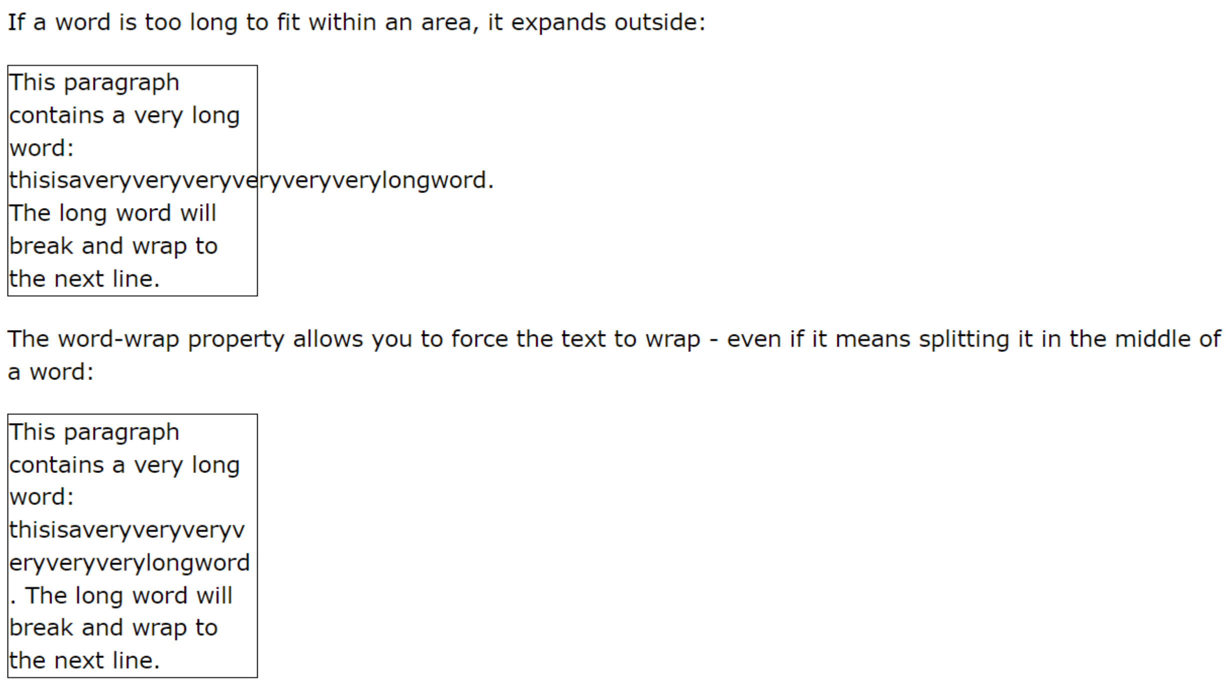
```
The text-overflow Property

The following two paragraphs contains a long text that will not fit in the box.

text-overflow: clip:

This is some long text that will

text-overflow: ellipsis:

This is some long text that ...
```

Fig 1.44 CSS Text Overflow

CSS Word Wrapping

The CSS word-wrap property allows long words to be able to be broken and wrap onto the next line.

If a word is too long to fit within an area, it expands outside:

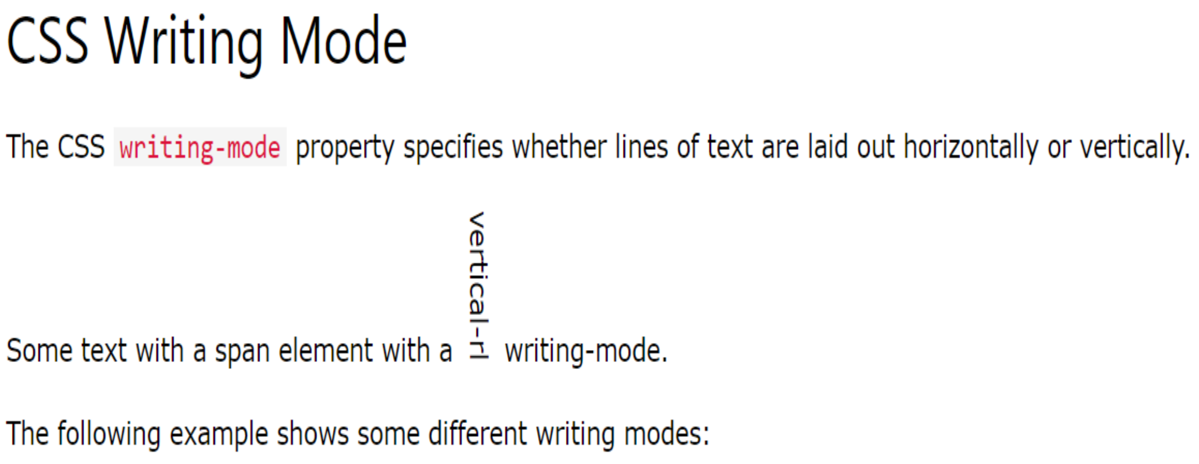

If a word is too long to fit within an area, it expands outside:

This paragraph contains a very long word: thisisaveryveryveryveryveryverylongword. The long word will break and wrap to the next line.

The word-wrap property allows you to force the text to wrap - even if it means splitting it in the middle of a word:

This paragraph contains a very long word: thisisaveryveryveryveryveryverylongword. The long word will break and wrap to the next line.

Fig 1.45 CSS Text wrapping


CSS Writing Mode

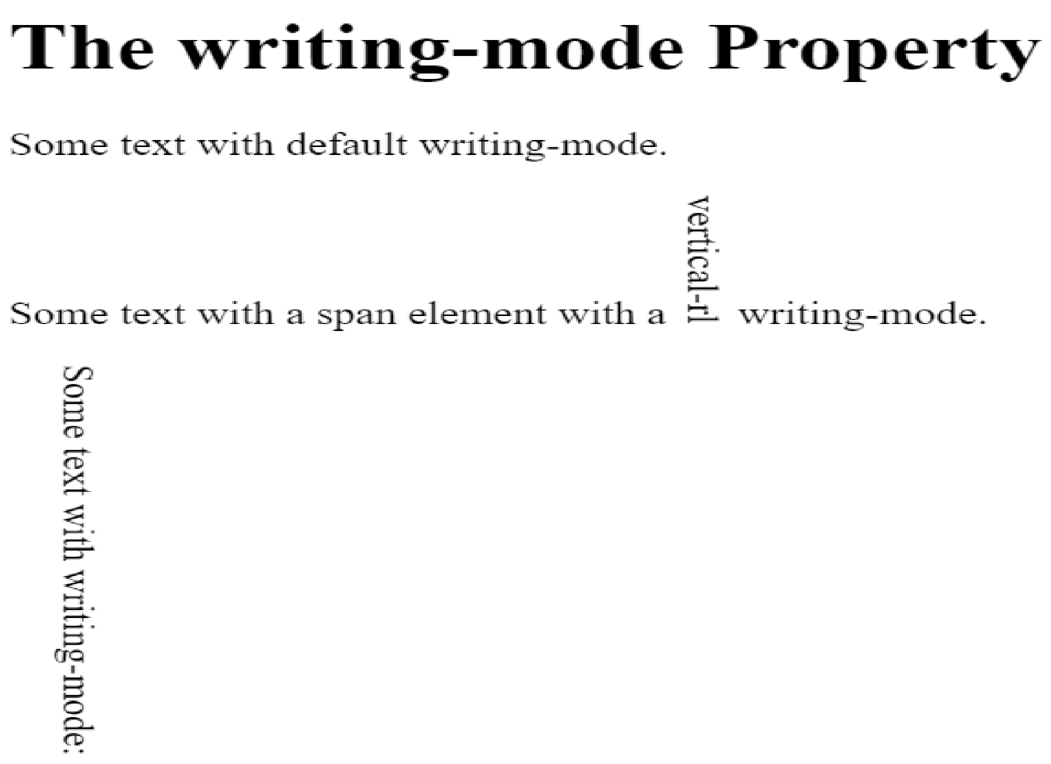
The CSS `writing-mode` property specifies whether lines of text are laid out horizontally or vertically.

Some text with a span element with a vertical-rl writing-mode.

The following example shows some different writing modes:

Fig 1.46 CSS Writing Mode

```
<!DOCTYPE html>
<html>
<head>
<style>
p.test1 {
  writing-mode: horizontal-tb;
}
span.test2 {
  writing-mode: vertical-rl;
}
p.test2 {
  writing-mode: vertical-rl;
}
</style>
</head>
<body>
<h1>The writing-mode Property</h1>
```


The writing-mode Property

Some text with default writing-mode.

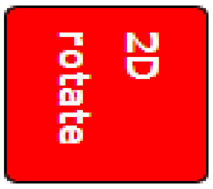
Some text with a span element with a vertical-rl writing-mode.

Some text with writing-mode:

Fig 1.47 CSS Writing Mode

CSS 2D Transforms

CSS transforms allow you to move, rotate, scale, and skew elements. Mouse over the element below to see a 2D transformation:

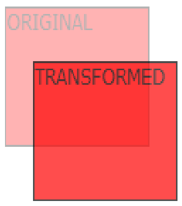


CSS 2D Transforms Methods

With the CSS transform property you can use the following 2D transformation methods:

- translate()
- rotate()
- scaleX()
- scaleY()
- scale()
- skewX()
- skewY()
- skew()
- matrix()

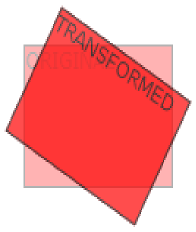
The translate() Method



The `translate()` method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

The following example moves the <div> element 50 pixels to the right, and 100 pixels down from its current position:

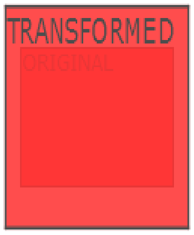
The rotate() Method



The `rotate()` method rotates an element clockwise or counter-clockwise according to a given degree.

The following example rotates the <div> element clockwise with 20 degrees:

The scale() Method



The `scale()` method increases or decreases the size of an element (according to the parameters given for the width and height).

The following example increases the <div> element to be two times of its original width, and three times of its original height:

The following example decreases the <div> element to be half of its original width and height:

```
div {
  transform: scale(0.5, 0.5);
}
```

The scaleX() Method

The scaleX() method increases or decreases the width of an element.

The following example increases the <div> element to be two times of its original width:

The following example decreases the <div> element to be half of its original width:

```
div {
  transform: scaleX(0.5);
}
```

The scaleY() Method

The scaleY() method increases or decreases the height of an element.

The following example increases the <div> element to be three times of its original height:

The following example decreases the <div> element to be half of its original height:

```
div {
  transform: scaleY(0.5);
}
```

The skewX() Method

The skewX() method skews an element along the X-axis by the given angle.

The following example skews the <div> element 20 degrees along the X-axis:

The skewY() Method

The skewY() method skews an element along the Y-axis by the given angle.

The following example skews the <div> element 20 degrees along the Y-axis:

The skew() Method

The skew() method skews an element along the X and Y-axis by the given angles.

The following example skews the <div> element 20 degrees along the X-axis, and 10 degrees along the Y-axis:

```
div {
  transform: skew(20deg, 10deg);
}
```

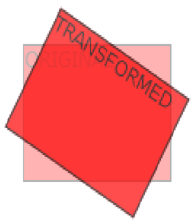
If the second parameter is not specified, it has a zero value.

So, the following example skews the <div> element 20 degrees along the X-axis:

Example

```
div {
  transform: skew(20deg);
}
```

The matrix() Method



The `matrix()` method combines all the 2D transform methods into one.

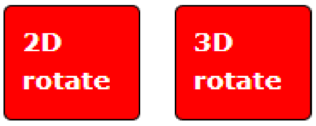
The matrix() method take six parameters, containing mathematic functions, which allows you to rotate, scale, move (translate), and skew elements.

The parameters are as follow: matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())

CSS 3D Transforms

CSS also supports 3D transformations.

Mouse over the elements below to see the difference between a 2D and a 3D transformation:

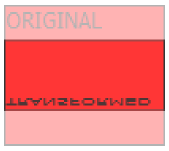


CSS 3D Transforms Methods

With the CSS `transform` property you can use the following 3D transformation methods:

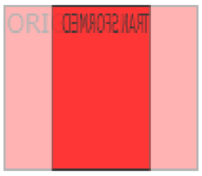
- `rotateX()`
- `rotateY()`
- `rotateZ()`

The rotateX() Method



The `rotateX()` method rotates an element around its X-axis at a given degree:

The rotateY() Method



The `rotateY()` method rotates an element around its Y-axis at a given degree:

The rotateZ() Method

The rotateZ() method rotates an element around its Z-axis at a given degree:

```
#myDiv {
  transform: rotateZ(90deg);
}
```

How to Use CSS Transitions?

To create a transition effect, you must specify two things:

- the CSS property you want to add an effect to
- the duration of the effect

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

The following example shows a 100px * 100px red <div> element.

The <div> element has also specified a transition effect for the width property, with a duration of 2 seconds:

```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
```

The transition effect will start when the specified CSS property (width) changes value.

Now, let us specify a new value for the width property when a user mouse over the <div> element:

```
div:hover {
  width: 300px;
}
```

Change Several Property Values

The following example adds a transition effect for both the width and height property, with a duration of 2 seconds for the width and 4 seconds for the height:

Example

```
div {
  transition: width 2s, height 4s;
}
```

Specify the Speed Curve of the Transition

The transition-timing-function property specifies the speed curve of the transition effect.

The transition-timing-function property can have the following values:

- ease - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- linear - specifies a transition effect with the same speed from start to end
- ease-in - specifies a transition effect with a slow start
- ease-out - specifies a transition effect with a slow end
- ease-in-out - specifies a transition effect with a slow start and end
- cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

Delay the Transition Effect

The transition-delay property specifies a delay (in seconds) for the transition effect.

The following example has a 1 second delay before starting:

Example

```
div {
  transition-delay: 1s;
}
```

Transition + Transformation

The following example adds a transition effect to the transformation:

Example

```
div {
  transition: width 2s, height 2s, transform 2s;
}
```

The CSS transition properties can be specified one by one, like this:

Example

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
```

or by using the shorthand property transition:

Example

```
div {
  transition: width 2s linear 1s;
}
```

What are CSS Animations?

An animation lets an element gradually change from one style to another. You can change as many CSS properties you want, as many times as you want. To use CSS animation, you must first specify some keyframes for the animation. Keyframes hold what styles the element will have at certain times.

CSS Animations

CSS allows animation of HTML5 elements without using JavaScript or Flash!

- @keyframes
- animation-name
- animation-duration
- animation-delay
- animation-iteration-count
- animation-direction
- animation-timing-function
- animation-fill-mode
- animation

The @keyframes Rule

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times. To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the <div> element.

The animation will last for 4 seconds, and it will gradually change the background-color of the <div> element from "red" to "yellow":

```
@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}
/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

Note: The animation-duration property defines how long an animation should take to complete.

If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds). In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)). It is also possible to use percent. By using percent, you can add as many style changes as you like.

The following example will change the background-color of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

```
/* The animation code */
@keyframes example {
  0%   {background-color: red;}
  25%  {background-color: yellow;}
  50%  {background-color: blue;}
  100% {background-color: green;}
}
/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

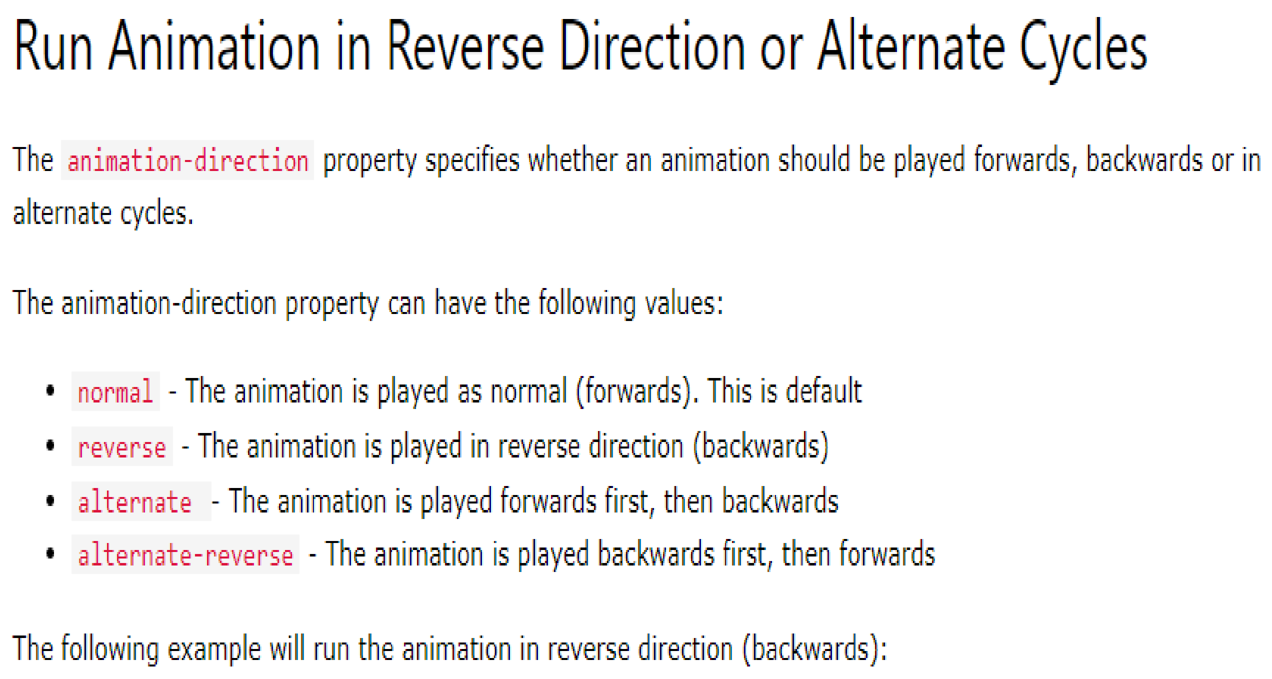
The following example will change both the background-color and the position of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

```
/* The animation code */
@keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}
/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```
Delay an Animation
The animation-delay property specifies a delay for the start of an animation.
The following example has a 2 seconds delay before starting the animation:
Example
```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-delay: 2s;
}
```
Delay an Animation
Negative values are also allowed. If using negative values, the animation will start as if it had already been playing for N seconds.
In the following example, the animation will start as if it had already been playing for 2 seconds:
Example
```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-delay: -2s;
}
```
Set How Many Times an Animation Should Run
The animation-iteration-count property specifies the number of times an animation should run.
The following example will run the animation 3 times before it stops:
Example
```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: 3;
}
```

The following example uses the value "infinite" to make the animation continue for ever:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: infinite;
}
```



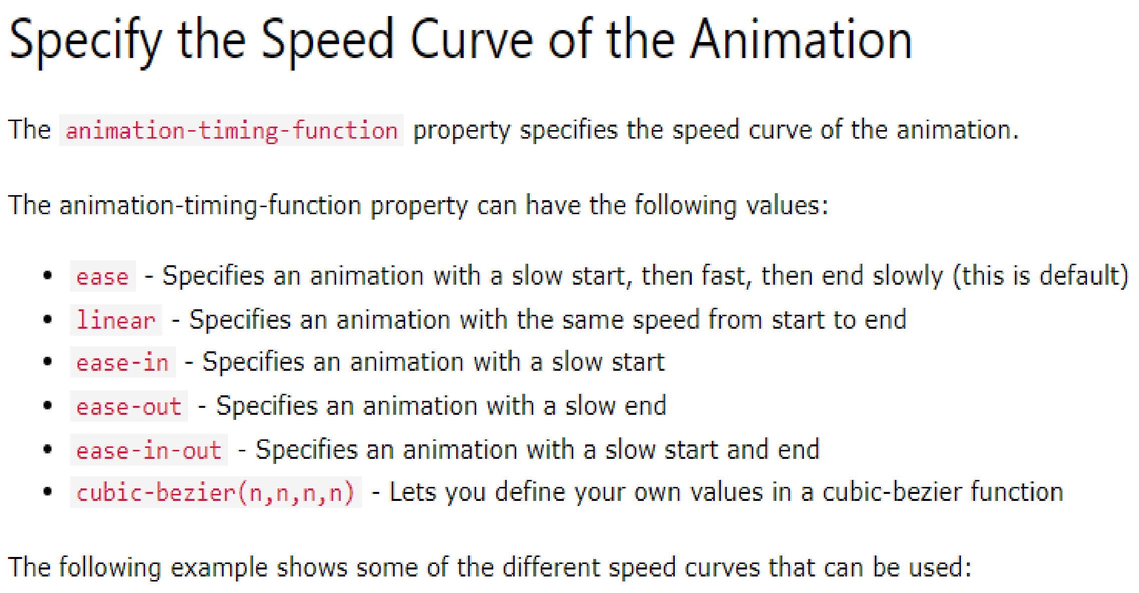
Run Animation in Reverse Direction or Alternate Cycles

The `animation-direction` property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

- `normal` - The animation is played as normal (forwards). This is default
- `reverse` - The animation is played in reverse direction (backwards)
- `alternate` - The animation is played forwards first, then backwards
- `alternate-reverse` - The animation is played backwards first, then forwards

The following example will run the animation in reverse direction (backwards):



Specify the Speed Curve of the Animation

The `animation-timing-function` property specifies the speed curve of the animation.

The animation-timing-function property can have the following values:

- `ease` - Specifies an animation with a slow start, then fast, then end slowly (this is default)
- `linear` - Specifies an animation with the same speed from start to end
- `ease-in` - Specifies an animation with a slow start
- `ease-out` - Specifies an animation with a slow end
- `ease-in-out` - Specifies an animation with a slow start and end
- `cubic-bezier(n,n,n,n)` - Lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

Animation Shorthand Property

The example below uses six of the animation properties:

Example

```
div {
  animation-name: example;
  animation-duration: 5s;
  animation-timing-function: linear;
  animation-delay: 2s;
  animation-iteration-count: infinite;
  animation-direction: alternate; }
```

The same animation effect as above can be achieved by using the shorthand animation property:

Example

```
div {

  animation: example 5s linear 2s infinite alternate;

}
```

TEXT / REFERENCE BOOKS

1.Harvey and Paul Deitel & Associates, Harvey Deitel and Abbey Deitel, “Internet and World Wide Web - How to Program”, 5th Edition, Pearson Education, 2011.

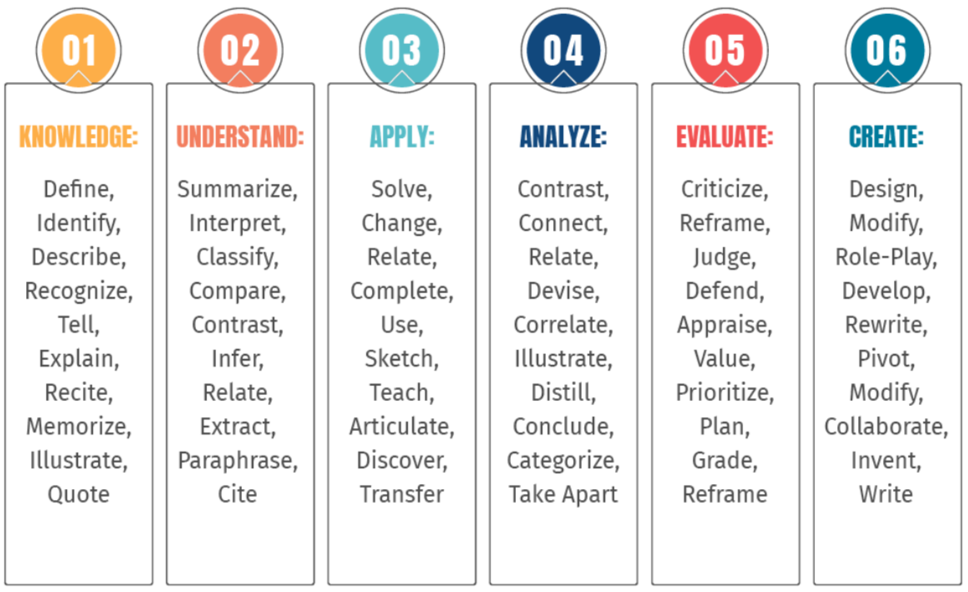
2.Achyut S Godbole and Atul Kahate, “Web Technologies”, 2nd Edition, Tata McGraw Hill, 2012.

Question Bank

Part-A			
Q.No	**Questions (2 Marks)**	**Competence**	**BT Level**
1.	Define HTML5.	Knowledge	BTL1
2.	State the features of HTML5.	Knowledge	BTL1
3.	Summarize the various HTML sematic tags.	Understand	BTL2
4.	Illustrate on how will the audio and video be embedded into an HTML document?	Apply	BTL3
5.	Write the syntax to create canvas area in an HTML document.	Create	BTL6
6.	Compare and contrast local and session storage.	Analyze	BTL4
7.	List out the features of CSS3.	Knowledge	BTL1
8.	Develop a program to create the box shadow using CSS.	Create	BTL6
9.	List out the methods used to perform 2D transformations.	Knowledge	BTL1
10.	Write the syntax to create transition delay over an HTML element.	Create	BTL6
11.	Write the syntax to create border image around an HTML element.	Create	BTL6
12.	State the method used to acquire the location coordinates of the user.	Knowledge	BTL1
13.	Construct the code to embed an image into an HTML document with various image attributes.	Create	BTL6
14.	Explain about drag and drop feature briefly.	Understand	BTL2
15.	Summarize the features of CSS3 features.	Understand	BTL2
Part-B			
Q.No	**Questions (16 marks)**	**Competence**	**BT Level**

1.	Explain various sematic elements of HTML with example program.	Understand	BTL2
2.	Write a HTML program to create a form with different input elements.	Create	BTL6
3.	Write a program to draw a line and circle with Canvas API.	Create	BTL6
4.	Show in detail about HTML web storage objects for storing the data on the client.	Apply	BTL3
5.	Classify the various kinds of properties of CSS text effects with example programs.	Analyze	BTL4
6.	Explain about 2D transforms and the various methods with example program.	Understand	BTL2
7.	Demonstrate the working process of CSS transitions with transition delay property.	Apply	BTL3
8.	Develop an HTML program to create a border image around an HTML element.	Create	BTL6
9.	Develop a program to drag and drop the elements from one location to another location using the features and methods available in HTML5.	Create	BTL6
10.	Use HTML and CSS to create multiple backgrounds to display the information of a marketing website.	Apply	BTL3

Bloom's Taxonomy



01	02	03	04	05	06
KNOWLEDGE:	UNDERSTAND:	APPLY:	ANALYZE:	EVALUATE:	CREATE:
Define, Identify, Describe, Recognize, Tell, Explain, Recite, Memorize, Illustrate, Quote	Summarize, Interpret, Classify, Compare, Contrast, Infer, Relate, Extract, Paraphrase, Cite	Solve, Change, Relate, Complete, Use, Sketch, Teach, Articulate, Discover, Transfer	Contrast, Connect, Relate, Devise, Correlate, Illustrate, Distill, Conclude, Categorize, Take Apart	Criticize, Reframe, Judge, Defend, Appraise, Value, Prioritize, Plan, Grade, Reframe	Design, Modify, Role-Play, Develop, Rewrite, Pivot, Modify, Collaborate, Invent, Write

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

UNIT – II Responsive Web Design (RWD) – SITA3004

II HTML Responsive Web Design

Responsive Design: What is RWD – Introduction to RWD Techniques – Fluid Layout, Fluid Images and Media queries - Introduction to RWD Frame work. Twitter Bootstrap – Bootstrap Background and Features - Getting Started with Bootstrap - Demystifying Grids – Off Canvas - Bootstrap Components - JS Plugins – Customization

Responsive web design is about creating web pages that look good on all devices! A responsive web design will automatically adjust for different screen sizes and viewports.

What is Responsive Web Design?

Responsive web design makes your web page look good on all devices. It uses only HTML and CSS. It is not a program or a JavaScript. Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device. Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device.
RWD is a design technique that allows website content to adapt to multiple devices screen sizes. This technique consists of three basic elements:

Flexible Grid-Based Layout
Flexible Media
Media Queries

Within the flexible grid-based layout of RWD, the website content can be viewed large on a desktop screen and then shift itself around to fit within a tablet or mobile phone screen. Rather than rely on pixels and points, the grid uses percentages to adjust its page sizes.
Within the flexible grid-based layout of RWD, the website content can be viewed large on a desktop screen and then shift itself around to fit within a tablet or mobile phone screen. Rather than rely on pixels and points, the grid uses percentages to adjust its page sizes.
Flexible media elements such as pictures or videos are coded to be responsive to the screen they are being displayed and served on in RWD.
Media Queries were introduced in CSS3, the latest evolution of the Cascading Style Sheets language, as a technique to help define screen breakpoints. The breakpoints allow for styling and layout of website design elements for different screen sizes and are a key part of RWD.

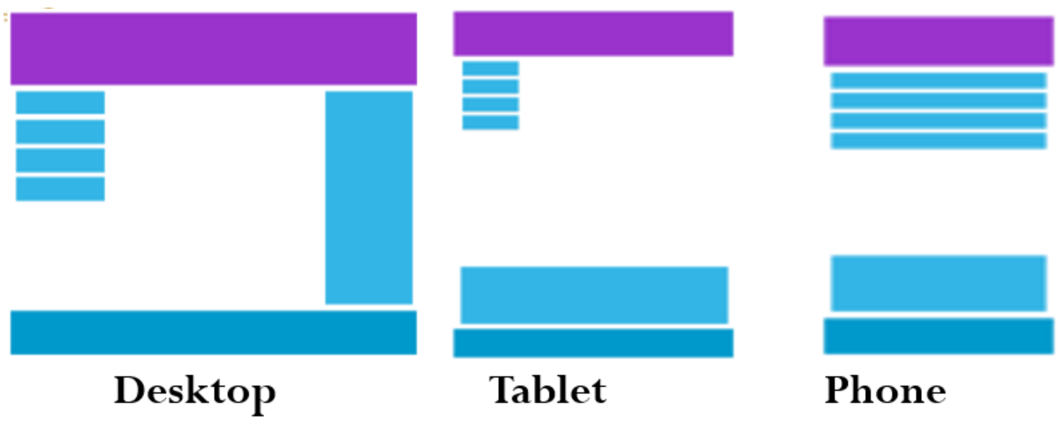


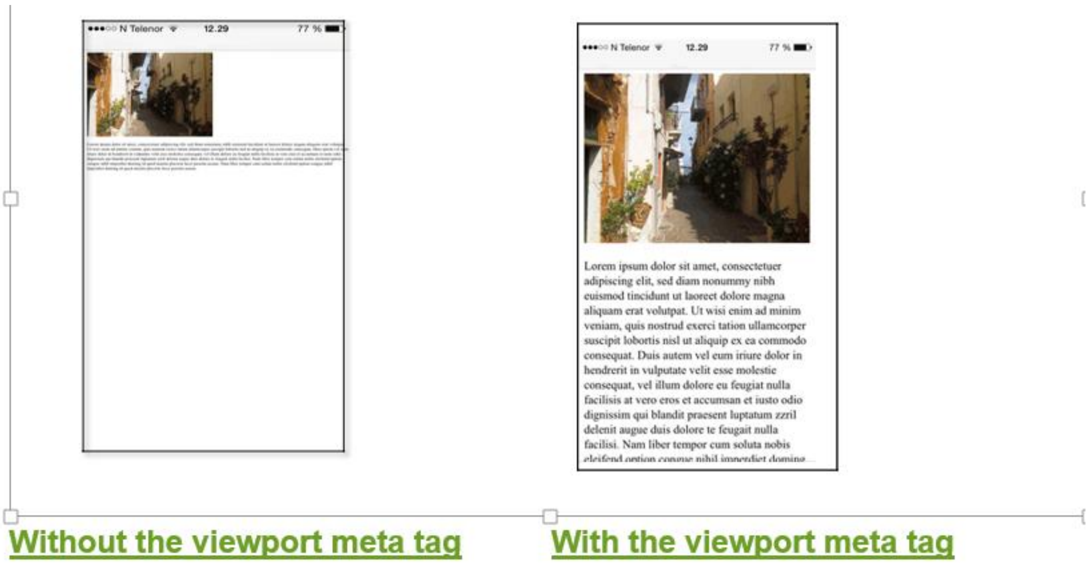
Fig 2.1 RWD Different Device View

What is The Viewport?
The viewport is the user's visible area of a web page. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen. Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size. Then, tablets and mobile phones were used for surfing the internet, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

Setting The Viewport

- HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.
- You should include the following <meta> viewport element in all your web pages:
- <meta name="viewport" content="width=device-width, initial-scale=1.0">
- This gives the browser instructions on how to control the page's dimensions and scaling.
- The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).
- The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:


●●●○○ N Telenor 12.29 77 %
●●●○○ N Telenor 12.29 77 %
Without the viewport meta tag
With the viewport meta tag

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

Fig 2.2 View Port

Size Content to The Viewport

Users are used to scroll websites vertically on both desktop and mobile devices. So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Some additional rules to follow:

Do NOT use large fixed width elements. Do NOT let the content rely on a particular viewport width to render well. Use CSS media queries to apply different styling for small and large screens

What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns: Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

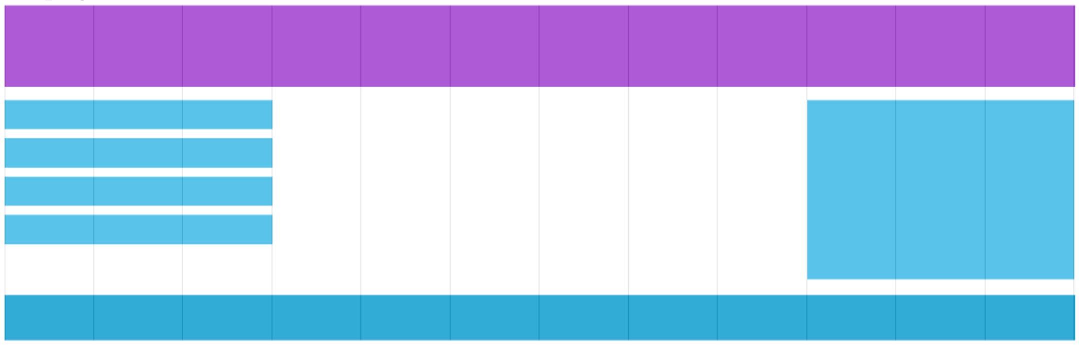
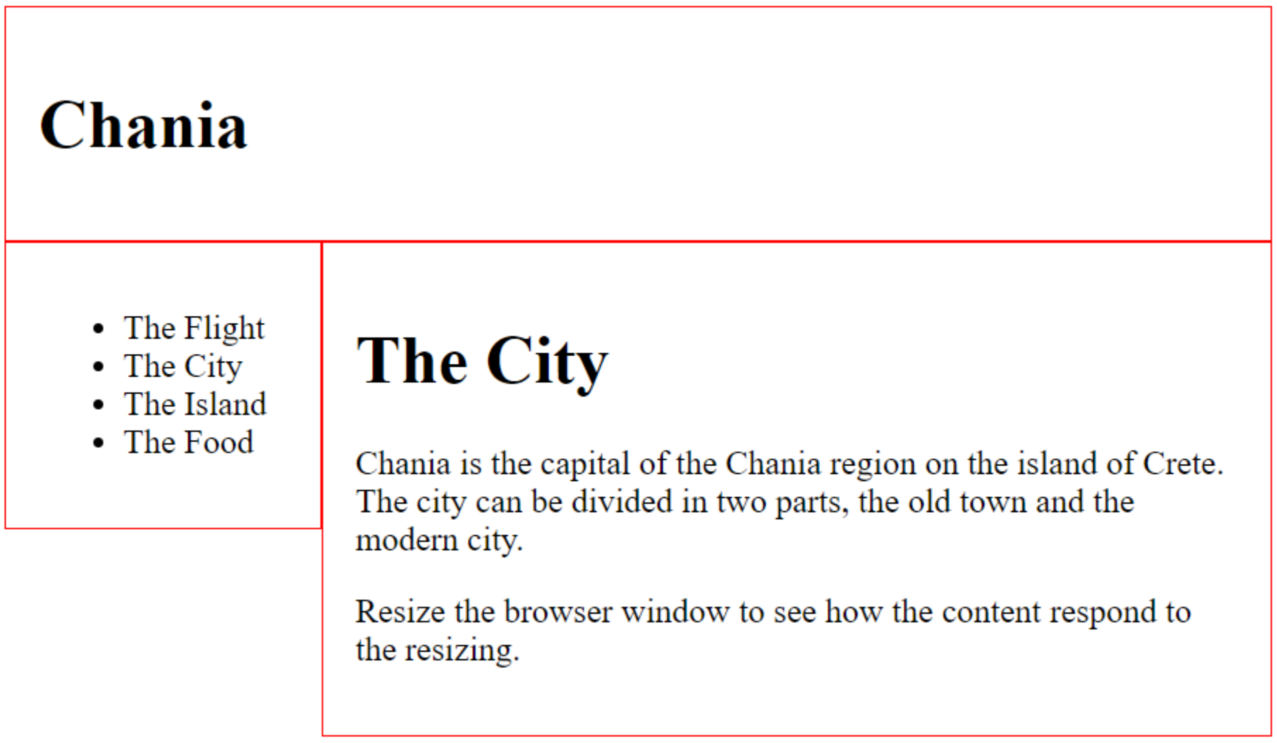


Fig 2.3 Grid View

A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

```
<!DOCTYPE html>
        <html>
        <head>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}
.header {
  border: 1px solid red;
  padding: 15px;
}
.menu {
  width: 25%;
  float: left;
  padding: 15px;
  border: 1px solid red;
}
.main {
  width: 75%;
  float: left;
  padding: 15px;
  border: 1px solid red;
}
</style>
</head>
<body>
<div class="header">
  <h1>Chania</h1>
</div>
<div class="menu">
  <ul>
    <li>The Flight</li>
    <li>The City</li>
    <li>The Island</li>
    <li>The Food</li>
  </ul>
</div>
<div class="main">
  <h1>The City</h1>
  <p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>
</body>
</html>
```

Output


Chania

- The Flight
- The City
- The Island
- The Food

The City

Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.

Resize the browser window to see how the content respond to the resizing.

Fig 2.4 Grid View output

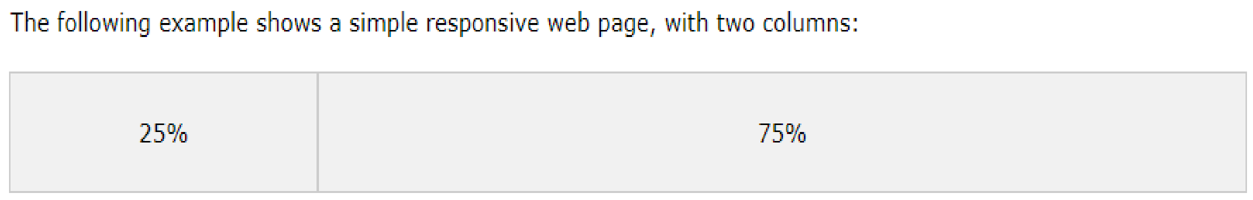
Building a Responsive Grid-View

First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.
Add the following code in your CSS:

```
    {
    box-sizing: border-box;
    }
```

Example

```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```


The following example shows a simple responsive web page, with two columns:

25%	75%

Fig 2.5 Grid View output

The example above is fine if the web page only contains two columns. However, we want to use a responsive grid-view with 12 columns, to have more control over the web page. First we must calculate the percentage for one column: 100% / 12 columns = 8.33%. Then we make one class for each of the 12 columns, class="col-" and a number defining how many columns the section should span:

CSS:

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

All these columns should be floating to the left, and have a padding of 15px:

CSS:

```
[class*="col-"] {
  float: left;
  padding: 15px;
  border: 1px solid red;
}
```

Each row should be wrapped in a <div>.

The number of columns inside a row should always add up to 12:

HTML:

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
</div>
```

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist.

To prevent this, we will add a style that clears the flow:

CSS:

```
.row::after {
  content: "";
  clear: both;
  display: table;
}
```

We also want to add some styles and colors to make it look better:

Example

```
html {
  font-family: "Lucida Sans", sans-serif;
}
.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color :#33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
  background-color: #0099cc;
}
```

What is a Media Query?

Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true.

Example

If the browser window is 600px or smaller, the background color will be lightblue:

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

What is a Breakpoint?

Essentially, media query breakpoints are pixel values that a developer/designer can define in CSS. When a responsive website reaches those pixel values, a transformation (such as the one detailed above) occurs so that the website offers an optimal user experience. Breakpoints are points where the website content responds according to the device width, allowing you to show the best possible layout to the user. CSS breakpoints are also called media query breakpoints, as they are used with media query.

Always Design for Mobile First

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).

This means that we must make some changes in our CSS.
Instead of changing styles when the width gets smaller than 768px, we should change the design when the width gets larger than 768px. This will make our design Mobile First:
Example

```
    /* For mobile phones: */
    [class*="col-"] {
      width: 100%;
    }
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

Add a Breakpoint

You can add as many breakpoints as you like. We will also insert a breakpoint between tablets and mobile phones.

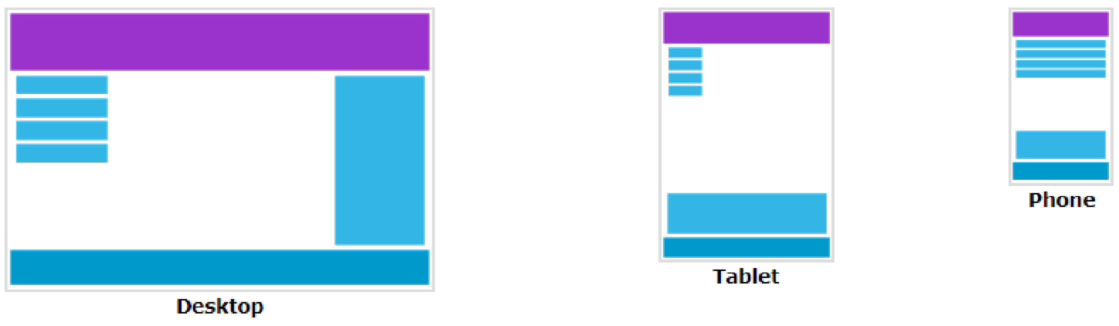


Fig 2.6 Breakpoint

We do this by adding one more media query (at 600px), and a set of new classes for devices larger than 600px (but smaller than 768px):

Example

Note that the two sets of classes are almost identical, the only difference is the name (col- and col-s-):

```
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}


@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}


@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

It might seem odd that we have two sets of identical classes, but it gives us the opportunity in HTML, to decide what will happen with the columns at each breakpoint:

HTML Example

For desktop:
The first and the third section will both span 3 columns each. The middle section will span 6 columns.
For tablets:
The first section will span 3 columns, the second will span 9, and the third section will be displayed below the first two sections, and it will span 12 columns:

```
<div class="row">
  <div class="col-3 col-s-3">...</div>
  <div class="col-6 col-s-9">...</div>
  <div class="col-3 col-s-12">...</div>
</div>
```

Typical Device Breakpoints

There are tons of screens and devices with different heights and widths, so it is hard to create an exact breakpoint for each device. To keep things simple you could target five groups:
Example

```
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {...}
/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {...}
/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {...}
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {...}
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}
```

Orientation: Portrait / Landscape
Media queries can also be used to change layout of a page depending on the orientation of the browser.
You can have a set of CSS properties that will only apply when the browser window is wider than its height, a so called "Landscape" orientation:
Example
The web page will have a lightblue background if the orientation is in landscape mode:

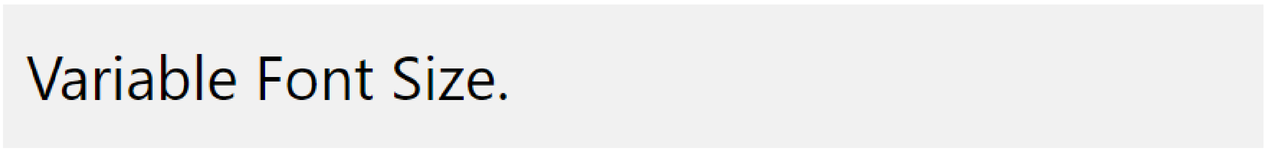
```
@media only screen and (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
```

Hide Elements With Media Queries
Another common use of media queries, is to hide elements on different screen sizes:
I will be hidden on small screens.
Example

```
/* If the screen size is 600px wide or less, hide the element */
@media only screen and (max-width: 600px) {
  div.example {
    display: none;
  }
}
```

Change Font Size With Media Queries

You can also use media queries to change the font size of an element on different screen sizes:



Variable Font Size.

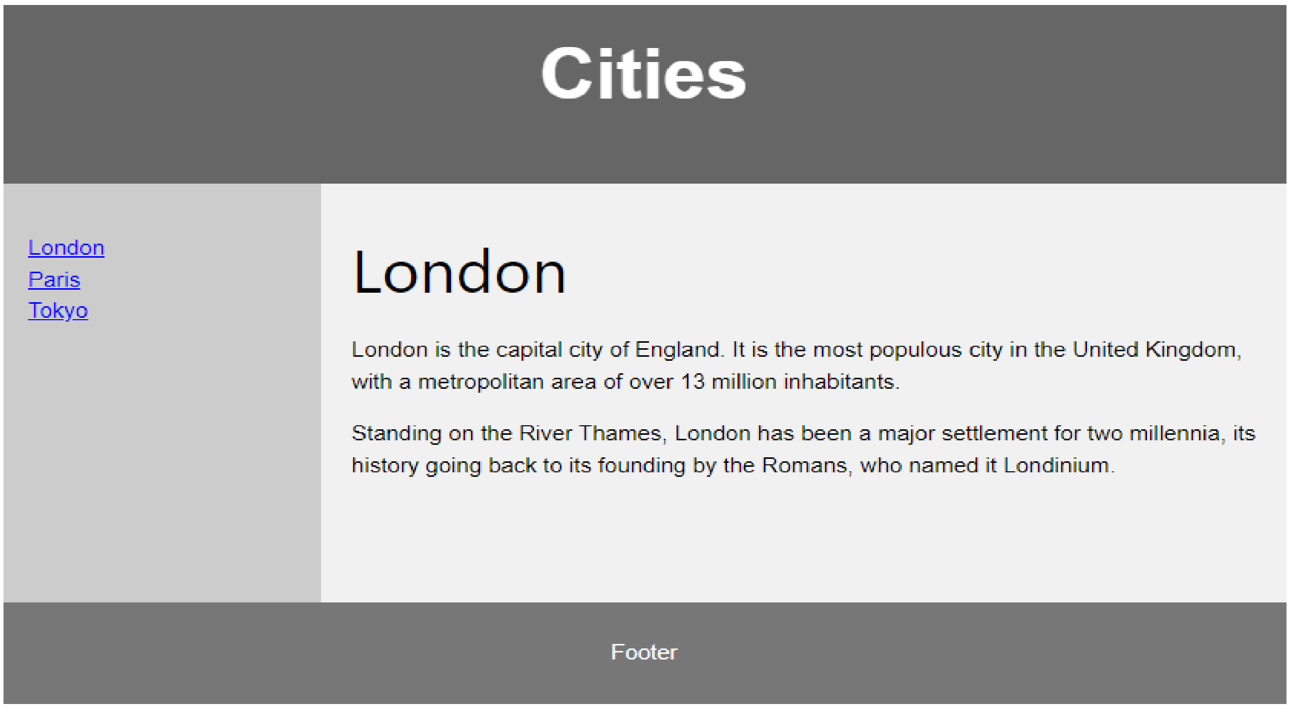
Fig 2.7 Font size

Example

```
/* If the screen size is 601px or more, set the font-size of <div> to 80px */
@media only screen and (min-width: 601px) {
  div.example {
    font-size: 80px;
  }
}
/* If the screen size is 600px or less, set the font-size of <div> to 30px */
@media only screen and (max-width: 600px) {
  div.example {
    font-size: 30px;
  }
}
```

Layout Elements and Techniques

Websites often display content in multiple columns (like a magazine or a newspaper).



Cities

London
Paris
Tokyo

London

London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.

Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

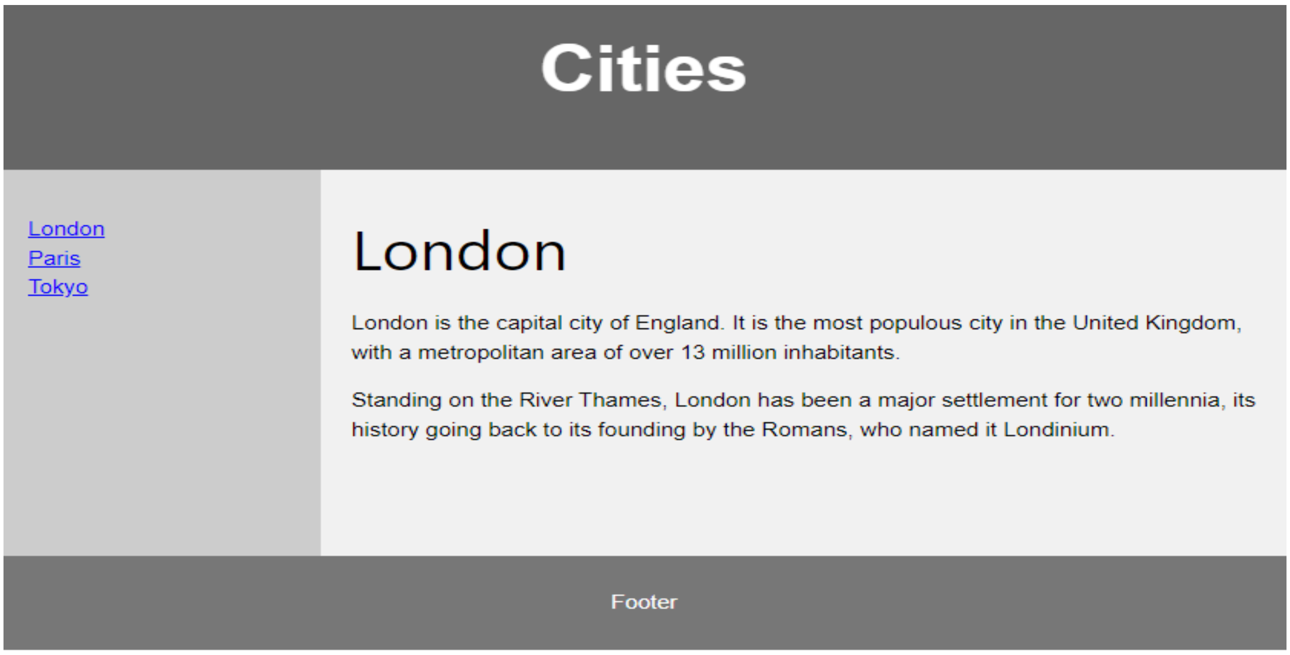
Footer

Fig 2.8 Layouts

HTML Layout Techniques
There are four different techniques to create multicolumn layouts.
Each technique has its pros and cons:

- CSS framework
- CSS float property
- CSS flexbox
- CSS grid

CSS Float Layout
It is common to do entire web layouts using the CSS float property. Float is easy to learn - you just need to remember how the float and clear properties work.
Disadvantages:
Floating elements are tied to the document flow, which may harm the flexibility.


Cities

London
Paris
Tokyo

London

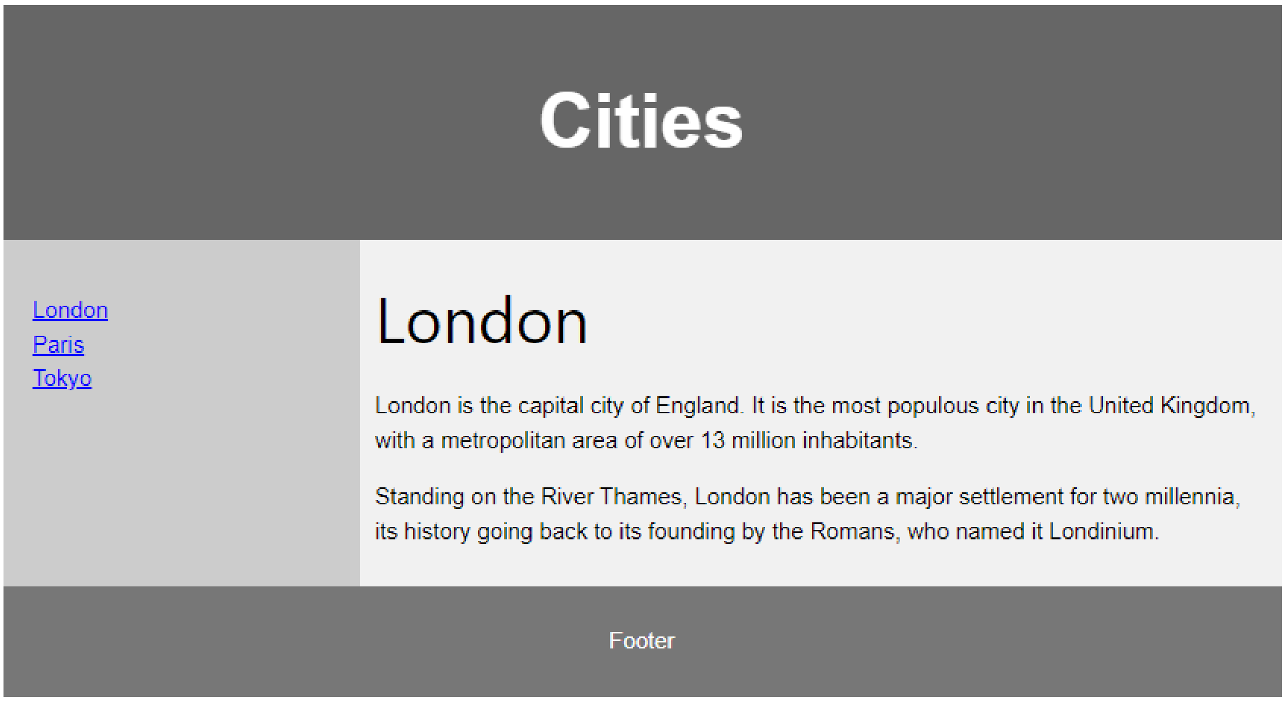
London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.

Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

Fig 2.9 Float Layout

CSS Flexbox Layout
Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices.


Cities

London
Paris
Tokyo

London

London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.

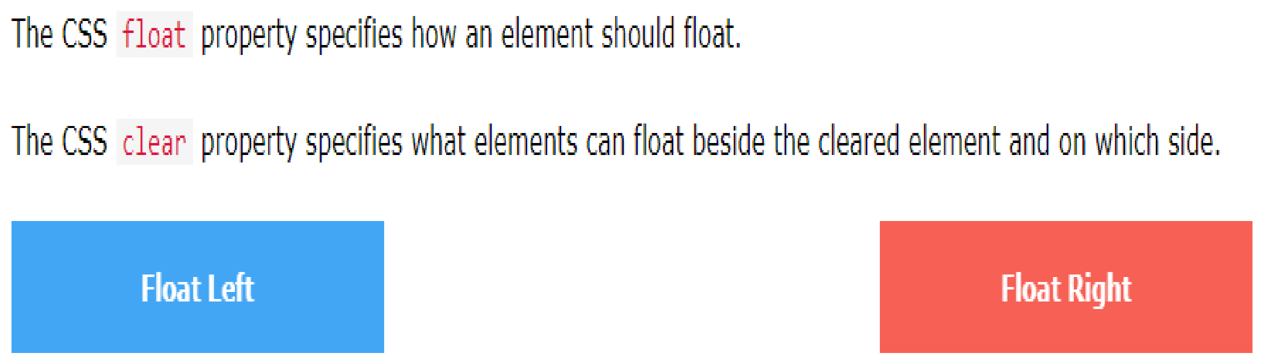
Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

Fig 2.10 Flex Layout

CSS Grid Layout
The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

CSS Layout - float and clear


The CSS `float` property specifies how an element should float.

The CSS `clear` property specifies what elements can float beside the cleared element and on which side.

Fig 2.11 CSS Layout – float and clear

The float Property

The float property is used for positioning and formatting content e.g. let an image float left to the text in a container.

The float property can have one of the following values:

- left - The element floats to the left of its container
- right - The element floats to the right of its container
- none - The element does not float (will be displayed just where it occurs in the text). This is default
- inherit - The element inherits the float value of its parent

In its simplest use, the float property can be used to wrap text around images.

Example - float: right;

The following example specifies that an image should float to the right in a text:

Example

```
img {
  float: right;
}
```

Example - No float

In the following example the image will be displayed just where it occurs in the text (float: none;):

Example

```
img {
  float: none;
}
```

The clear Property

When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.

The clear property specifies what should happen with the element that is next to a floating element.

The clear property can have one of the following values:

- none - The element is not pushed below left or right floated elements. This is default
- left - The element is pushed below left floated elements
- right - The element is pushed below right floated elements
- both - The element is pushed below both left and right floated elements

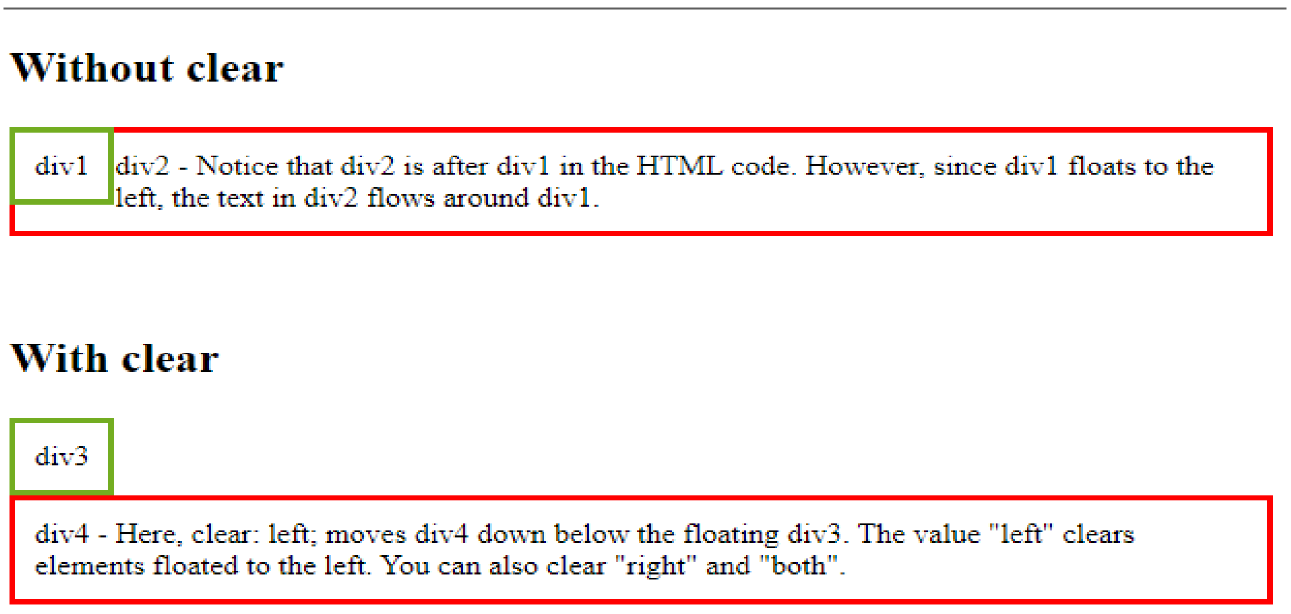
- inherit - The element inherits the clear value from its parent

When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.

Example

This example clears the float to the left. Here, it means that the <div2> element is pushed below the left floated <div1> element:

```
div1 {
  float: left;
}
div2 {
  clear: left;
}
```



Without clear

div1 div2 - Notice that div2 is after div1 in the HTML code. However, since div1 floats to the left, the text in div2 flows around div1.

With clear

div3

div4 - Here, clear: left; moves div4 down below the floating div3. The value "left" clears elements floated to the left. You can also clear "right" and "both".

Fig 2.12 With Clear and Without Clear method

The clearfix Hack

If a floated element is taller than the containing element, it will "overflow" outside of its container. We can then add a clearfix hack to solve this problem:

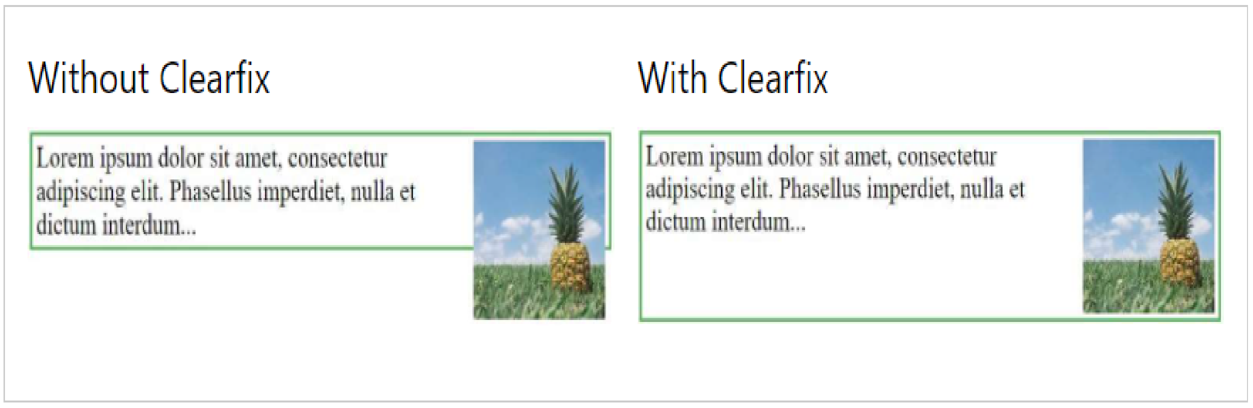

Without Clearfix
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...
With Clearfix
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

Fig 2.13 With Clearfix and Without Clearfix method

What is box-sizing?

You can easily create three floating boxes side by side. However, when you add something that enlarges the width of each box (e.g. padding or borders), the box will break. The box-sizing property allows us to include the padding and border in the box's total width (and height), making sure that the padding stays inside of the box and that it does not break.

Example

```
.img-container {
  float: left;
  width: 33.33%; /* three containers (use 25% for four, and 50% for two, etc) */
  padding: 5px; /* if you want space between the images */
}
```

Bootstrap

Bootstrap is a powerful front-end framework for faster and easier web development. It includes HTML and CSS based design templates for creating common user interface components like forms, buttons, navigations, dropdowns, alerts, modals, tabs, accordions, carousels, tooltips, and so on. Bootstrap gives you ability to create flexible and responsive web layouts with much less efforts. Bootstrap was originally created by a designer and a developer at Twitter in mid-2010. Before being an open-sourced framework, Bootstrap was known as Twitter Blueprint. You can save a lot of time and effort with Bootstrap.

What You Can Do with Bootstrap

There are lot more things you can do with Bootstrap. You can easily create responsive websites.
You can quickly create multi-column layout with pre-defined classes. You can quickly create different types of form layouts. You can quickly create different variation of navigation bar. You can easily create components like accordions, modals, etc. without writing any JS code. You can easily create dynamic tabs to manage large amount of content. You can easily create tooltips and popovers

to show hint text. You can easily create carousel or image slider to showcase your content. You can quickly create different types of alert boxes.

Advantages of Using Bootstrap

- ❖ Save lots of time — You can save lots of time and efforts using the Bootstrap predefined design templates and classes and concentrate on other development work.
- ❖ Responsive features — Using Bootstrap you can easily create responsive websites that appear more appropriately on different devices and screen resolutions without any change in markup.
- ❖ Consistent design — All Bootstrap components share the same design templates and styles through a central library, so the design and layout of your web pages will be consistent.
- ❖ Easy to use — Bootstrap is very easy to use. Anybody with the basic working knowledge of HTML, CSS and JavaScript can start development with Bootstrap.
- ❖ Compatible with browsers — Bootstrap is created with modern web browsers in mind and it is compatible with all modern browsers such as Chrome, Firefox, Safari, Internet Explorer, etc.
- ❖ Open Source — And the best part is, it is completely free to download and use.

Creating Web Page with Bootstrap

- Now we're going to create a basic Bootstrap template by including the Bootstrap CSS and JS files via CDN.
- Bootstrap requires a third-party library Popper.js for some of its components like popovers and tooltips.
- You can either include it separately or simply include Bootstrap JS bundled with Popper.
- We recommend adding Bootstrap in your project via CDN (Content Delivery Network) because CDN offers performance benefit by reducing the loading time, since they are hosting the files on multiple servers spread across the globe so that when a user requests the file it will be served from the server nearest to them.

Step 1: Creating a Basic HTML file

Open up code editor and create a new HTML file.

Start with an empty window and type the following code and save it as "basic.html" on your desktop.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Basic HTML File</title>
</head>
<body>
   <h1>Hello, world!</h1>
</body>
</html>
```

Always include the viewport <meta> tag inside the <head> section of your document to enable touch zooming and ensure proper rendering on mobile devices.

Step 2: Making this HTML File a Bootstrap Template

In order to make this plain HTML file a Bootstrap template, just include the Bootstrap CSS and JS files using their CDN links.
Also, we should include JavaScript files at the bottom of the page, right before the closing </body> tag to improve the performance of your web pages.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Basic Bootstrap Template</title>
   <!-- Bootstrap CSS -->
   <link              href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
   <h1>Hello, world!</h1>
   <!-- Bootstrap JS Bundle with Popper -->
   <script       src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
</body>
</html>
```

After adding the Bootstrap's CSS and JS files to our HTML page, we can begin to develop any responsive site or application with the Bootstrap framework. The attributes integrity and cross origin have been added to CDN links to implement Sub resource Integrity (SRI). It is a security feature that enables you to mitigate the risk of attacks originating from compromised CDNs, by ensuring that the files your website fetches (from a CDN or anywhere) have been delivered without unexpected or malicious modifications. It works by allowing you to provide a cryptographic hash that a fetched file must match.

Step 3: Saving and Viewing the File
Now save the file on your desktop as "bootstrap-template.html". To open this file in a web browser, navigate to it, then right click on it, and select your favorite web browser. Alternatively, you can open your browser and drag this file to it.

Downloading the Bootstrap Files
Alternatively, you can also download the Bootstrap's CSS and JS files from their official website and include in your project. There are two versions available for download, compiled Bootstrap and Bootstrap source files. You can download Bootstrap 5 files from here.

- Compiled download contains compiled and minified version of CSS and JavaScript files for faster and easier web development.

- The compiled version also includes optional JavaScript dependencies such as Popper bundled with Bootstrap JS (bootstrap.bundle.*).
- Whereas, the source download contains original source files for all CSS and JavaScript, along with a local copy of the docs.
- Download and unzip the compiled Bootstrap.
- Now if you look inside the folders you'll find it contains the compiled CSS and JS files (bootstrap.*), as well as the compiled and minified CSS and JS (bootstrap.min.*).
- Use the bootstrap.min.css and bootstrap.bundle.min.js files.
- Using the minified version of CSS and JS files will improve the performance of your website and saves the precious bandwidth because of lesser HTTP request and download size.

Creating Containers with Bootstrap

- Containers are the most basic layout element in Bootstrap and are required when using the grid system.
- Containers are basically used to wrap content with some padding.
- They are also used to align the content horizontally center on the page in case of fixed width layout.

Bootstrap provides three different types containers:

❖ .container, which has a max-width at each responsive breakpoint.
❖ .container-fluid, which has 100% width at all breakpoints.
❖ .container-{breakpoint}, which has 100% width until the specified breakpoint.

The table below illustrates how each container's max-width changes across each breakpoint.
For example, when using the .container class the actual width of the container will be 100% if the viewport width is <576px, 540px
if the viewport width is ≥576px but <768px, 720px if the viewport width is ≥768px but <992px, 960px
if the viewport width is ≥992px but <1200px, 1140px if the viewport width is ≥1200px but <1400px, and 1320px if the viewport width is ≥1400px.
Similarly, when you use the .container-lg class the actual width of the container will be 100% until the viewport width is <992px, 960px if the viewport width is ≥992px but <1200px, 1140px if the viewport width ≥1200px but <1400px, and 1320px if the viewport width is ≥1400px.

Creating Responsive Fixed-width Containers

You can simply use the .container class to create a responsive, fixed-width container.
The width of the container will change at different breakpoints or screen sizes, as shown above.
Example

```
<div class="container">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
```

Creating Fluid Containers

You can use the .container-fluid class to create a full width container.
The width of the fluid container will always be 100% irrespective of the devices or screen sizes.
Example

```
<div class="container-fluid">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
```

Specify Responsive Breakpoints for Containers

Since Bootstrap v4.4, you can also create containers that is 100% wide until the specified breakpoint is reached, after which max-width for each of the higher breakpoints will be applied. For example, .container-xl will be 100% wide until the xl breakpoint is reached (i.e., viewport width ≥ 1200px), after which max-width for xl breakpoint is applied, which is 1140px.

Example

```
<div class="container-sm">100% wide until screen size less than 576px</div>
<div class="container-md">100% wide until screen size less than 768px</div>
<div class="container-lg">100% wide until screen size less than 992px</div>
<div class="container-xl">100% wide until screen size less than 1200px</div>
```

Adding Background and Borders to Containers

By default, container doesn't have any background-color or border. But if you need you can apply your own styles, or simply use the Bootstrap background-color and border utility classes to add background-color or border on them, as shown in the following example.

```
<!-- Container with dark background and white text color -->
<div class="container bg-dark text-white">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
<!-- Container with light background -->
<div class="container bg-light">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
<!-- Container with border -->
<div class="container border">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
```

Applying Paddings and Margins to Containers

By default, containers have padding of 12px on the left and right sides, and no padding on the top and bottom sides. To apply extra padding and margins you can use the spacing utility classes.

```
<!-- Container with border, extra paddings and margins -->
<div class="container border py-3 my-3">
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</div>
```

Bootstrap Grid System

The Bootstrap grid system is the fastest and easy way to create responsive website layout.

What is Bootstrap Grid System?

Bootstrap grid system provides an easy and powerful way to create responsive layouts of all shapes and sizes. It is built with flexbox with mobile-first approach. Also, it is fully responsive and uses twelve column system (12 columns available per row) and six default responsive tiers.

You can use the Bootstrap's predefined grid classes for quickly making the layouts for different types of devices like mobile phones, tablets, laptops, desktops, and so on. For example, you can use the .col-* classes to create grid columns for extra small devices like mobile phones in portrait mode, and the .col-sm-* classes for mobile phones in landscape mode. Similarly, you can use the .col-md-* classes to create grid columns for medium screen devices like tablets, the .col-lg-* classes for devices like small laptops, the .col-xl-* classes for laptops and desktops, and the .col-xxl-* classes for large desktop screens.

Creating Two Column Layouts

The following example will show you how to create two column layouts for medium, large and extra large devices like tables, laptops and desktops etc. However, on mobile phones (screen width less than 768px), the columns will automatically become horizontal (2 rows, 1 column).

```
<div class="container">
   <!--Row with two equal columns-->
   <div class="row">
      <div class="col-md-6">Column left</div>
      <div class="col-md-6">Column right</div>
   </div>

   <!--Row with two columns divided in 1:2 ratio-->
   <div class="row">
      <div class="col-md-4">Column left</div>
      <div class="col-md-8">Column right</div>
   </div>

   <!--Row with two columns divided in 1:3 ratio-->
   <div class="row">
      <div class="col-md-3">Column left</div>
      <div class="col-md-9">Column right</div>
   </div>
</div>
```

Since the Bootstrap grid system is based on 12 columns, therefore to keep the columns in a one line (i.e. side by side), the sum of the grid column numbers within a single row should not be greater than 12. If you go through the above example code carefully you will find the numbers of grid columns (i.e. col-md-*) add up to twelve (6+6, 4+8 and 3+9) for every row.

Creating Three Column Layouts

Similarly, you can create other layouts based on the above principle. For instance, the following example will typically create three column layouts for laptops and desktops screens. It also works in tablets in landscape mode if screen resolution is more than or equal to 992 pixels (e.g. Apple iPad). However, in portrait mode the grid columns will be horizontal as usual.

```
<div class="container">
<!--Row with three equal columns-->
    <div class="row">
        <div class="col-lg-4">Column left</div>
        <div class="col-lg-4">Column middle</div>
        <div class="col-lg-4">Column right</div>
    </div>
<!--Row with three columns divided in 1:4:1 ratio-->
    <div class="row">
        <div class="col-lg-2">Column left</div>
        <div class="col-lg-8">Column middle</div>
        <div class="col-lg-2">Column right</div>
    </div>
 <!--Row with three columns divided unevenly-->
    <div class="row">
        <div class="col-lg-3">Column left</div>
        <div class="col-lg-7">Column middle</div>
        <div class="col-lg-2">Column right</div>
    </div>
</div>
```
Bootstrap Auto-layout Columns
You can also create equal width columns for all devices (x-small, small, medium, large, x-large, and xx-large) through simply using the class .col, without specifying any column number. Let's try out the following example to understand how it exactly works:
```
<div class="container">
    <!--Row with two equal columns-->
    <div class="row">
        <div class="col">Column one</div>
        <div class="col">Column two</div>
    </div>
   <!--Row with three equal columns-->
    <div class="row">
        <div class="col">Column one</div>
        <div class="col">Column two</div>
        <div class="col">Column three</div>
    </div>
</div>
```
Additionally, you can also set the width of one column and let the sibling columns automatically resize around it equally. You may use the predefined grid classes or inline widths. If you try the following example you'll find columns in a row with class .col has equal width.

```
<div class="container">
    <!--Row with two equal columns-->
    <div class="row">
        <div class="col">Column one</div>
        <div class="col">Column two</div>
    </div>

    <!--Row with three columns divided in 1:2:1 ratio-->
    <div class="row">
        <div class="col">Column one</div>
        <div class="col-sm-6">Column two</div>
        <div class="col">Column three</div>
    </div>
</div>
```

Column Wrapping Behavior

Now we are going to create more flexible layouts that changes the column orientation based on the viewport size. The following example will create a three column layout on large devices like laptops and desktops, as well as on tablets (e.g. Apple iPad) in landscape mode, but on medium devices like tablets in portrait mode (768px ≤ screen width < 992px), it will change into a two column layout where the third column moves at the bottom of the first two columns.

```
<div class="container">
    <div class="row">
        <div class="col-md-4 col-lg-3">Column one</div>
        <div class="col-md-8 col-lg-6">Column two</div>
        <div class="col-md-12 col-lg-3">Column three</div>
    </div>
</div>
```

Bootstrap Fixed Layout

Creating Fixed Layout with Bootstrap

With Bootstrap you can still create web page layouts based on fixed number of pixels, however the container width vary depending on the viewport width and the layout is responsive too. The process of creating the fixed yet responsive layout basically starts with the .container class. After that you can create rows with the .row class to wrap the horizontal groups of columns. Rows must be placed within a .container for proper alignment and padding. Further columns can be created inside a row using the predefined grid classes such as .col, col-{xs|sm|md|lg|xl|xxl}-*, where * represent grid number and should be from 1 to 12.

The following example will create a fixed width responsive layout that is 720px pixels wide on medium devices like tablets (viewport ≥ 768px), whereas 960px wide on large devices like small laptops (viewport ≥ 992px), 1140px wide on extra large devices like desktops (viewport ≥ 1200px), and 1320px wide on extra-extra large devices like large desktops (viewport ≥ 1400px).

However, on small devices such as mobile phones (576px ≤ viewport < 768px) the layout will be 540px wide. But, on extra-small devices (viewport < 576px) the layout will cover 100% width. Also, columns will be stacked vertically and navbar will be collapsed in both cases.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a href="#" class="navbar-brand">Tutorial Republic</a>
        <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-
target="#navbarCollapse">
```

```
            <span class="navbar-toggler-icon"></span>
        </button>
       <div class="collapse navbar-collapse" id="navbarCollapse">
            <div class="navbar-nav">
                <a href="#" class="nav-item nav-link active">Home</a>
                <a href="#" class="nav-item nav-link">Services</a>
                <a href="#" class="nav-item nav-link">About</a>
                <a href="#" class="nav-item nav-link">Contact</a>
            </div>
            <div class="navbar-nav ms-auto">
                <a href="#" class="nav-item nav-link">Register</a>
                <a href="#" class="nav-item nav-link">Login</a>
            </div>
        </div>
    </div>
</nav>
```

Bootstrap Fluid Layout

Creating Fluid Layout with Bootstrap

In Bootstrap you can use the class .container-fluid to create fluid layouts to utilize the 100% width of the viewport across all devices (extra small, small, medium, large, extra large, and extra-extra large). The class .container-fluid simply applies the width: 100% instead of different width for different viewport sizes. However, the layout will still responsive and you can use the grid classes as usual.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a href="#" class="navbar-brand">Tutorial Republic</a>
        <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-
target="#navbarCollapse">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarCollapse">
<div class="navbar-nav">
                <a href="#" class="nav-item nav-link active">Home</a>
                <a href="#" class="nav-item nav-link">Services</a>
                <a href="#" class="nav-item nav-link">About</a>
                  <a href="#" class="nav-item nav-link">Contact</a>
            </div>
<div class="navbar-nav ms-auto">
                <a href="#" class="nav-item nav-link">Login</a>
                  <a href="#" class="nav-item nav-link">Register</a>
            </div>
        </div>
    </div>
</nav>
```

Bootstrap Nav: Tabs and Pills

Bootstrap Nav Components

Bootstrap provides an easy and quick way to create basic navigation as well as components like tabs and pills which are very flexible and elegant. All the Bootstrap's nav components, including tabs and pills, share the same base markup and styles through the base .nav class.

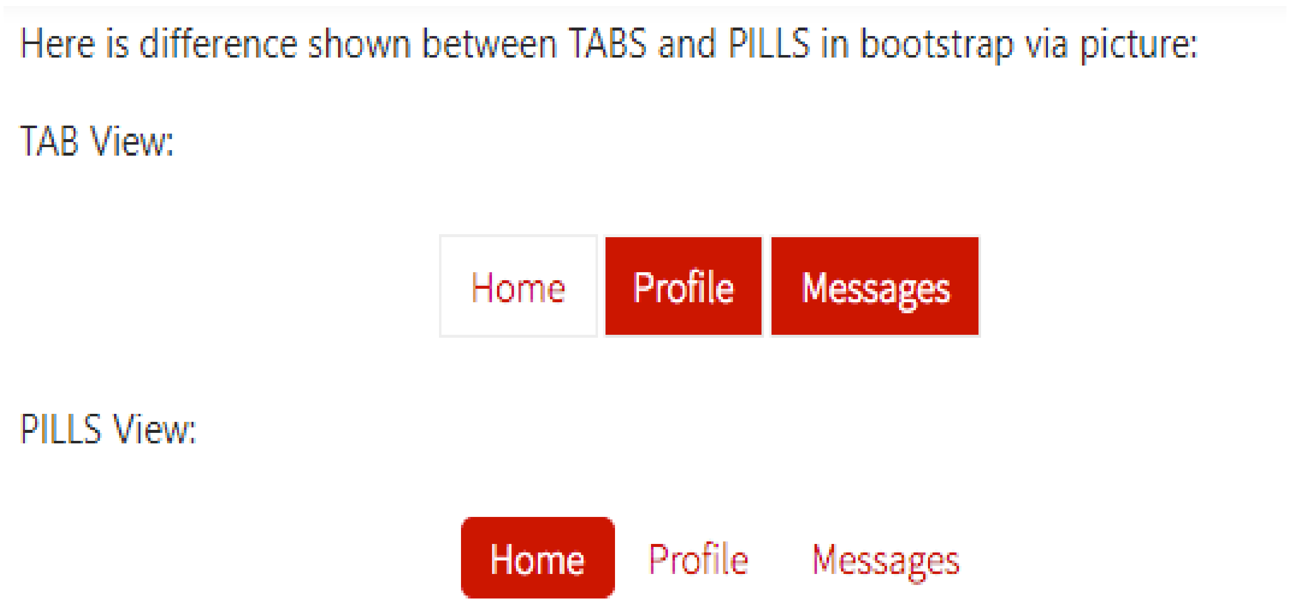

Here is difference shown between TABS and PILLS in bootstrap via picture:
TAB View:
Home
Profile
Messages
PILLS View:
Home
Profile
Messages

Fig 2.14 Tab and Pills

Creating Basic Nav with Bootstrap

You can use the Bootstrap .nav class to create a basic navigation menu, like this:

```
<nav class="nav">
   <a href="#" class="nav-item nav-link active">Home</a>
   <a href="#" class="nav-item nav-link">Profile</a>
   <a href="#" class="nav-item nav-link">Messages</a>
   <a href="#" class="nav-item nav-link disabled" tabindex="-1">Reports</a>
</nav>
```

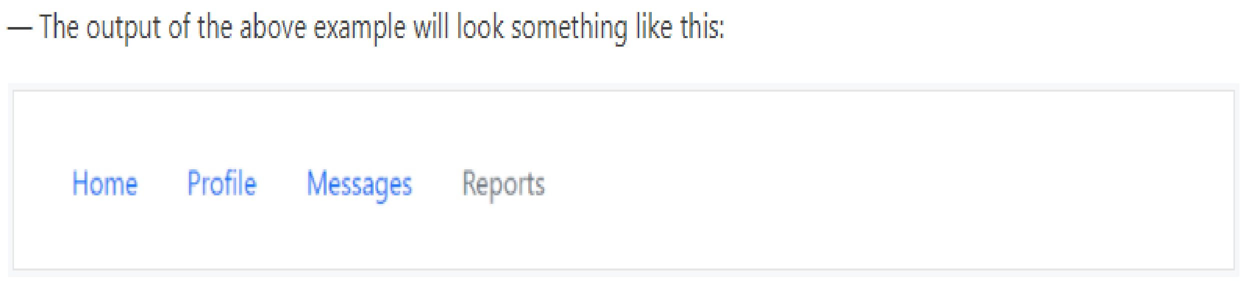

— The output of the above example will look something like this:
Home
Profile
Messages
Reports

Fig 2.15 Nav Item

Alignment of Nav Items

By default, navs are left-aligned, but you can easily align them to center or right using flexbox utilities.

The following example uses the class .justify-content-center to align nav items to center.

```
<nav class="nav justify-content-center">
   <a href="#" class="nav-item nav-link active">Home</a>
```

```
    <a href="#" class="nav-item nav-link">Profile</a>
    <a href="#" class="nav-item nav-link">Messages</a>
    <a href="#" class="nav-item nav-link disabled" tabindex="-1">Reports</a>
</nav>
```

— The output of the above example will look something like this:

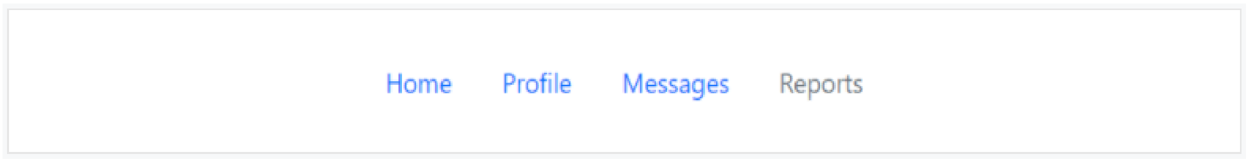


Fig 2.16 Nav Item Alignment

Creating the Basic Tabs

Simply, add the class .nav-tabs to the basic nav to generate a tabbed navigation, like this:

```
<nav class="nav nav-tabs">
    <a href="#" class="nav-item nav-link active">Home</a>
    <a href="#" class="nav-item nav-link">Profile</a>
    <a href="#" class="nav-item nav-link">Messages</a>
    <a href="#" class="nav-item nav-link disabled" tabindex="-1">Reports</a>
</nav>
```

— The output of the above example will look something like this:

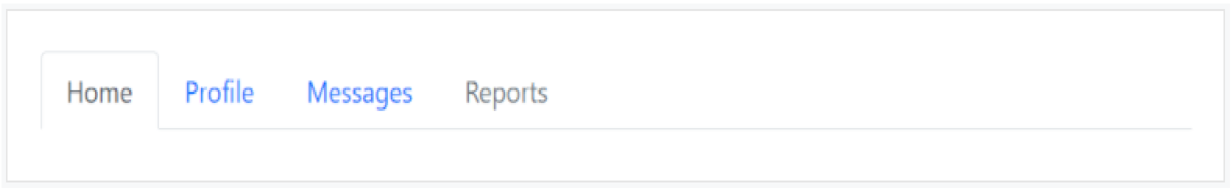


Fig 2.17 Basic Tab

Bootstrap Nav with Dropdown Menus

You can add dropdown menus to a link inside tabs and pills nav with a little extra markup. 66faThe four CSS classes .dropdown, .dropdown-toggle, .dropdown-menu and .dropdown-item are required in addition to the .nav, .nav-tabs or .nav-pills classes to create a simple dropdown menu inside tabs and pills nav without using any JavaScript code.

Creating Tabs with Dropdowns

The following example will show you how to add simple dropdown menu to a tab.

```
<nav class="nav nav-tabs">
    <a href="#" class="nav-item nav-link active">Home</a>
    <a href="#" class="nav-item nav-link">Profile</a>
    <div class="nav-item dropdown">
        <a href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">Messages</a>
        <div class="dropdown-menu">
            <a href="#" class="dropdown-item">Inbox</a>
            <a href="#" class="dropdown-item">Sent</a>
            <a href="#" class="dropdown-item">Drafts</a>
        </div>
    </div>
    <a href="#" class="nav-item nav-link disabled" tabindex="-1">Reports</a>
</nav>
```

— The output of the above example will look something like this:

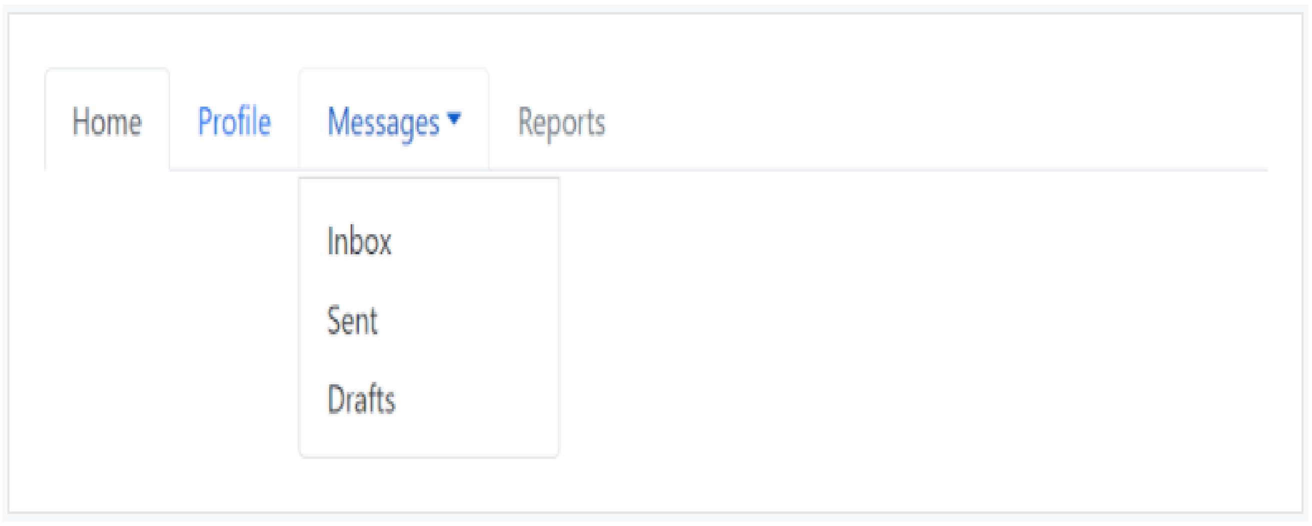


Fig 2.18 Basic Tab with Dropdown

Changing the Color Scheme of Navbars

You can also change the color scheme of the navbar by using the .navbar-light for the light background colors, or .navbar-dark for the dark background colors. Then, customize it with the background color utility classes, such as .bg-dark, .bg-primary, and so on. Alternatively, you can also apply the CSS background-color property on the .navbar element yourself to customize the navbar theme, as shown in the following example:

```
<nav class="navbar navbar-dark bg-dark">
    <!-- Navbar content -->
</nav>
<nav class="navbar navbar-dark bg-primary">
    <!-- Navbar content -->
</nav>
<nav class="navbar navbar-light" style="background-color: #ddeeff;">
    <!-- Navbar content -->
</nav>
```

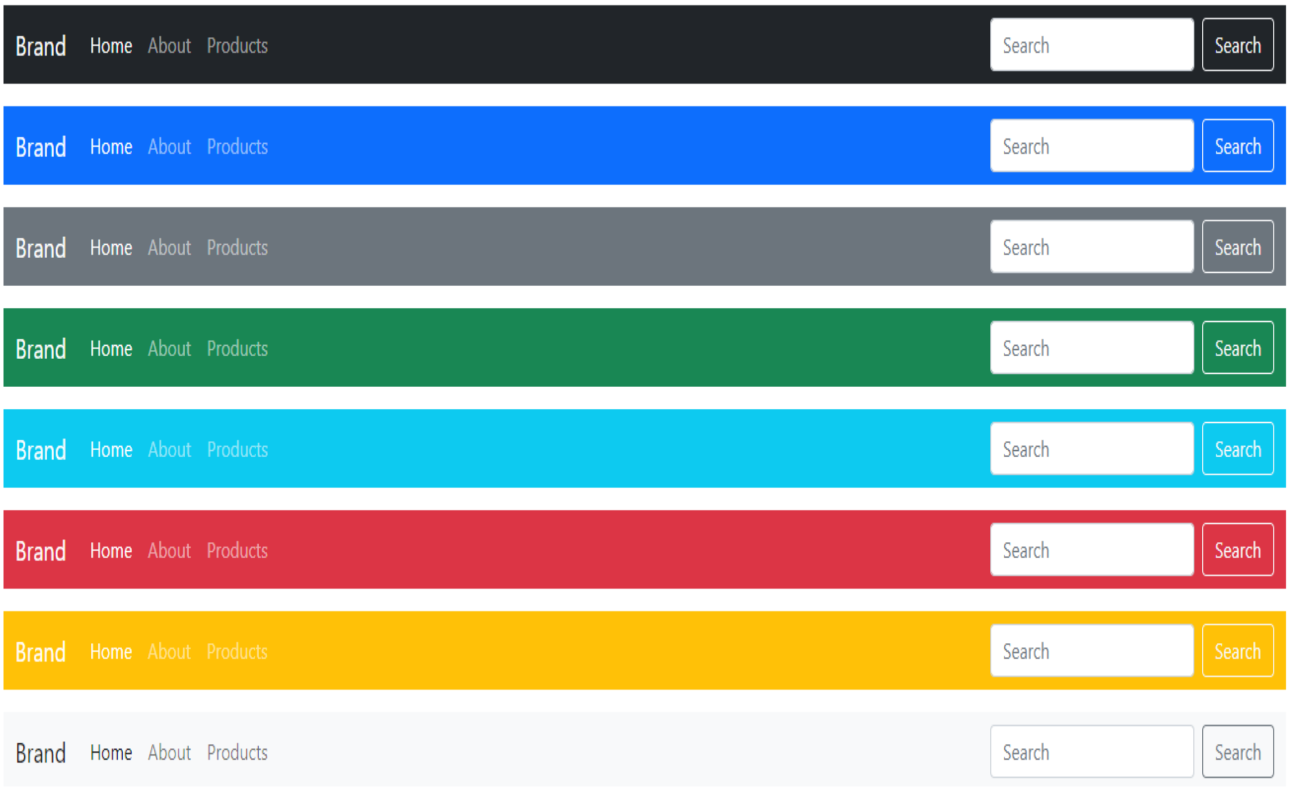


Fig 2.19 Color scheme of Navbar

Java Script Plugins

What is plugins?

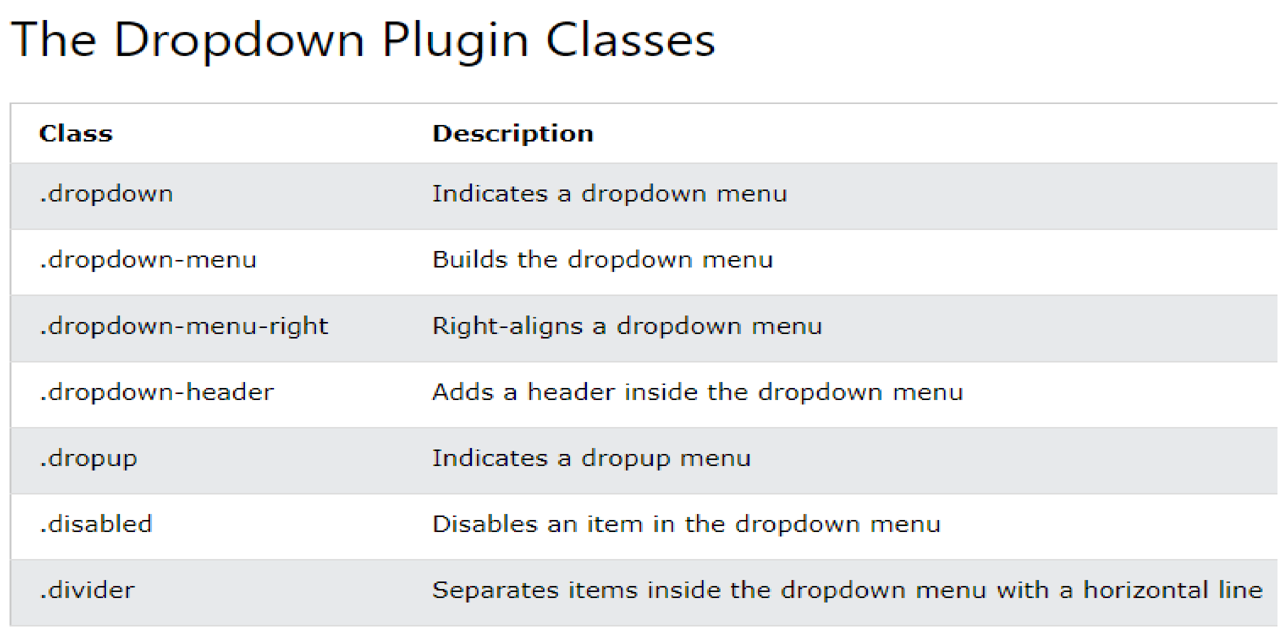
A plugin is a software add-on that is installed on a program, enhancing its capabilities. For example, if you wanted to watch a video on a website, you may need a plugin to do so. If the plugin is not installed, your browser will not understand how to play the video.

What is Javascript Plugin?

A plug-in is piece of code written in a standard JavaScript file. These files provide useful jQuery methods which can be used along with jQuery library methods. There are plenty of jQuery plug-in available which you can download from repository link at https://jquery.com/plugins.

JS Dropdown (dropdown.js)

A dropdown menu is a toggleable menu that allows the user to choose one value from a predefined list.



The Dropdown Plugin Classes

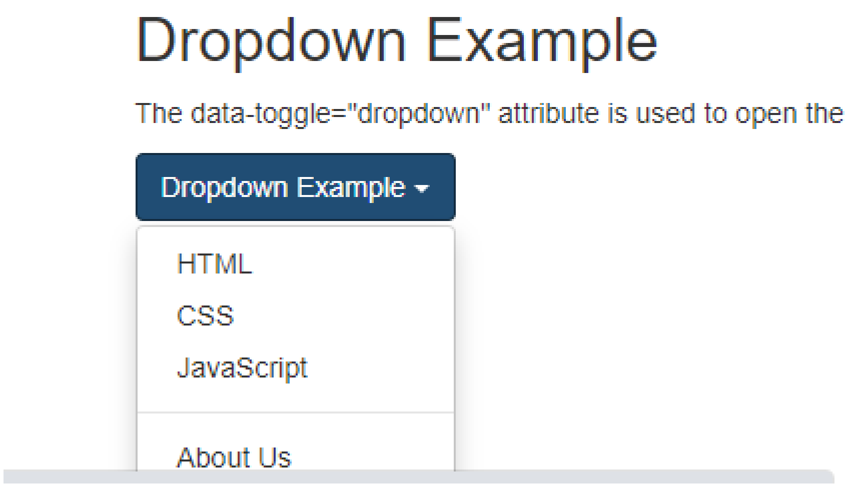
Class	Description
.dropdown	Indicates a dropdown menu
.dropdown-menu	Builds the dropdown menu
.dropdown-menu-right	Right-aligns a dropdown menu
.dropdown-header	Adds a header inside the dropdown menu
.dropup	Indicates a dropup menu
.disabled	Disables an item in the dropdown menu
.divider	Separates items inside the dropdown menu with a horizontal line

Via data-* Attributes

Add data-toggle="dropdown" to a link or a button to toggle a dropdown menu.

Example

<a href="#" class="dropdown-toggle" data-toggle="dropdown">Dropdown Example</a>



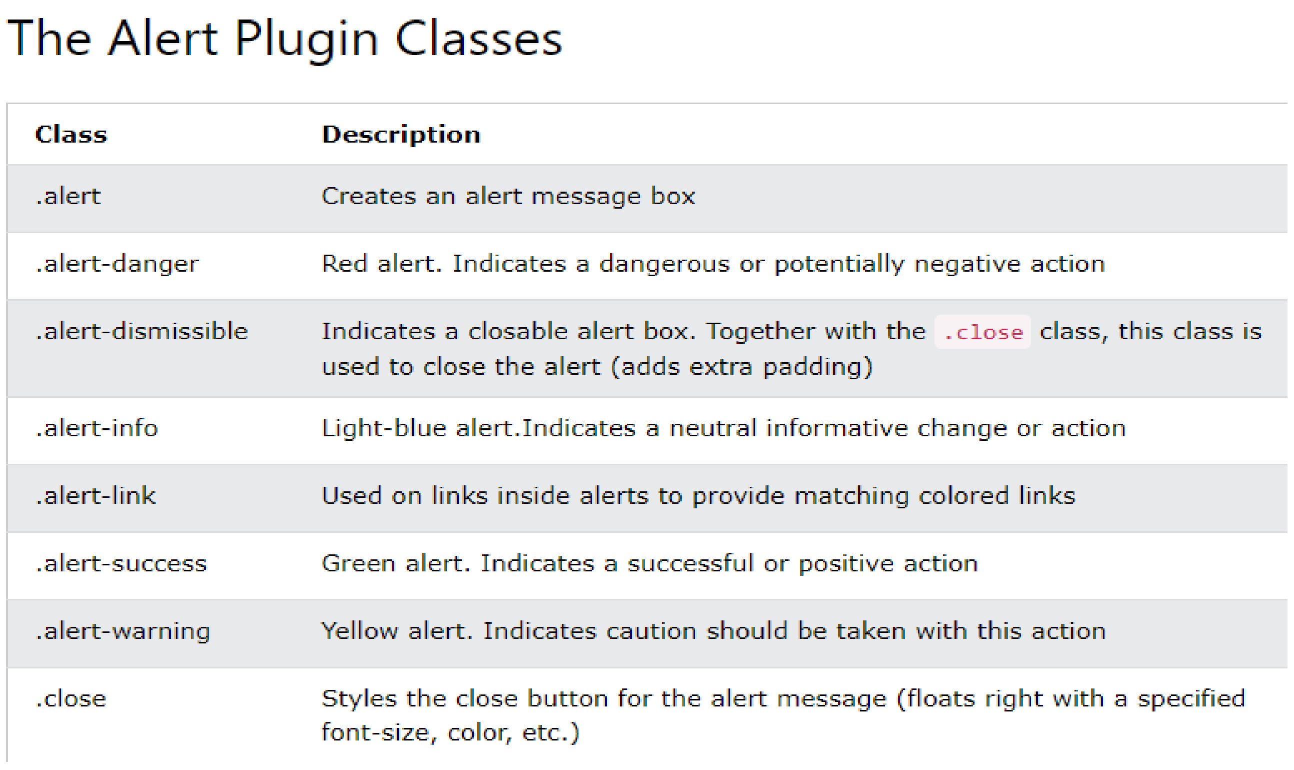
Dropdown Example

The data-toggle="dropdown" attribute is used to open the dropdown menu.

Fig 2.20 Javascript Plugins

JS Alert (alert.js)

The alert plugin include options and methods to close alert messages.



The Alert Plugin Classes

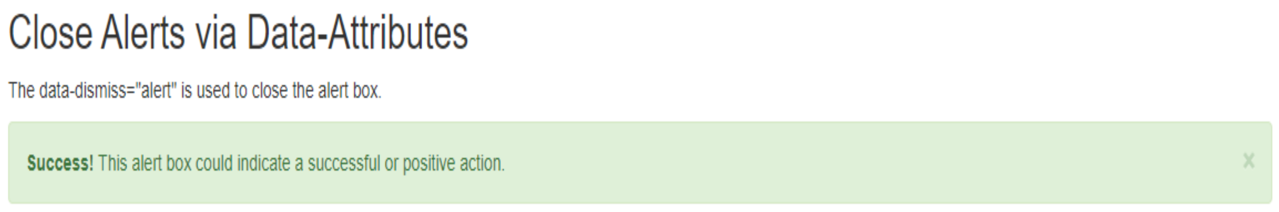
Class	Description
.alert	Creates an alert message box
.alert-danger	Red alert. Indicates a dangerous or potentially negative action
.alert-dismissible	Indicates a closable alert box. Together with the `.close` class, this class is used to close the alert (adds extra padding)
.alert-info	Light-blue alert.Indicates a neutral informative change or action
.alert-link	Used on links inside alerts to provide matching colored links
.alert-success	Green alert. Indicates a successful or positive action
.alert-warning	Yellow alert. Indicates caution should be taken with this action
.close	Styles the close button for the alert message (floats right with a specified font-size, color, etc.)

Close Alerts Via data-* Attributes

Add data-dismiss="alert" to a link or a button element to close the alert message.

Example

```
<a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
```

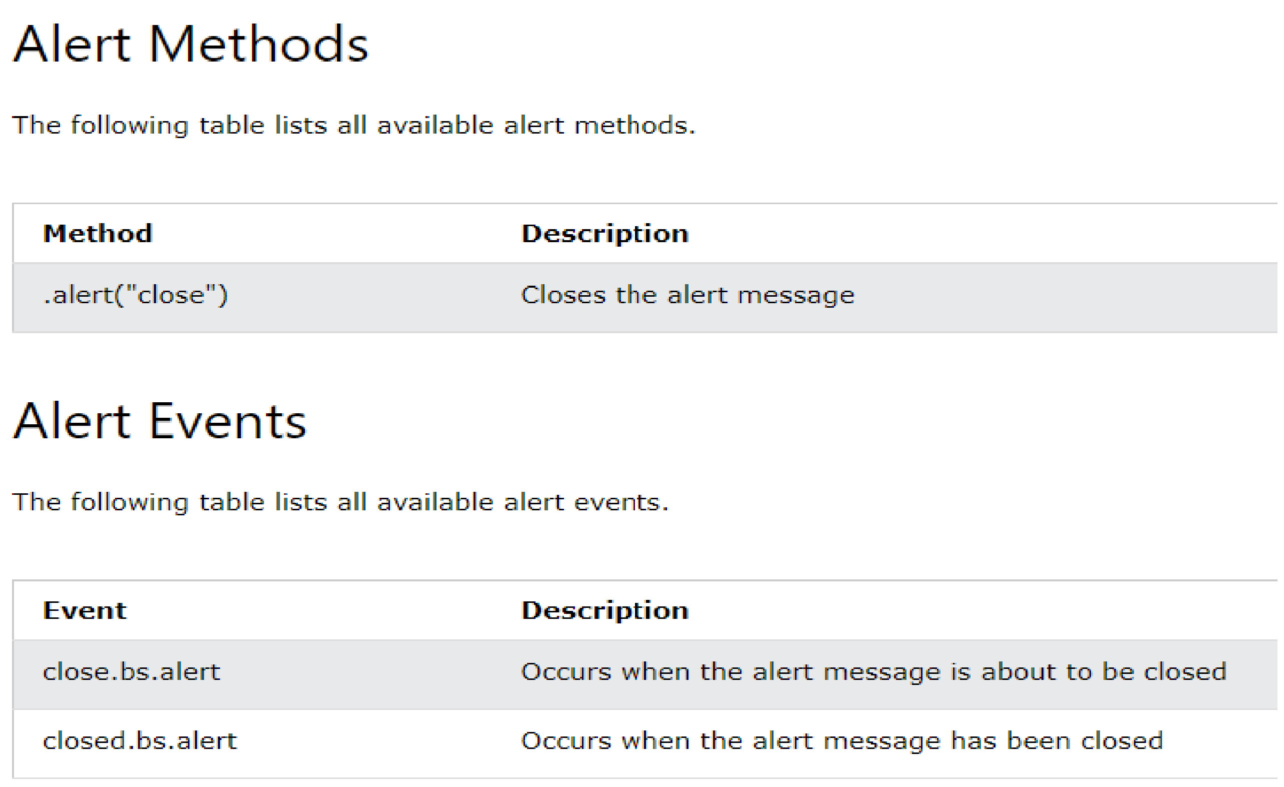


Close Alerts via Data-Attributes

The data-dismiss="alert" is used to close the alert box.

Success! This alert box could indicate a successful or positive action. x

Fig 2.21 Close Alerts



Alert Methods

The following table lists all available alert methods.

Method	Description
.alert("close")	Closes the alert message

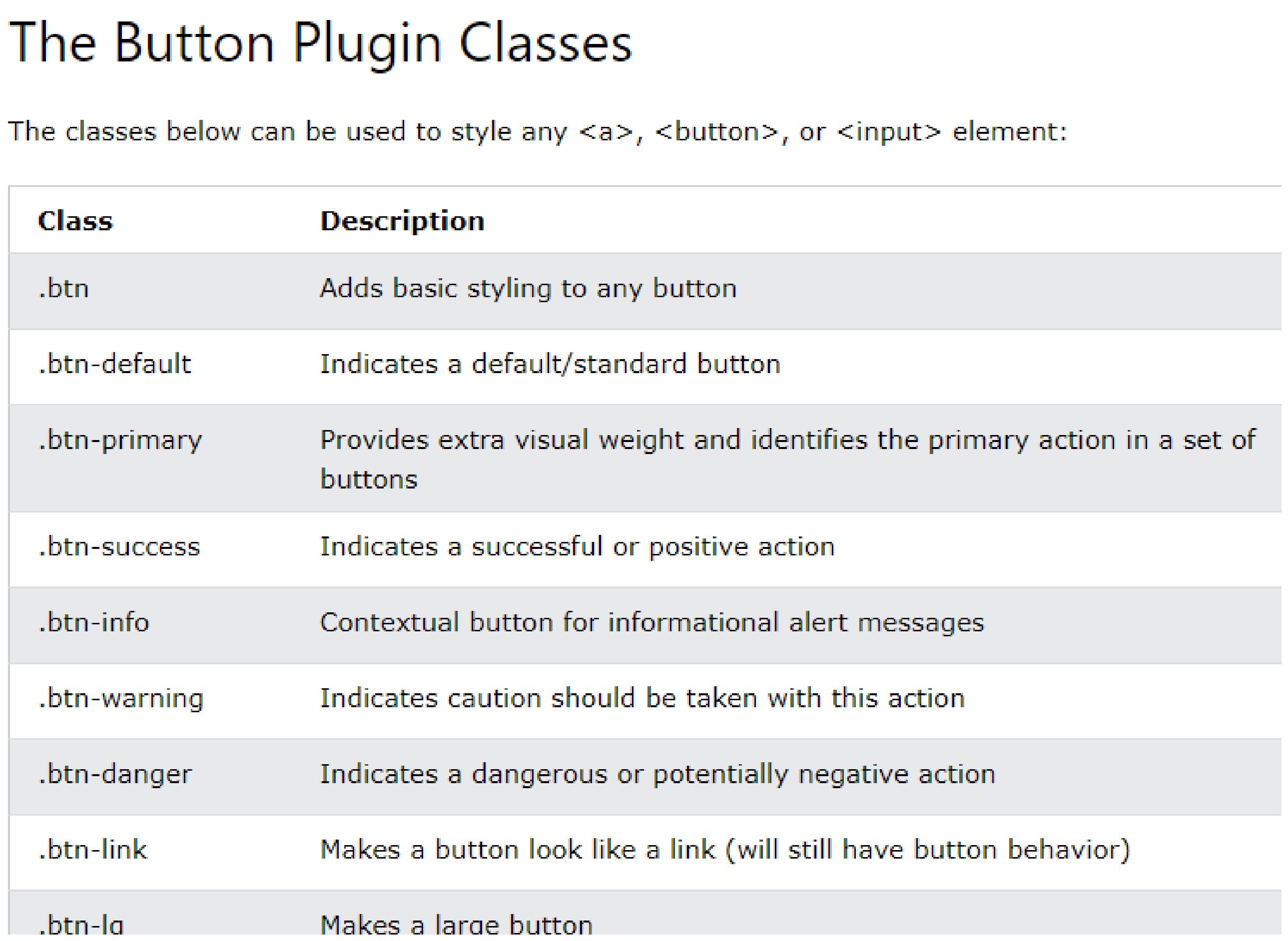
Alert Events

The following table lists all available alert events.

Event	Description
close.bs.alert	Occurs when the alert message is about to be closed
closed.bs.alert	Occurs when the alert message has been closed

JS Button (button.js)

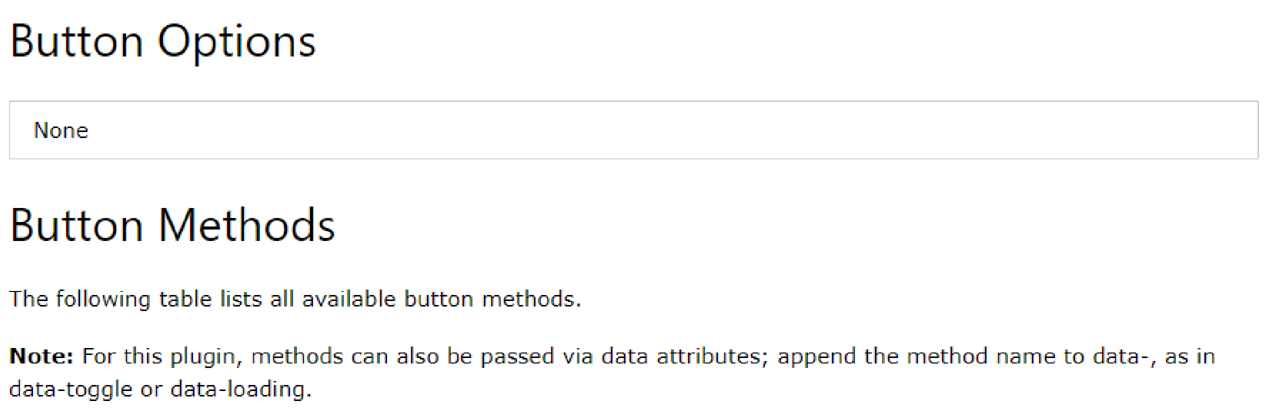
Use this plugin if you want to have more control over your buttons.



The Button Plugin Classes

The classes below can be used to style any <a>, <button>, or <input> element:

Class	Description
.btn	Adds basic styling to any button
.btn-default	Indicates a default/standard button
.btn-primary	Provides extra visual weight and identifies the primary action in a set of buttons
.btn-success	Indicates a successful or positive action
.btn-info	Contextual button for informational alert messages
.btn-warning	Indicates caution should be taken with this action
.btn-danger	Indicates a dangerous or potentially negative action
.btn-link	Makes a button look like a link (will still have button behavior)
.btn-lg	Makes a large button



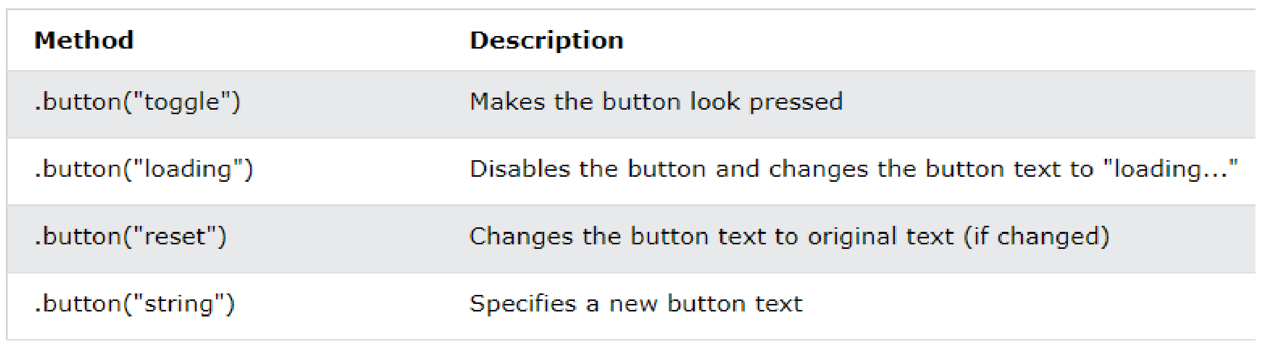
Button Options

None

Button Methods

The following table lists all available button methods.

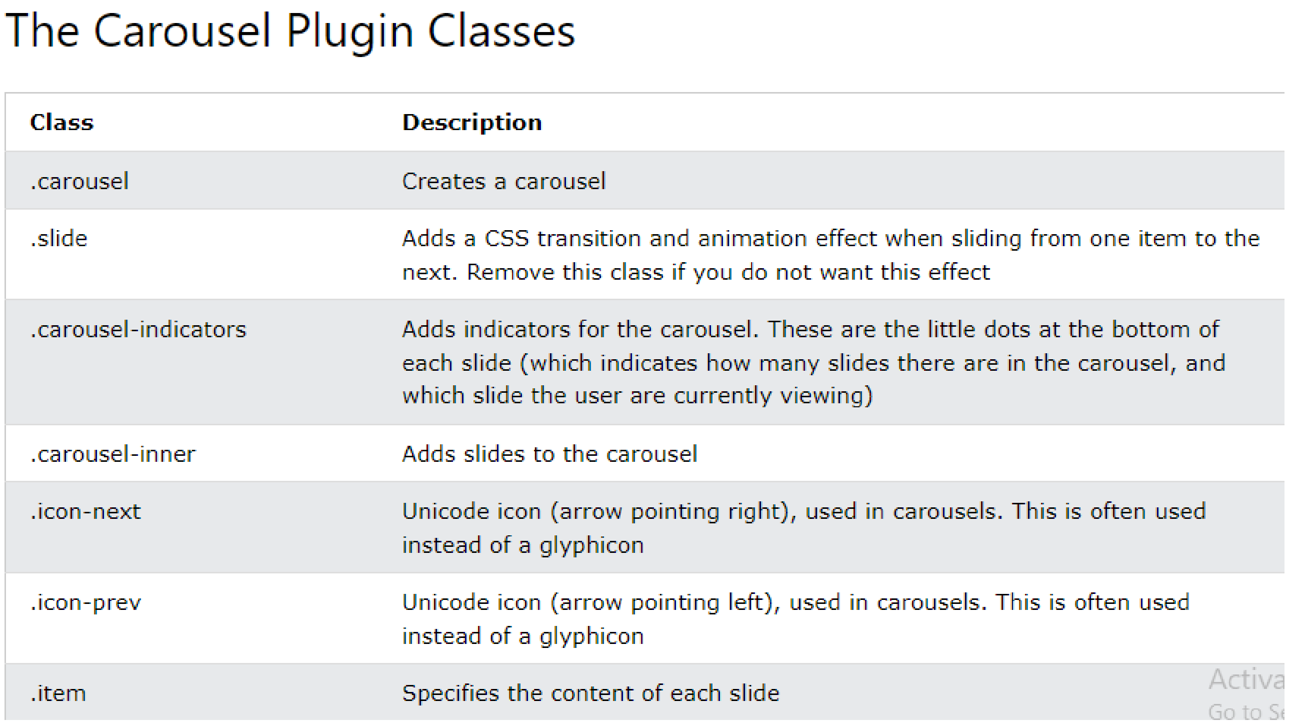
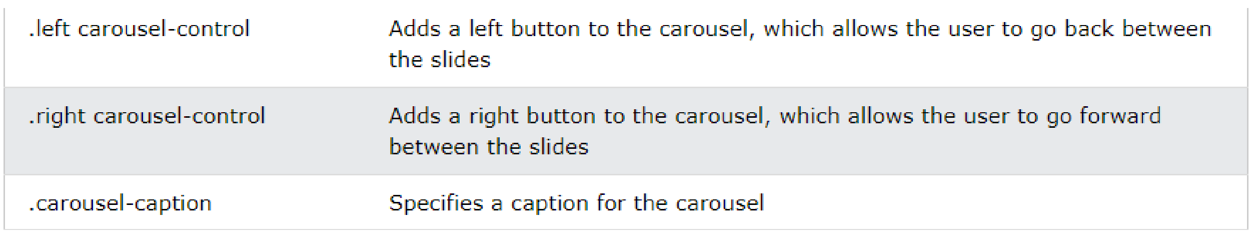
Note: For this plugin, methods can also be passed via data attributes; append the method name to data-, as in data-toggle or data-loading.



Method	Description
.button("toggle")	Makes the button look pressed
.button("loading")	Disables the button and changes the button text to "loading..."
.button("reset")	Changes the button text to original text (if changed)
.button("string")	Specifies a new button text

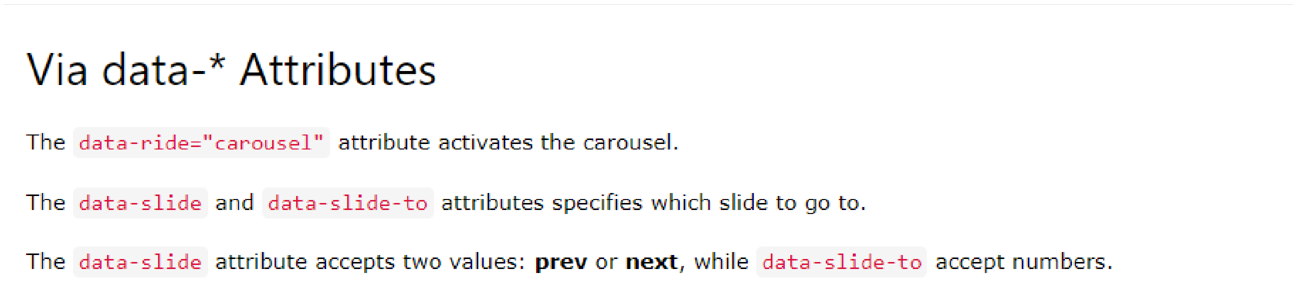
JS Carousel (carousel.js)

The Carousel plugin is a component for cycling through elements, like a carousel (slideshow).

The Carousel Plugin Classes

Class	Description
.carousel	Creates a carousel
.slide	Adds a CSS transition and animation effect when sliding from one item to the next. Remove this class if you do not want this effect
.carousel-indicators	Adds indicators for the carousel. These are the little dots at the bottom of each slide (which indicates how many slides there are in the carousel, and which slide the user are currently viewing)
.carousel-inner	Adds slides to the carousel
.icon-next	Unicode icon (arrow pointing right), used in carousels. This is often used instead of a glyphicon
.icon-prev	Unicode icon (arrow pointing left), used in carousels. This is often used instead of a glyphicon
.item	Specifies the content of each slide
.left carousel-control	Adds a left button to the carousel, which allows the user to go back between the slides
.right carousel-control	Adds a right button to the carousel, which allows the user to go forward between the slides
.carousel-caption	Specifies a caption for the carousel



Via data-* Attributes

The `data-ride="carousel"` attribute activates the carousel.

The `data-slide` and `data-slide-to` attributes specifies which slide to go to.

The `data-slide` attribute accepts two values: **prev** or **next**, while `data-slide-to` accept numbers.

<!-- Carousel -->

<div id="myCarousel" class="carousel slide" data-ride="carousel">

<!-- Carousel Indicators -->

<li data-target="#myCarousel" data-slide-to="1"></li>

<!-- Carousel Controls -->

<a class="left carousel-control" href="#myCarousel" data-slide="prev">

Via JavaScript

Enable manually with:

Example

```
// Activate Carousel
$("#myCarousel").carousel();
// Enable Carousel Indicators
$(".item").click(function(){
  $("#myCarousel").carousel(1);
});
// Enable Carousel Controls
$(".left").click(function(){
  $("#myCarousel").carousel("prev");
});
```

TEXT / REFERENCE BOOKS

1.Web link for Responsive Web Design - https://bradfrost.github.io/this-is-responsive/.

2.Web link for Responsive Web Design Introduction - W3Schoolshttps://www.w3schools.com › css › css_rwd_intro

Question Bank

Part-A			
Q.No	**Questions (2 Marks)**	**Competence**	**BT Level**
1.	What is Responsive Web Design	Knowledge	BTL1
2.	Write the syntax to set the viewport.	Knowledge	BTL1
3.	Illustrate on how to building a Responsive Grid-View.	Understand	BTL2
4.	Point out the use of media query in RWD.	Apply	BTL3
5.	Construct a code to Hide Elements With Media Queries.	Create	BTL6
6.	Construct the code to cover the background with an image in RWD.	Analyze	BTL4
7.	Name few CSS responsive framework that are used in the Industry for developing websites.	Knowledge	BTL1

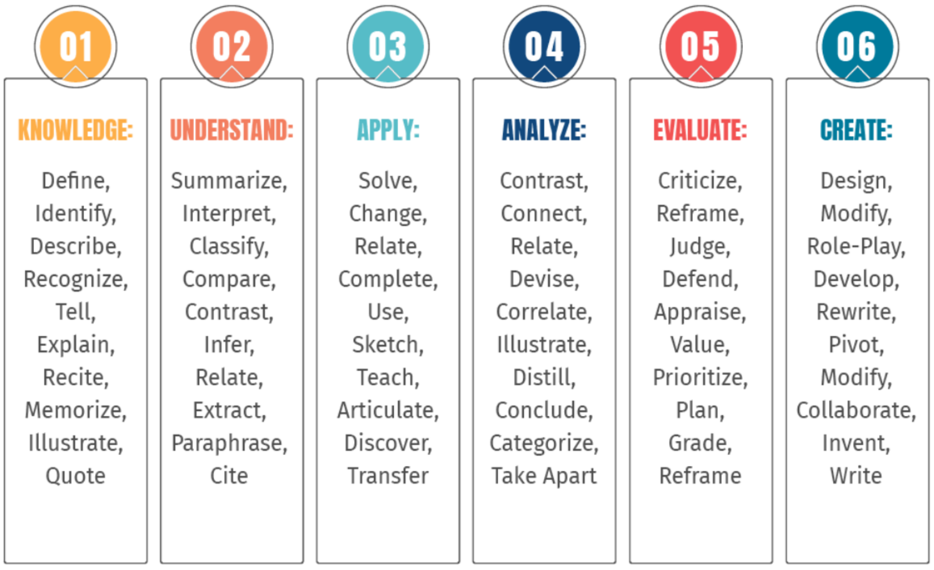
8.	Summarize the prerequisite web technologies that one must understand for learning Responsive Web Designin (RWD)	Understand	BTL2
9.	Generalize the best standards for using responsive design custom queries or Framework like Bootstrap	Create	BTL6
10.	Show the Meta viewport settings are different for Responsive Email?	Apply	BTL3
11.	Connect the future scope and challenges that you see in the Responsive Web Designing?	Analyze	BTL4
12.	Explain the main tags and attributes used in the below mentioned statement. <meta name="viewport" content="width=device-width, initial-scale=1.0">	Understand	BTL2
13.	Predict the major sign that your website is not responsive or having issues?	Evaluate	BTL5
14.	Write the syntax to make the webpage responsive using bootstrap framework.	Create	BTL6
15.	Define JS plugins.	Knowledge	BTL1

Part-B

Q.No	**Questions (16 Marks)**	**Competence**	**BT Level**
1.	Construct a program to implement RWD gridview.	Create	BTL6
2.	Develop a program to design a webpage using media queries with different properties.	Create	BTL6
3.	Discuss the various ways to add a background image in a webpage using the properties.	Understand	BTL2
4.	List out the advantages of Bootstrap and explain the concept in detail with an example program.	Knowledge	BTL1
5.	Design a webpage with offcanvas sidebar and explain it briefly.	Create	BTL6
6.	Explain the bootstrap components in detail with example.	Analyze	BTL4
7.	Summarize the ways to create the customized RWD.	Evaluate	BTL5

8.	Explain the JS plugins with example programs in detail.	Analyze	BTL4
9.	Design a responsive webpage with the properties of grid container.	Create	BTL6
10.	Explain the various features of bootstrap and background properties in detail.	Understand	BTL2

Bloom's Taxonomy



01	02	03	04	05	06
KNOWLEDGE:	UNDERSTAND:	APPLY:	ANALYZE:	EVALUATE:	CREATE:
Define, Identify, Describe, Recognize, Tell, Explain, Recite, Memorize, Illustrate, Quote	Summarize, Interpret, Classify, Compare, Contrast, Infer, Relate, Extract, Paraphrase, Cite	Solve, Change, Relate, Complete, Use, Sketch, Teach, Articulate, Discover, Transfer	Contrast, Connect, Relate, Devise, Correlate, Illustrate, Distill, Conclude, Categorize, Take Apart	Criticize, Reframe, Judge, Defend, Appraise, Value, Prioritize, Plan, Grade, Reframe	Design, Modify, Role-Play, Develop, Rewrite, Pivot, Modify, Collaborate, Invent, Write

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

UNIT – III -INTRODUCTION TO JAVASCRIPT AND JQUERY – SITA3004

III Introdiction to Java Script and JQuerry

Introduction - Core features - Data types and Variables - Operators, Expressions and Statements - Functions & Scope - Objects - Array, Date and Math related Objects - Document Object Model - Event Handling – Browser Object Model - Windows and Documents - Form handling and validations. Object-Oriented Techniques in JavaScript - Classes – Constructors and Prototyping (Sub classes and Super classes) – JSON – Introduction to AJAX. Introduction – jQuery Selectors – jQuery HTML - Animations – Effects – Event Handling – DOM – jQuery DOM Traversing, DOM Manipulation – jQuery AJAX

What is JavaScript?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers. The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser. It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
The JavaScript client-side mechanism provides many advantages over traditional CGI server- side scripts. For example, you might use JavaScript to check if the user has entered a valid e- mail address in a form field. The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server. JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are:

Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server. Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something. Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard. Richer interfaces: You can use JavaScript to include such items as dragand-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason. JavaScript cannot be used for networking applications because there is no such support available. JavaScript doesn't have any multithreading or multiprocessor capabilities. Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here:

•Microsoft FrontPage: Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.

•Macromedia Dreamweaver MX: Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

•Macromedia HomeSite 5: HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

SYNTAX

•JavaScript can be implemented using JavaScript statements that are placed within the

<script>….</script>HTML tags in a web page.

•You can place the <script> tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the <head> tags.

•The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>

JavaScript

code
```

```
</script>
```

The script tag takes two important attributes:

•Language: This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

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

So your JavaScript syntax will look as follows.

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

JavaScript code

</script>
```

First JavaScriptCode

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-- >". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function document.write which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>

<body>

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

<!--

document.write ("Hello World!")

//-->

</script>

</body>

</html>
```

This code will produce the following result: **Hello World!**

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

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

<!--

var1 = 10

var2 = 20

//-->

</script
```

But when formatted in a single line as follows, you must use semicolons:

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

<!--

var1 = 10; var2 = 20;

//-->

</script>
```

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters. So the identifiers Time and TIME will convey different meanings in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments. Thus:

•Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.

•Any text between the characters /* and */ is treated as a comment. This may span multiple lines.

•JavaScript also recognizes the HTML comment opening sequence is not recognized by JavaScript so it should be written as //-->.

•The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

The following example shows how to use comments in JavaScript

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

<!--

// This is a comment. It is similar to comments in C++

/*

*This is a multiline comment in JavaScript

*It is very similar to comments in C Programming

*/

//-->

</script
```

PLACEMENT

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:

Script in <head>...</head> section.

Script in <body>...</body> section.

Script in <body>...</body> and <head>...</head> sections.

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

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

JavaScript in <head>...</head> Section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows.

```
html>

<head>

<script type="text/javascript">

<!--

function sayHello() { alert("Hello World")

}

//-->

</script>
```

```
</head>

<body>

Click here for the result

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>
```

JavaScript in <body>...</body> Section

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

```
<html>

<head>

</head>

<body>

<script type="text/javascript">

<!--

document.write("Hello World")

//-->

</script>

<p>This is web page body </p>

</body>

</html>
```

```
Hello World

This is web page body
```

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as Follows.

```
<html>

<head>

<script type="text/javascript">

<!--
```

```
function sayHello()
{
alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Click here for the result Say Hello

JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

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

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

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello

function in your HTML file after including the filename.js file.

```
function sayHello()
{
alert("Hello World")
}
```

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers, e.g., 123, 120.50 etc.
- Strings of text, e.g. "This text string" etc.
- Boolean, e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object. We will cover objects in detail in a separate chapter.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--
var money; var name;
```

```
//-->
</script>
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
<!--
var name = "Ali"; var money; money = 2000.50;
//-->
</script>
```

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- Local Variables: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
 <!--
var myVar = "global"; // Declare a global variable function checkscope( ) {
var myVar = "local"; // Declare a local variable document.write(myVar);
}
//-->
</script>'
```

Result: Local

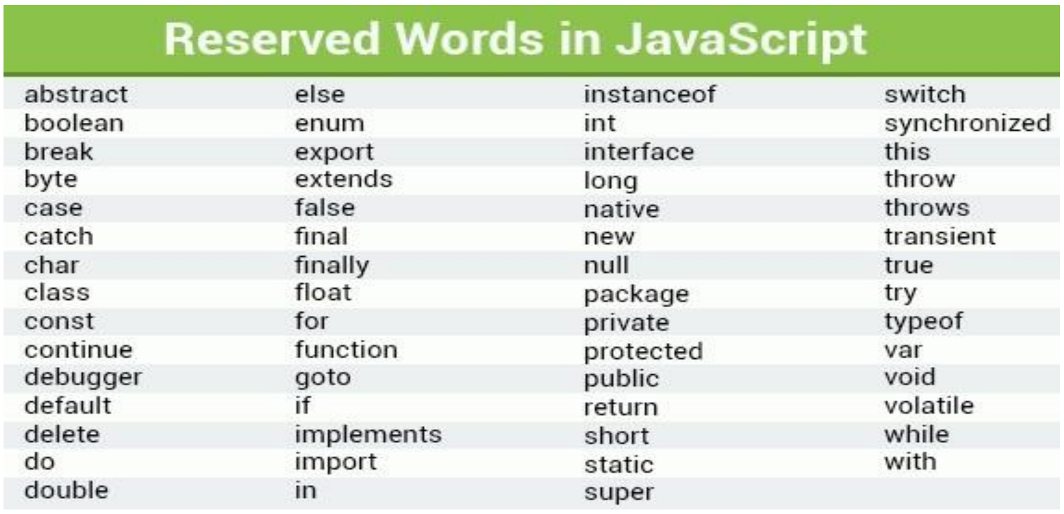
JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.



Reserved Words in JavaScript

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Fig 3.1 Reserved Words

OPERATORS

What is an Operator?

Let us take a simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 33; var b = 10;

var c = "Test";

var linebreak = "<br />"; document.write("a + b = "); result = a + b; document.write(result);
document.write(linebreak); document.write("a - b = "); result = a - b; document.write(result);
document.write(linebreak); document.write("a / b = "); result = a / b; document.write(result);
document.write(linebreak); document.write("a % b = ");

result = a % b; document.write(result); document.write(linebreak); document.write("a + b + c = ");
result = a + b + c; document.write(result); document.write(linebreak);

a = a++; document.write("a++ = "); result = a++; document.write(result);
document.write(linebreak); b = b--;

document.write("b-- = "); result = b--; document.write(result); document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and then try...</p>

</body>

</html>
```

Output

```
a + b = 43 a - b = 23 a / b = 3.3 a % b = 3

a + b + c = 43Test a++ = 33

b-- = 10
```

Comparison Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 10; var b = 20;

var linebreak = "<br />"; document.write("(a == b) => "); result = (a == b); document.write(result);
document.write(linebreak); document.write("(a < b) => "); result = (a < b); document.write(result);
document.write(linebreak); document.write("(a > b) => "); result = (a > b); document.write(result);
document.write(linebreak); document.write("(a != b) => "); result = (a != b);
document.write(result); document.write(linebreak); document.write("(a >= b) => "); result = (a >=
b); document.write(result); document.write(linebreak); document.write("(a <= b) => ");

 result = (a <= b); document.write(result); document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>
```

Output

```
(a == b) => false (a < b) => true (a > b) => false (a != b) => true (a >= b) => false (a <= b) => true
```

Logical Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = true; var b = false;

var linebreak = "<br />"; document.write("(a && b) => "); result = (a && b);
document.write(result); document.write(linebreak); document.write("(a || b) => ");

 result = (a || b); document.write(result); document.write(linebreak); document.write("!(a && b) =>
"); result = (!(a && b)); document.write(result); document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>
```

```
</body>

</html>
```

Output

```
(a && b) => false (a || b) => true

!(a && b) => true
```

Bitwise Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 2; // Bit presentation 10 var b = 3; // Bit presentation 11 var linebreak = "<br />";
document.write("(a & b) => "); result = (a & b); document.write(result); document.write(linebreak);
document.write("(a | b) => ");

 result = (a | b); document.write(result); document.write(linebreak); document.write("(a ^ b) => ");
result = (a ^ b); document.write(result); document.write(linebreak); document.write("(~b) => ");
result = (~b); document.write(result); document.write(linebreak); document.write("(a << b) => ");
result = (a << b); document.write(result); document.write(linebreak); document.write("(a >> b) =>
"); result = (a >> b); document.write(result); document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>
```

Output

```
(a & b) => 2 (a | b) => 3 (a ^ b) => 1 (~b) => -4

 (a << b) => 16 (a >> b) => 0
```

Assignment Operators

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 33; var b = 10;
var linebreak = "<br />"; document.write("Value of a => (a = b) => "); result = (a = b);

document.write(result); document.write(linebreak); document.write("Value of a => (a += b) => ");
result = (a += b);

document.write(result); document.write(linebreak); document.write("Value of a => (a -= b) => ");
result = (a -= b);

document.write(result); document.write(linebreak); document.write("Value of a => (a *= b) => ");
result = (a *= b);

document.write(result); document.write(linebreak); document.write("Value of a => (a /= b) => ");
result = (a /= b);

document.write(result); document.write(linebreak);

document.write("Value of a => (a %= b) => "); result = (a %= b);

document.write(result); document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>
```

Output

```
Value of a => (a = b) => 10 Value of a => (a += b) => 20 Value of a => (a -= b) => 10 Value of a
=> (a *= b) => 100 Value of a => (a /= b) => 10 Value of a => (a %= b) => 0
```

Miscellaneous Operators

conditional operator (? :) and the typeof operator

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

? : (Conditional)

If Condition is true? Then value X : Otherwise value Y

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 10;
```

```
 var b = 20;

var linebreak = "<br />";

document.write ("((a > b) ? 100 : 200) => "); result = (a > b) ? 100 : 200; document.write(result);
document.write(linebreak);

document.write ("((a < b) ? 100 : 200) => "); result = (a < b) ? 100 : 200; document.write(result);
document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>
```

Output

```
((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100
```

typeof Operator

The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand. The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Example

```
<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = "String";

var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric"); document.write("Result => ");

document.write(result); document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric"); document.write("Result => ");

document.write(result); document.write(linebreak);

//-->
```

```
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

```
Result => B is String Result => A is Numeric
```

CONTROL STATEMENTS

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

JavaScript supports the following forms of if..else statement:

- if statement
- if...else statement
- if...else if... statement

if Statement

The 'if' statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows:

```
if (expression)
{
Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

```
<html>
<body>
<script type="text/javascript">
<!--
var age = 20; if( age > 18 ){
document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

if...else Statement

The ‘if...else’ statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression){
Statement(s) to be executed if expression is true
}else{
Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

```
 <html>
<body>
<script type="text/javascript">
<!--
var age = 15; if( age > 18 ){
document.write("<b>Qualifies for driving</b>");
}else{
document.write("<b>Does not qualify for driving</b>");
}//-->
```

```
</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

Output

```
Does not qualify for driving

Set the variable to different value and then try...
```

if...else if... Statement

The ‘if...else if...’ statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

Syntax

```
if (expression 1){

Statement(s) to be executed if expression 1 is true

}else if (expression 2){

Statement(s) to be executed if expression 2 is true

}else if (expression 3){

Statement(s) to be executed if expression 3 is true

}else{

Statement(s) to be executed if no expression is true

}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

```
<html>

<body>

<script type="text/javascript">

<!--

var book = "maths";

if( book == "history" ){ document.write("<b>History Book</b>");

}else if( book == "maths" ){ document.write("<b>Maths Book</b>");
}else if( book == "economics" ){ document.write("<b>Economics Book</b>");
}else{
document.write("<b>Unknown Book</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html> Output Maths Book
```

Set the variable to different value and then try...

SWITCH-CASE

Syntax

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression)
{
case condition 1: statement(s) break;
case condition 2: statement(s) break;
...
case condition n: statement(s) break;
default: statement(s)
}
```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Example

```
<html>
<body>
<script type="text/javascript">
<!--
var grade='A';
```

```
document.write("Entering switch block<br />"); 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>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

Output

```
Entering switch block Good job

Exiting switch block

Set the variable to different value and then try...
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>

<body>

<script type="text/javascript">

<!--

var grade='A';

document.write("Entering switch block<br />");

switch (grade)

{

case 'A': document.write("Good job<br />"); case 'B': document.write("Pretty good<br />"); case 'C':
document.write("Passed<br />");
case 'D': document.write("Not so good<br />"); case 'F': document.write("Failed<br />");

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

Output

```
Entering switch block Good job

Pretty good Passed

Not so good Failed Unknown grade

Exiting switch block

Set the variable to different value and then try...
```

WHILE LOOP

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines. JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop

The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Syntax

```
while (expression){

Statement(s) to be executed if expression is true

}
```

Example

```
<html>

<body>

<script type="text/javascript">

<!--
```

```
var count = 0; document.write("Starting Loop "); while (count < 10){

document.write("Current Count : " + count + "<br />"); count++;

}

document.write("Loop stopped!");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

Output

```
Starting Loop Current Count : 0 Current Count : 1

Current Count : 2 Current Count : 3 Current Count : 4 Current Count : 5 Current Count : 6

 Current Count : 7 Current Count : 8 Current Count : 9 Loop stopped!

Set the variable to different value and then try...
```

The do...while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do

{

Statement(s) to be executed;

 } while (expression);
```

Example

```
<html>

<body>

<script type="text/javascript">

<!--

var count = 0;

document.write("Starting Loop" + "<br />"); do{

document.write("Current Count : " + count + "<br />"); count++;

}while (count < 5); document.write ("Loop stopped!");
```

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html> Output Starting Loop

Current Count : 0 Current Count : 1 Current Count : 2 Current Count : 3 Current Count : 4 Loop Stopped!

Set the variable to different value and then try...

FOR LOOP

The ‘for’ loop is the most compact form of looping. It includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter. You can put all the three parts in a single line separated by semicolons.

Syntax

for (initialization; test condition; iteration statement)

{

Statement(s) to be executed if test condition is true

}

Example

<html>

<body>

<script type="text/javascript">

<!--

var count;

document.write("Starting Loop" + "<br />"); for(count = 0; count < 10; count++){ document.write("Current Count : " + count); document.write("<br />");

}

document.write("Loop stopped!");

//-->

```
</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

Output

```
Starting Loop Current Count : 0 Current Count : 1 Current Count : 2 Current Count : 3 Current Count
: 4 Current Count : 5 Current Count : 6 Current Count : 7 Current Count : 8 Current Count : 9 Loop
stopped!

Set the variable to different value and then try...
```

FOR-IN LOOP

Syntax

```
for (variablename in object){ statement or block to execute

}
```

Example

```
<html>

<body>

<script type="text/javascript">

<!--

var aProperty;

document.write("Navigator Object Properties<br /> "); for (aProperty in navigator)

{

document.write(aProperty); document.write("<br />");

}

document.write ("Exiting from the loop!");

//-->

</script>

<p>Set the variable to different object and then try...</p>

</body>

</html>
```

Output

```
Navigator Object Properties
```

```
Exiting from the loop!

Set the variable to different object and then try.
```

LOOP CONTROL

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the look. To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

Example

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> "); while (x < 20)
{
if (x == 5){
break; // breaks out of loop completely
}
x = x + 1;
document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop 2

3
```

```
4
5
```

Exiting the loop!

Set the variable to different value and then try...

The continue Statement

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> "); while (x < 10)
{
x = x + 1; if (x == 5){
continue; // skill rest of the loop body
 }
document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop 2
3
```

```
4
6
7
8
9
10
Exiting the loop!
```

FUNCTIONS

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write().

JavaScript allows us to write our own functions as well.

Syntax

```
<script type="text/javascript">
<!--
 function functionname(parameter-list)
{
statements
}
//-->
</script>
```

Example

```
<script type="text/javascript">
<!--
function sayHello()
{
alert("Hello there");
}
//-->
```

```
</script>
<html>
<head>
<script type="text/javascript"> function sayHello()
{
document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
 </form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

Function Parameters

```
<html>
<head>
<script type="text/javascript"> function sayHello(name, age)
{
document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
```

```
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Nested Functions

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) { function square(x) { return x*x; }
return Math.sqrt(square(a) + square(b));
}
function secondFunction(){ var result;
result = hypotenuse(1,2); document.write ( result );
}
//-->
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Javascript Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

- Encapsulation − the capability to store related information, whether data or methods, together in an object.

- Aggregation − the capability to store one object inside another object.
- Inheritance − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- Polymorphism − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

objectName.objectProperty = propertyValue;

For example − The following code gets the document title using the "title" property of the document object.

var str = document.title;

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the this keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example − Following is a simple example to show how to use the write() method of document object to write any content on the document.

document.write("This is test");

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called Object. The new Operator, the new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method. In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable. The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

Example

```
<html>

<head>

<title>User-defined objects</title>

<script type = "text/javascript">

var book = new Object(); // Create the object book.subject = "Perl";        // Assign properties to
the object book.author = "Mohtashim";

</script>

</head>

<body>

<script type = "text/javascript">

document.write("Book name is : " + book.subject + "<br>"); document.write("Book author is : " +
book.author + "<br>");

</script>

</body>

</html>
```

Output

```
Book name is : Perl

Book author is : Mohtashim
```

JavaScript - The Number Object

The Number object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about Number objects because the browser automatically converts number literals to instances of the number class.

Syntax

The syntax for creating a number object is as follows − var val = new Number(number);

In the place of number, if you provide any non-number argument, then the argument cannot be

converted into a number, it returns NaN (Not-a-Number).

Number Properties

Here is a list of each property and their description.

Sr.No.	Property & Description
1	MAX_VALUE The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	MIN_VALUE The smallest possible value a number in JavaScript can have 5E-324
3	NaN Equal to a value that is not a number.
4	NEGATIVE_INFINITY A value that is less than MIN_VALUE.
5	POSITIVE_INFINITY A value that is greater than MAX_VALUE
6	prototype A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	constructor Returns the function that created this object's instance. By default this is the Number object.

Number Methods

The Number object contains only the default methods that are a part of every object's definition.

Sr.No.	Method & Description
1	toExponential() Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.

2	toFixed() Formats a number with a specific number of digits to the right of the decimal.
3	toLocaleString() Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	toPrecision() Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	toString() Returns the string representation of the number's value.
6	valueOf() Returns the number's value.

JavaScript Number - toExponential()

This method returns a string representing the number object in exponential notation.

Syntax

Its syntax is as follows − number.toExponential([fractionDigits])

Parameter Details

fractionDigits − An integer specifying the number of digits after the decimal point. Defaults to as many digits as necessary to specify the number.

Return Value

A string representing a Number object in exponential notation with one digit before the decimal point, rounded to fraction. Digits digits after the decimal point. If the fraction Digits argument is omitted, the number of digits after the decimal point defaults to the number of digits necessary to represent the value uniquely.

Example

```
<html>

<head>

<title>Javascript Method toExponential()</title>

</head>
```

```
<body>

<script type = "text/javascript"> var num = 77.1234;

var val = num.toExponential(); document.write("num.toExponential() is : " + val ); document.write("<br />");

val = num.toExponential(4); document.write("num.toExponential(4) is : " + val ); document.write("<br />");

val = num.toExponential(2); document.write("num.toExponential(2) is : " + val); document.write("<br />");

val = 77.1234.toExponential(); document.write("77.1234.toExponential()is : " + val ); document.write("<br />");

val = 77.1234.toExponential(); document.write("77 .toExponential() is : " + val);

</script>

</body>

</html>
```

Output

```
num.toExponential() is : 7.71234e+1

num.toExponential(4) is : 7.7123e+1

num.toExponential(2) is : 7.71e+1

77.1234.toExponential()is:7.71234e+1

77 .toExponential() is : 7.71234e+1
```

JavaScript Number - valueOf()

This method returns the primitive value of the specified number object.

Syntax

Its syntax is as follows − number.valueOf()

Return Value

Returns the primitive value of the specified number object.

```
<html>

<head>

<title>JavaScript valueOf() Method </title>

</head>
```

```
<body>
<script type = "text/javascript">
var num = new Number(15.11234); document.write("num.valueOf() is " + num.valueOf());
</script>
</body>
</html>
```

Output

```
num.valueOf() is 15.11234
```

Boolean

Two values, either "true" or "false". If value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Syntax

Use the following syntax to create a boolean object. var val = new Boolean(value);

JavaScript - Boolean valueOf() Method

JavaScript boolean valueOf() method returns the primitive value of the specified boolean object.

Syntax

Its syntax is as follows − boolean.valueOf()

Return Value

Returns the primitive value of the specified boolean object.

Example

```
<html>
<head>
<title>JavaScript valueOf() Method</title>
</head>
<body>
<script type = "text/javascript"> var flag = new Boolean(false);
document.write( "flag.valueOf is : " + flag.valueOf() );
</script>
</body>
</html>
```

Output

flag.valueOf is : false

To work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods. As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

Syntax

Use the following syntax to create a String object − var val = new String(string);

The String parameter is a series of characters that has been properly encoded.

String Properties

Here is a list of the properties of String object and their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the String function that created the object.
2	length Returns the length of the string.
3	prototype The prototype property allows you to add properties and methods to an object.

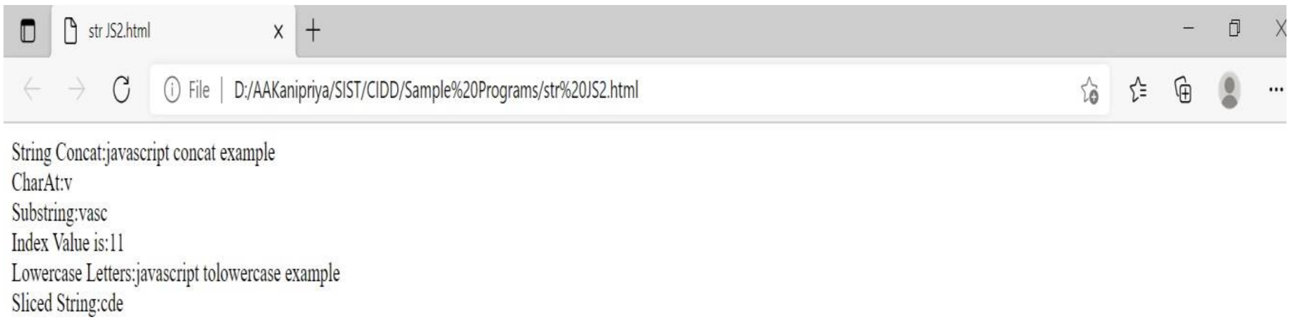
String Methods

Here is a list of the methods available in String object along with their description.

Sr.No.	Method & Description
1	charAt() Returns the character at the specified index.
2	charCodeAt() Returns a number indicating the Unicode value of the character at the given index.
3	concat() Combines the text of two strings and returns a new string.

4	indexOf() Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf() Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare() Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	match() Used to match a regular expression against a string.

```
<!DOCTYPE html>
<html>
<body>
<script>
var s1="javascript ";
var s2="concat example"; var s3=s1.concat(s2);
document.write("String Concat:"+s3+"</br>"); var str="javascript";
document.write("CharAt:"+str.charAt(2)+"</br>");
document.write("Substring:"+str.substr(2,4)+"</br>"); var s1="javascript from sample programs";
var n=s1.indexOf("from"); document.write("Index Value is:"+n+"</br>"); var s1="JavaScript
toLowerCase Example"; var s2=s1.toLowerCase();
document.write("Lowercase Letters:"+s2+"</br>"); var s1="abcdefgh";
var s2=s1.slice(2,5); document.write("Sliced String:"+s2);
</script>
</body>
</html>
```


str JS2.html
File | D:/AAKanipriya/SIST/CIDD/Sample%20Programs/str%20JS2.html

```
String Concat:javascript concat example
CharAt:v
Substring:vasc
Index Value is:11
Lowercase Letters:javascript tolowercase example
Sliced String:cde
```

JavaScript - The Arrays Object

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

Syntax

Use the following syntax to create an Array object − var fruits = new Array("apple", "orange", "mango");

The Array parameter is a list of strings or integers. When you specify a single numeric parameter

with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows −

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

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

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

Array Properties

Here is a list of the properties of the Array object along with their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the array function that created the object.
2	**index** The property represents the zero-based index of the match in the string

3	**input** This property is only present in arrays created by regular expression matches.
4	length Reflects the number of elements in an array.
5	prototype The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every() Returns true if every element in this array satisfies the provided testing function.
3	filter() Creates a new array with all of the elements of this array for which the provided filtering function returns true.

JavaScript - Array concat() Method

```
<html>

<head>

<title>JavaScript Array concat Method</title>

</head>

<body>

<script type = "text/javascript"> var alpha = ["a", "b", "c"];

var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric); document.write("alphaNumeric : " + alphaNumeric );
</script>
</body>
</html>
```

Output

```
alphaNumeric : a,b,c,1,2,3
```

JavaScript - Array pop() Method

```
<html>
<head>
<title>JavaScript Array pop Method</title>
</head>
<body>
<script type = "text/javascript"> var numbers = [1, 4, 9];
var element = numbers.pop(); document.write("element is : " + element );
var element = numbers.pop(); document.write("<br />element is : " + element );
</script>
</body>
</html>
```

Output

```
element is : 9 element is : 4
```

Date Object

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.
- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

Syntax:

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

Example:

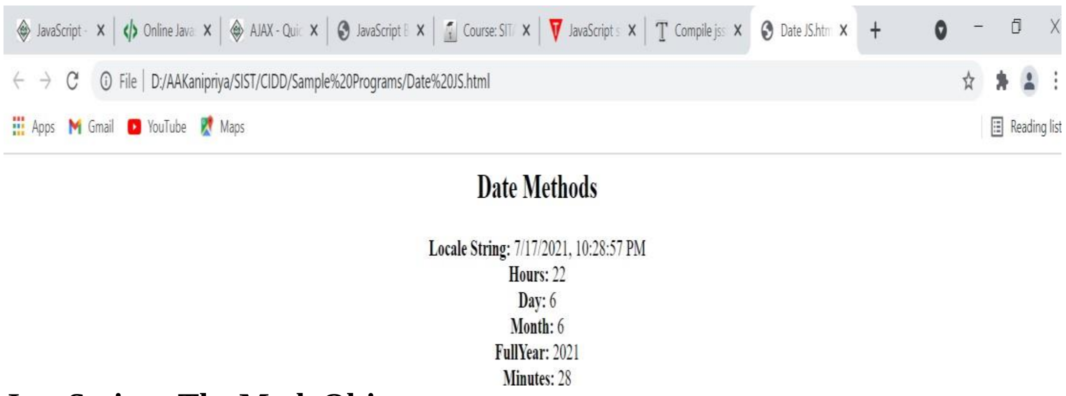
```
var current_date = new Date();
```

Date Methods

Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.
getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.
setMilliseconds()	Sets the milliseconds.

```
<html>
<body>
<center>
<h2>Date Methods</h2>
<script type="text/javascript"> var d = new Date();
document.write("<b>Locale String:</b> " + d.toLocaleString()+"<br>");
document.write("<b>Hours:</b> " + d.getHours()+"<br>"); document.write("<b>Day:</b> " +
d.getDay()+"<br>"); document.write("<b>Month:</b> " + d.getMonth()+"<br>");
document.write("<b>FullYear:</b> " + d.getFullYear()+"<br>");
document.write("<b>Minutes:</b> " + d.getMinutes()+"<br>");
</script>
```

</center>
</body>
</html>



Date Methods

Locale String: 7/17/2021, 10:28:57 PM
Hours: 22
Day: 6
Month: 6
FullYear: 2021
Minutes: 28

JavaScript - The Math Object

The math object provides you properties and methods for mathematical constants and functions. Unlike other global objects, Math is not a constructor. All the properties and methods of Math are static and can be called by using Math as an object without creating it.

Thus, you refer to the constant pi as Math.PI and you call the sine function as Math.sin(x), where x is the method's argument.

Syntax

The syntax to call the properties and methods of Math are as follows var pi_val = Math.PI;

var sine_val = Math.sin(30);

Math Methods

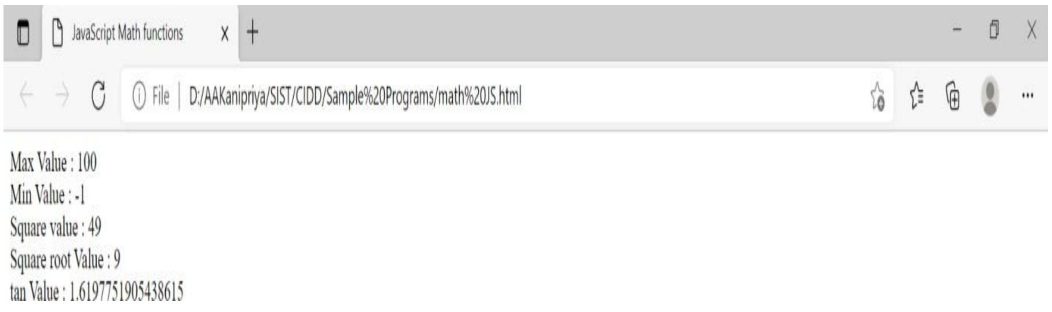
Here is a list of the methods associated with Math object and their description

Sr.No.	Method & Description
1	abs() Returns the absolute value of a number.
2	acos() Returns the arccosine (in radians) of a number.
3	asin() Returns the arcsine (in radians) of a number.
4	atan() Returns the arctangent (in radians) of a number.

5	atan2() Returns the arctangent of the quotient of its arguments.
6	ceil() Returns the smallest integer greater than or equal to a number.

```
<html>
<head>
<title>JavaScript Math functions </title>
</head>
<body>
<script type = "text/javascript">
var value = Math.max(10, 20, -1, 100); document.write("Max Value : " + value );
var value = Math.min(10, 20, -1, 100); document.write("<br />Min Value : " + value );
var value = Math.pow(7, 2); document.write("<br />Square value : " + value );
var value = Math.sqrt( 81 );
document.write("<br />Square root Value : " + value ); var value = Math.tan( 45 );
document.write("<br />tan Value : " + value );
</script>
</body>
</html>
```

Output


JavaScript Math functions
File | D:/AAKanipriya/SIST/CIDD/Sample%20Programs/math%20JS.html

```
Max Value : 100
Min Value : -1
Square value : 49
Square root Value : 9
tan Value : 1.6197751905438615
```

Window Object

- Window object is a top-level object in Client-Side JavaScript.
- Window object represents the browser's window.
- It represents an open window in a browser.

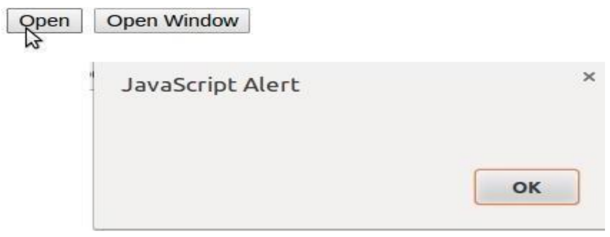
- It supports all browsers.
- The document object is a property of the window object. So, typing window.document.write is same as document.write.
- All global variables are properties and functions are methods of the window object.

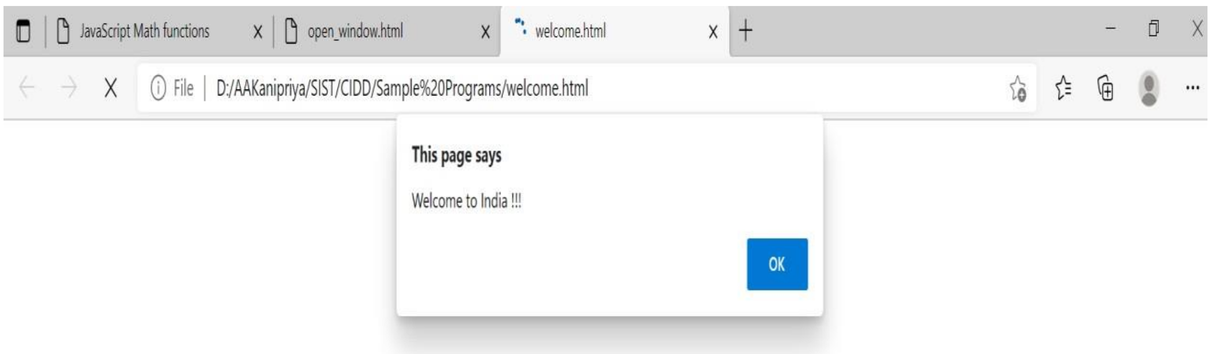
Property	Description
Document	It returns the document object for the window (DOM).
Frames	It returns an array of all the frames including iframes in the current window.
Closed	It returns the boolean value indicating whether a window has been closed or not.
History	It returns the history object for the window.
innerHeight	It sets or returns the inner height of a window's content area.
innerWidth	It sets or returns the inner width of a window's content area.
Length	It returns the number of frames in a window.
Location	It returns the location object for the window.
Name	It sets or returns the name of a window.
Navigator	It returns the navigator object for the window.
Opener	It returns a reference to the window that created the window.
outerHeight	It sets or returns the outer height of a window, including toolbars/scrollbars.

```
<html>
<head>
<script type="text/javascript"> function openwindow()
{
window.open("welcome.html");
}
</script>
</head>
<body>
<form>
<input type="button" value="Open" onClick=window.alert()>
<input type="button" onClick="openwindow()" value="Open Window">
```

```
</form>
</body>
</html>
welcome.html //File name
<html>
<body>
<script type="text/javascript">
{
document.write("<b>Welcome to India !!!</b>");
}
</script>
 </body>
</html>
```

Output:





D:/AAKanipriya/SIST/CIDD/Sample%20Programs/welcome.html

This page says

Welcome to India !!!

- In the above JavaScript program, when you click on the 'Open Window', you will see the 'welcome.html' opened in another window.
- When you click on the 'Open' button, you will see the alert message box.

Navigator Object

- Navigator object is the representation of Internet browser.
- It contains all the information about the visitor's (client) browser.

```
<html>
```

Property	Description
appName	It returns the name of the browser.
appCodeName	It returns the code name of the browser.

```
<body>

<script type="text/javascript">

document.write("<b>Browser: </b>"+navigator.appName+"<br><br>");
document.write("<b>Browser Version: </b>"+navigator.appVersion+"<br><br>");
document.write("<b>Browser Code: </b>"+navigator.appCodeName+"<br><br>");
document.write("<b>Platform: </b>"+navigator.platform+"<br><br>");
document.write("<b>Cookie Enabled: </b>"+navigator.cookieEnabled+"<br><br>");
document.write("<b>User Agent: </b>"+navigator.userAgent+"<br><br>");
document.write("<b>Java Enabled: </b>"+navigator.javaEnabled()+"<br><br>");

</script>

</body>

 </html>
```

Frame Object

Frame object represents an HTML frame which defines one particular window(frame) within a frameset.
It defines the set of frame that make up the browser window.
It is a property of the window object.
Syntax:<frame>
It has no end tag but they need to be closed properly.
It is an HTML element.
It defines a particular area in which another HTML document can be displayed.
A frame should be used within a <FRAMESET> tag.
<FRAME> Tag Attributes

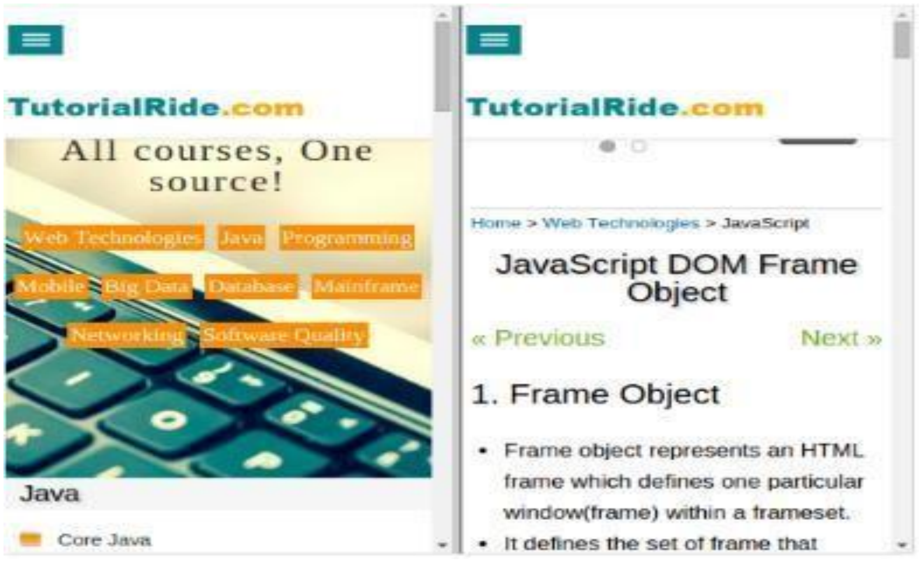
Frameset Object

Frameset object holds two or more frame elements and each frame element in turn holds a separate document.
This object states only how many columns or rows there will be in the frameset.
Syntax:
<frameset> . . . </frameset>

<FRAMESET> Tag Attributes

<frameset cols="50%,50%">

```
<frame src="<frame src="http://www.tutorialride.com" />
<frame src="http://www.tutorialride.com/javascript/javascript-dom-frame-object.htm" />
</frameset>
```


TutorialRide.com

All courses, One source!

Web Technologies Java Programming Mobile Big Data Database Mainframe Networking Software Quality

Java

Core Java

TutorialRide.com

Home > Web Technologies > JavaScript

JavaScript DOM Frame Object

« Previous Next »

1. Frame Object

- Frame object represents an HTML frame which defines one particular window(frame) within a frameset.
- It defines the set of frame that

IFrame Object

IFrame object represents an HTML inline frame that contains another document.

Frame Object Properties

Property	Description
Align	It aligns the iframe.
contentDocument	It returns the document object generated by a frame/iframe.
contentWindow	It returns the window object generated by a frame/iframe.
frameBorder	It sets frame border in a frame/iframe.
Height	It sets the height in an iframe.

Link Object

Link object represents an HTML link element.
It is used to link to external stylesheets.
It must be placed inside the head section of an HTML document.
It specifies a link to an external resources.

Link Object Properties

Property	Description
href	It sets the URL of a linked document.
host	It sets the URL host name and port.
hostname	It sets the URL hostname.
pathname	It sets the URL pathname.
Port	It sets the URL port section.
Protocol	It sets the URL protocol section including the colon (:) after protocol name.
Search	It sets the URL query string section that is after including the questions mark (?).
Target	It sets the URL link's target name.

Form Object

Form object represents an HTML form.
It is used to collect user input through elements like text fields, check box and radio button, select option, text area, submit buttons and etc.

Syntax:
<form> . . . </form>

Form Object Properties

Property	Description
Action	It sets and returns the value of the action attribute in a form.
enctype	It sets and returns the value of the enctype attribute in a form.
Length	It returns the number of elements in a form.
Method	It sets and returns the value of the method attribute in a form that is GET or POST.
Name	It sets and returns the value of the name attribute in a form.
Target	It sets and returns the value of the target attribute in a form.

Hidden Object

- Hidden object represents a hidden input field in an HTML form and it is invisible to the user.
- This object can be placed anywhere on the web page.
- It is used to send hidden form of data to a server.

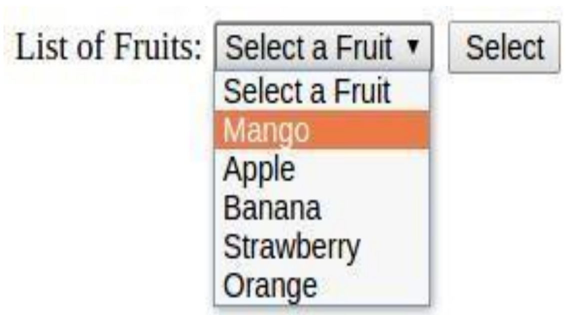
Syntax:

<input type= “hidden”>

Property	Description
Name	It sets and returns the value of the name attribute of the hidden input field.
Type	It returns type of a form element.

```
<html>
<head>
<script type="text/javascript"> function optionfruit(select)
{
var a = select.selectedIndex;
var fav = select.options[a].value; if(a==0)
{
alert("Please select a fruit");
}
else
{
document.write("Your Favorite Fruit is <b>"+fav+".</b>");
}
}
</script>
</head>
<body>
<form>
List of Fruits:
<select name="fruit">
<option value="0">Select a Fruit</option>
<option value="Mango">Mango</option>
<option value="Apple">Apple</option>
<option value="Banana">Banana</option>
<option value="Strawberry">Strawberry</option>
<option value="Orange">Orange</option>
```

```
</select>
<input type="button" value="Select" onClick="optionfruit(this.form.fruit);">
</form>
</body>
</html>
```



What is AJAX?

AJAX = Asynchronous JavaScript And XML. AJAX is not a programming language. AJAX just uses a combination of: A browser built-in XMLHttpRequest object (to request data from a web server) JavaScript and HTML DOM (to display or use the data) AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works

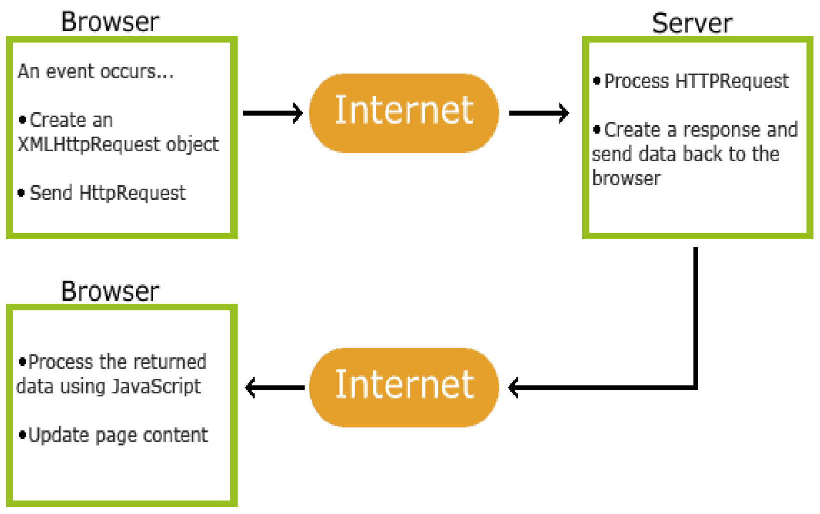

Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

Fig 3.3 AJAX Work Flow

An event occurs in a web page (the page is loaded, a button is clicked)

An XMLHttpRequest object is created by JavaScript

The XMLHttpRequest object sends a request to a web server

The server processes the request

The server sends a response back to the web page

The response is read by JavaScript

Proper action (like page update) is performed by JavaScript

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object. The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
var xhttp = new XMLHttpRequest();
```

Old Versions of Internet Explorer (IE5 and IE6)

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX object instead of the XMLHttpRequest object:

```
variable = new ActiveXObject("Microsoft.XMLHTTP");
```

To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveX object:

```
if (window.XMLHttpRequest) {
// code for modern browsers xmlhttp = new XMLHttpRequest();
} else {
// code for old IE browsers
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

The XMLHttpRequest object is used to exchange data with a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

xhttp.open("GET", "ajax_info.txt", true); xhttp.send();

GET or POST?

GET is simpler and faster than POST, and can be used in most cases. However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

POST Requests

A simple POST request:
xhttp.open("POST", "demo_post.asp", true);
xhttp.send();
AJAX Example

```
<!DOCTYPE html>

<html>

<body>

<div id="demo">

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Change Content</button>

</div>

<script>

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();
}

</script>

</body>

</html>
```

ajax_info.txt
```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server: xhttp.open("GET", "ajax_test.asp", true);
The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .ph can perform actions on the server before sending the response back

Asynchronous - True or False?

Server requests should be sent asynchronously.The async parameter of the open() method should be set to true:
xhttp.open("GET", "ajax_test.asp", true);

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- execute other scripts while waiting for server response
- deal with the response after the response is read

The onreadystatechange Property With the XMLHttpRequest object you can define a function to be executed when the request receives an answer. The function is defined in the onreadystatechange property of the XMLHttpResponse object:

```
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();
```

Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false: xhttp.open("GET", "ajax_info.txt", false);
Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.Since the code will wait for server completion, there is no need for an onreadystatechange function:
```
xhttp.open("GET", "ajax_info.txt", false);
xhttp.send();
document.getElementById("demo").innerHTML = xhttp.responseText;
```
AJAX - Server Response
The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a function to be executed when the readyState changes. The status property and the statusText property holds the status of the XMLHttpRequest object. The onreadystatechange function is called every time the readyState changes. When readyState is 4 and status is 200, the response is ready:

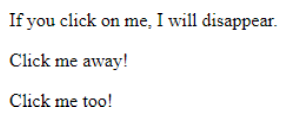
```
function loadDoc() {
var xhttp = new XMLHttpRequest(); xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) { document.getElementById("demo").innerHTML
= this.responseText;
}
};
xhttp.open("GET", "ajax_info.txt", true); xhttp.send();
}
```

Jquery

jQuery is a JavaScript Library. jQuery greatly simplifies JavaScript programming. jQuery is easy to learn. With our online editor, you can edit the code, and click on a button to view the result.

Example
```
$(document).ready(function(){
$("p").click(function(){
$(this).hide();
});
});
```

Output:



```
If you click on me, I will disappear.

Click me away!

Click me too!
```

jQuery There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

Download the jQuery library from jQuery.com
Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

Production version - this is for your live website because it has been minified and compressed
Development version - this is for testing and development (uncompressed and readable code) Both versions can be downloaded from jQuery.com. The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

jQuery CDN

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network). Google is an example of someone who host jQuery:

Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

With jQuery you select (query) HTML elements and perform "actions" on them.

jQuery Syntax

The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s). Basic syntax is: $(selector).action()

- A $ sign to define/access jQuery
- A (selector) to "query (or find)" HTML elements
- A jQuery action() to be performed on the element(s) Examples:

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test". The Document Ready Event
You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){
// jQuery methods go here...
});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).
It is good practice to wait for the document to be fully loaded and ready before working with it.
This also allows you to have your JavaScript code before the body of your document, in the head section.
Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

```
$(function(){
// jQuery methods go here...
});
```

Use the syntax you prefer. We think that the document ready event is easier to understand when reading the code.

jQuery Selectors
jQuery selectors are one of the most important parts of the jQuery library. jQuery selectors allow you to select and manipulate HTML element(s). jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors. All selectors in jQuery start with the dollar sign and parentheses: $(). The element Selector The jQuery element selector selects elements based on the element name. You can select all <p> elements on a page like this:

```
$("p")
```

Example

When a user clicks on a button, all <p> elements will be hidden:

Example:

```
$(document).ready(function(){
$("button").click(function(){
$("p").hide();
});
});
```

The #id Selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

Example

When a user clicks on a button, the element with id="test" will be hidden:

Example

```
$(document).ready(function(){
$("button").click(function(){
$("#test").hide();
});
});
```

The .class Selector

The jQuery .class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

Example

When a user clicks on a button, the elements with class="test" will be hidden:

```
$(document).ready(function(){
$("button").click(function(){
$(".test").hide();
});
});
```

Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the <head> section. However, sometimes it is preferable to place them in a separate file, like this (use the src attribute to refer to the .js file):

Example

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="my_jquery_functions.js"></script>
</head>
```

jQuery is tailor-made to respond to events in an HTML page. What are Events?
All the different visitors' actions that a web page can respond to are called events. An event represents the precise moment when something happens.
Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term "fires/fired" is often used with events. Example: "The keypress event is fired; the moment you press a key".

jQuery Traversing Traversing
jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.

Traversing the DOM
jQuery provides a variety of methods that allow us to traverse the DOM. The largest category of traversal methods is tree-traversal.
jQuery Traversing - Ancestors

With jQuery you can traverse up the DOM tree to find ancestors of an element. An ancestor is a parent, grandparent, great-grandparent, and so on.

Traversing Up the DOM Tree

Three useful jQuery methods for traversing up the DOM tree are:

- parent()
- parents()
- parentsUntil() jQuery parent() Method

The parent() method returns the direct parent element of the selected element. This method only traverse a single level up the DOM tree. The following example returns the direct parent element of each <span> elements: Example

```
$(document).ready(function(){
$("span").parent();
});
```

jQuery parents() Method

The parents() method returns all ancestor elements of the selected element, all the way up to the document's root element (<html>).
The following example returns all ancestors of all <span> elements: Example

```
$(document).ready(function(){
$("span").parents();
});
```

You can also use an optional parameter to filter the search for ancestors.
The following example returns all ancestors of all <span> elements that are <ul> elements:
Example

```
$(document).ready(function(){
$("span").parents("ul");
});
```

jQuery parentsUntil() Method

The parentsUntil() method returns all ancestor elements between two given arguments.
The following example returns all ancestor elements between a <span> and a <div> element:
Example

```
$(document).ready(function(){
$("span").parentsUntil("div");
});
```

jQuery Traversing - Descendants

With jQuery you can traverse down the DOM tree to find descendants of an element. A descendant is a child, grandchild, great-grandchild, and so on.
Traversing Down the DOM Tree

Two useful jQuery methods for traversing down the DOM tree are:

- children()
- find()

jQuery children() Method

The children() method returns all direct children of the selected element. This method only traverses a single level down the DOM tree.
The following example returns all elements that are direct children of each <div> elements:

Example

```
$(document).ready(function(){
$("div").children();
});
```

You can also use an optional parameter to filter the search for children.

The following example returns all <p> elements with the class name "first", that are direct children of <div>:

Example
```
$(document).ready(function(){
$("div").children("p.first");
});
```

jQuery find() Method

The find() method returns descendant elements of the selected element, all the way down to the last descendant.
The following example returns all <span> elements that are descendants of <div>: Example
```
$(document).ready(function(){
$("div").find("span");
});
```

The following example returns all descendants of <div>:

Example
```
$(document).ready(function(){
$("div").find("*");
});
```

jQuery Traversing - Siblings
With jQuery you can traverse sideways in the DOM tree to find siblings of an element. Siblings share the same parent.

Traversing Sideways in The DOM Tree

There are many useful jQuery methods for traversing sideways in the DOM tree:

- siblings()
- next()
- nextAll()
- nextUntil()
- prev()
- prevAll()
- prevUntil() jQuery siblings() Method

The siblings() method returns all sibling elements of the selected element. The following example returns all sibling elements of <h2>:
Example
```
$(document).ready(function(){
$("h2").siblings();
});
```

You can also use an optional parameter to filter the search for siblings.
The following example returns all sibling elements of <h2> that are <p> elements: Example
```
$(document).ready(function(){
$("h2").siblings("p");
});
```

jQuery next() Method
The next() method returns the next sibling element of the selected element. The following example returns the next sibling of <h2>:
Example
```
$(document).ready(function(){
$("h2").next();
});
```

jQuery nextAll() Method
The nextAll() method returns all next sibling elements of the selected element.

The following example returns all next sibling elements of <h2>:

Example
```
$(document).ready(function(){
$("h2").nextAll();
});
```

jQuery nextUntil() Method
The nextUntil() method returns all next sibling elements between two given arguments.
The following example returns all sibling elements between a <h2> and a <h6> element: Example
```
$(document).ready(function(){
$("h2").nextUntil("h6");
});
```

jQuery prev(), prevAll() & prevUntil() Methods
The prev(), prevAll() and prevUntil() methods work just like the methods above but with reverse functionality: they return previous sibling elements (traverse backwards along sibling elements in the DOM tree, instead of forward).

jQuery Traversing - Filtering

The first(), last(), eq(), filter() and not() Methods

The most basic filtering methods are first(), last() and eq(), which allow you to select a specific element based on its position in a group of elements.

Other filtering methods, like filter() and not() allow you to select elements that match, or do not match, a certain criteria.

jQuery first() Method

The first() method returns the first element of the specified elements. The following example selects the first <div> element:
Example
```
$(document).ready(function(){
$("div").first();
});
```

jQuery last() Method

The last() method returns the last element of the specified elements. The following example selects the last <div> element:
Example
```
$(document).ready(function(){
$("div").last();
});
```

jQuery eq() method

The eq() method returns an element with a specific index number of the selected elements.

The index numbers start at 0, so the first element will have the index number 0 and not 1. The following example selects the second <p> element (index number 1):

Example
```
$(document).ready(function(){
$("p").eq(1);
});
```

jQuery filter() Method

The filter() method lets you specify a criteria. Elements that do not match the criteria are removed from the selection, and those that match will be returned.
The following example returns all <p> elements with class name "intro": Example
```
$(document).ready(function(){
$("p").filter(".intro");
});
```

jQuery not() Method

The not() method returns all elements that do not match the criteria.

Tip: The not() method is the opposite of filter().
The following example returns all <p> elements that do not have class name "intro": Example
```
$(document).ready(function(){
$("p").not(".intro");
});
```

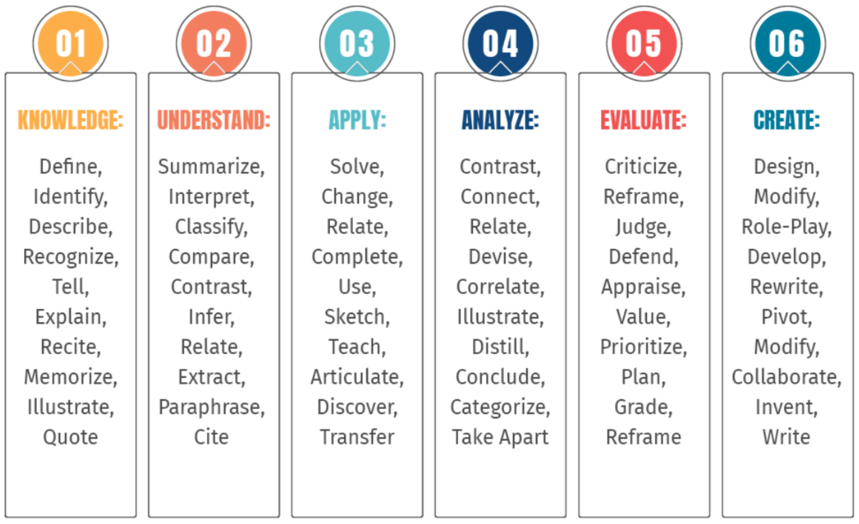
TEXT / REFERENCE BOOKS
1. Thomas A Powell, Fritz Schneider, “JavaScript: The Complete Reference”, 3rd Edition, Tata McGraw Hill, 2013.
2. David Flanagan, “JavaScript: The Definitive Guide, 6th Edition”, O'Reilly Media, 2011.
3. Bear Bibeault and Yehuda Katz, “jQuery in Action”, January 2008.

Question Bank

Part-A			
Q.No	**Questions (2 Marks)**	**Competence**	**BT Level**
1.	Write the JavaScript to print "Good Day" using IF-ELSEcondition.	Create	BTL5
2.	What is a JavaScript statement? Give an example.	Knowledge	BTL1
3.	Write some advantages of using JavaScript.	Create	BTL2
4.	List the JavaScript datatypes.	Knowledge	BTL1
5.	Differentiate the return and continue statements in JavaScript	Analyze	BTL4
6.	Construct the statement to read the properties of an object injavascript.	Create	BTL6
7.	Identify the built-in method returns the character at thespecified index?	Understand	BTL2
8.	List out the properties to determine the size of the browserwindow.	Knowledge	BTL1
9.	Define Java Script.	Knowledge	BTL1
10.	Write the syntax to create an object in javascript.	Create	BTL6
11.	Express the statement in java script to display content onwebpage.	Understand	BTL2
12.	Connect the role of a callback function in performing a partialpage update in an AJAX application.	Analyze	BTL4
13.	Create an appropriate Javascript code to remove an element(current element) from a DOM.	Create	BTL6
14.	Write down the Javascript code to swap the contents of thetwo text boxes <form name="form1"> Input text 1: <input type="text" name="A">Input text 2: <input type="text" name="B"> </form>	Create	BTL6
15.	Summarize benefits of using Javascript code in an HTMLdocument.	Understand	BTL2
Part-B			

Q.No	Questions (16 Marks)	Competence	BT Level
1.	Explain how local and global functions can be written usingJavaScript with example	Understand	BTL2
2.	Write JavaScript to find sum of first ‘n’ even numbers anddisplay the result. Get the value of ‘n’ from user.	Create	BTL6
3.	Write JavaScript to find factorial of a given number.	Create	BTL6
4.	Show in detail about JavaScript variables and operators.	Apply	BTL3
5.	Classify the various kinds of javascript pop up boxes withexample programs.	Analyze	BTL4
6.	Explain about Math object and the various methods injavascript with example program.	Understand	BTL2
7.	Demonstrate the working process of AJAX. Develop the program to retrieve the names of the artists from an XMLfile.	Apply	BTL3
8.	Summarize the methods associated with array object inJavaScript. Explain any two methods with an example.	Evaluate	BTL5
9.	Develop a javascript program to perform the email validation. Assume the following criteria to validate the email id. i. email id must contain the @ and . character ii. There must be at least one character before and afterthe @. iii. There must be at least two characters after . (dot).	Create	BTL6
10.	Use Javascript and HTML to create a page with two panes. The first pane (on the left) should have a text area where HTML code can be typed by the user. The pane on the rightside should display the preview of the HTML code typed bythe user, as it would be seen on the browser.	Apply	BTL3

Bloom’s Taxonomy



01	02	03	04	05	06
KNOWLEDGE:	UNDERSTAND:	APPLY:	ANALYZE:	EVALUATE:	CREATE:
Define, Identify, Describe, Recognize, Tell, Explain, Recite, Memorize, Illustrate, Quote	Summarize, Interpret, Classify, Compare, Contrast, Infer, Relate, Extract, Paraphrase, Cite	Solve, Change, Relate, Complete, Use, Sketch, Teach, Articulate, Discover, Transfer	Contrast, Connect, Relate, Devise, Correlate, Illustrate, Distill, Conclude, Categorize, Take Apart	Criticize, Reframe, Judge, Defend, Appraise, Value, Prioritize, Plan, Grade, Reframe	Design, Modify, Role-Play, Develop, Rewrite, Pivot, Modify, Collaborate, Invent, Write

UNIT – IV – INTRODUCTION TO SERVER SIDE JS FRAMEWORK NODE.JS – SITA3004

IV INTRODUCTION TO SERVER SIDE JS FRAMEWORK NODE.JS

Introduction - What is Node JS – Architecture – Feature of Node JS - Installation and Setup - Creating web servers with HTTP (Request & Response) – Event Handling - GET & POST implementation - Connect to SQL Database using Node JS – Implementation of CRUD operations.

What is Node.js?

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentation

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.

- 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 Events of Node.js helps the server to get a response from the previous API call.
- Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.
- License − Node.js is released under the MIT license.

Who Uses Node.js?

Following is the link on github wiki containing an exhaustive list of projects, applications and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, and Yammer to name a few.

- Projects, Applications, and Companies Using Node

Where to Use Node.js?

Following are the areas where Node.js is proving itself as a perfect technology partner.

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

NodeJS Architecture

Node JS applications use the “Single Threaded Event Loop Model” architecture to handle multiple concurrent clients. I would like to explain the architecture of NodeJS with the help of the diagram inline below :

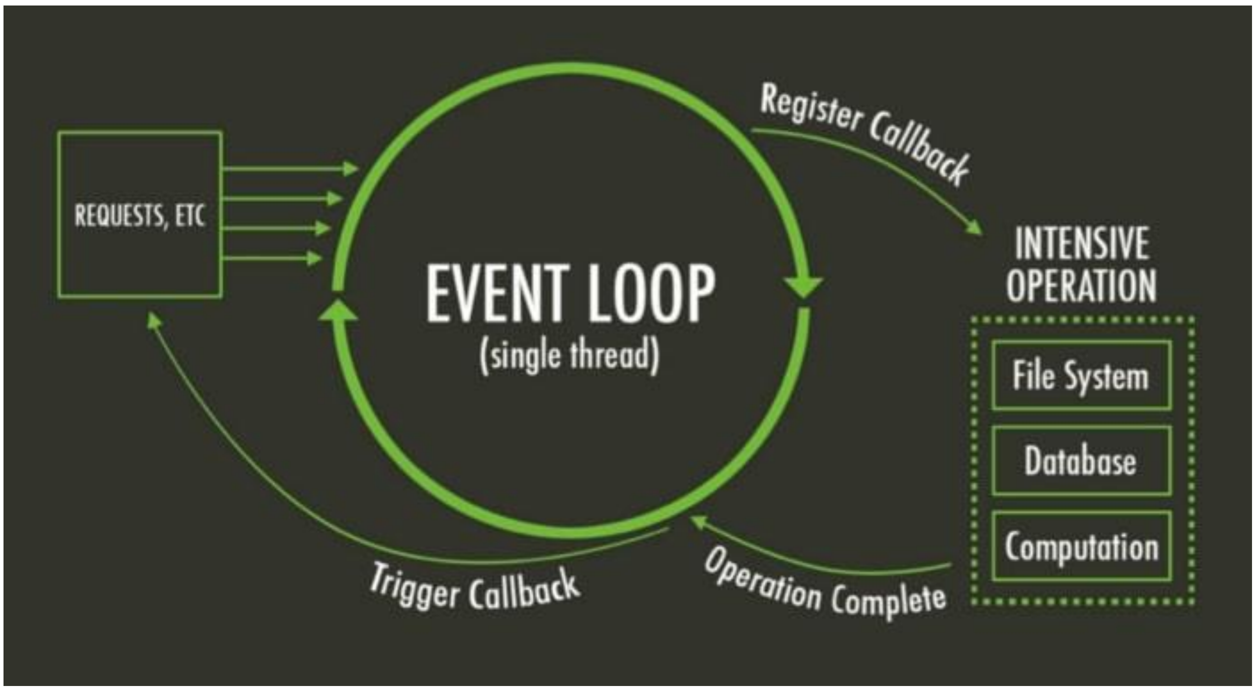


Fig 4.1 Node JS Architecture

Basically, NodeJS architecture contains 3 layers:

1. Request Queue
2. Single-Threaded Event Loop
3. Callback Operations

Let me help you understand every layer one by one in the simplest way possible. For the same let's consider a request statement as quoted below.

Rajeev's mom asked him to wash clothes and recharge the set-top box.

Request Queue

Here if we try to decode the above-quoted statement then we can decipher 2 requests/tasks for Rajeev:

1. Rajeev will put clothes in the washing machine.

2. Rajeev needs to recharge the expired set-top box TV connection.

Once NodeJS receives the quoted request, it will generate 2 requests and put them in a request queue internally and will wait for the Event Loop to process them.

Single-Threaded Event Loop

From Request Queue, Event Loop will pick up the first request and put that request into operation. Event Loop will check whether this request is Synchronous or Asynchronous.

Request 1 has a dependency on the washing machine which makes it an asynchronous request. So, till the time washing machine is not done washing the clothes, Rajeev has 2 choices to make a decision from :

1. Either he has a choice to wait for 20 minutes and sit idle.
2. Or he can in the meantime recharge the set-top box.

If Rajeev, decides to use his time wisely then he will work on executing the second task instead of sitting idle. So he will let the washing machine complete the task and would like to be notified about its completion in the form of a Callback or here in the form of an Alarm from the Washing Machine. Feeling connected with NodeJS now? That's Great !!! Gladly you didn't notice but we have reached our final layer.

Callback Operations

Are you now scared of Callbacks and Promises? Okay, let me try to explain them.

Callbacks

A callback function is a function that is passed as an argument to another function. Callback functions are a technique that's possible in JavaScript because of the fact that functions are objects.

As per our above Request 1, our alarm served as a callback for Rajeev. Got it right?

Promises

Promises; just like in real life is a token of appreciation that will be fulfilled once a particular objective is achieved.

A mother promises her child to buy her child chocolate if she completes her homework.

Try to understand Request 1 and you'll find that the washing machine is internally programmed that it will ring the alarm once it completes the task of washing the clothes. So the Washing machine promised Rajeev that it will call back once the task is completed.

Installing Node.js

Step 0: The Quick Guide (TL;DR) to Get Node.js Installed on Windows

Here's the abbreviated guide, highlighting the major steps:

1. Open the official page for Node.js downloads and download Node.js for Windows by clicking the "Windows Installer" option
2. Run the downloaded Node.js .msi Installer - including accepting the license, selecting the destination, and authenticating for the install.
   - This requires Administrator privileges, and you may need to authenticate
3. To ensure Node.js has been installed, run node -v in your terminal - you should get something like v6.9.5
4. Update your version of npm with npm install npm --global
   - This requires Administrator privileges, and you may need to authenticate
5. Congratulations - you've now got Node.js installed, and are ready to start building!

Step 1: Download the Node.js .msi Installer

As the first step to installing Node.js on Windows, you'll need to download the installer. You'll be able to grab the installer from the official downloads page for Node.js.

You'll be able to download the Windows Node.js installer by clicking the Windows Installer option at the top of the page - when you click this, you'll get an MSI installer download. Make sure to save it somewhere that you'll be able to find it!

Step 2: Run the Node.js Installer
You've got the Windows Installer - great! Now, you need to install it on your PC. The installer is a pretty typical Wizard interface for installing software on Windows - there are a few steps to it, but you can have it done in under a minute. You can get through it by following the guide below:

- Welcome to the Node.js Setup Wizard
  - Select Next
- End-User License Agreement (EULA)
  - Check I accept the terms in the License Agreement
  - Select Next
- Destination Folder
  - Select Next
- Custom Setup
  - Select Next
- Ready to install Node.js
  - Select Install
  - *Note:* This step requires Administrator privlidges.
  - If prompted, authenticate as an Administrator
- Installing Node.js
  - Let the installer run to completion
- Completed the Node.js Setup Wizard
  - Click Finish

Step 3: Verify that Node.js was Properly installed
To double check that Node.js was installed fully on your PC, you can test the following command in your Command Prompt (regardless of if you're using cmd.exe, Powershell, or any other command prompt):

```
$ node -v
```

If Node.js was installed fully, the command prompt will print something similar to (but probably not *exactly*) this:

```
$ node -v // The command we ran - prints out the version of Node.js that's currently installed
v6.9.5 // The printed version of Node.js that's currently installed - v6.9.5 was the most current LTS release at the time of writing.
```

Step 4: Update the Local npm Version
As the final step in getting Node.js installed, we'll update your version of npm - the package manager that comes bundled with Node.js.
Node.js always ships with a specific version of npm - Node.js doesn't (and shouldn't!) automatically update npm. The release cycle of the npm CLI client isn't in sync with the Node.js releases. Because of this, there's almost *certainly* going to be a newer version of npm available than the one that is installed as a default in any given Node release.
To quickly and easily update npm, you can run the following command:

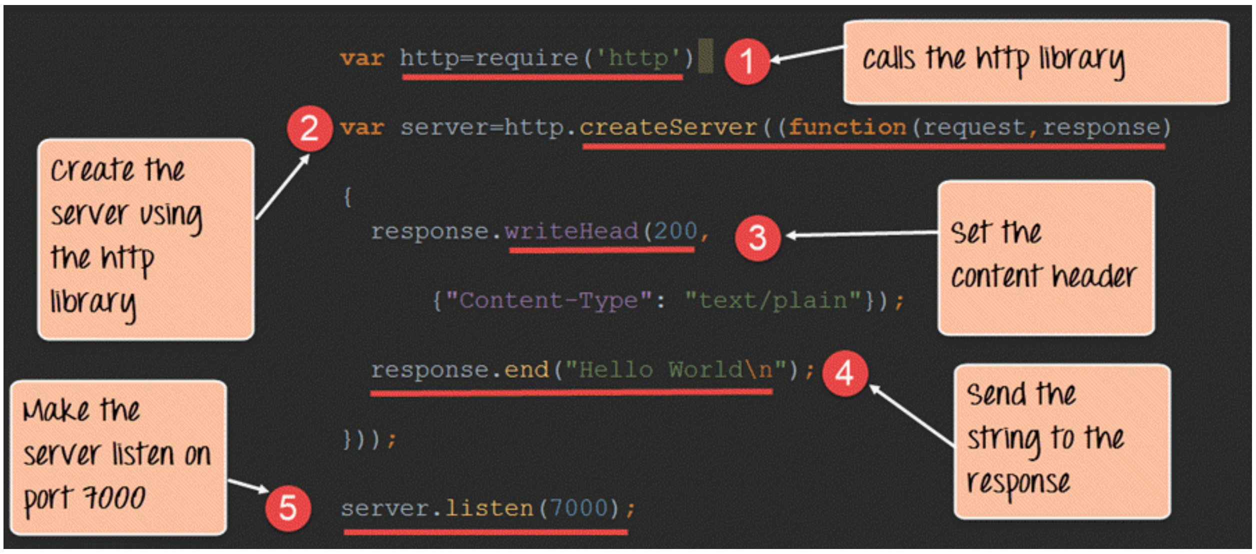
```
npm install npm --global // Update the `npm` CLI client
```

Step 5: Go build applications, APIs, tools, and more with Node.js!
Now you've got Node.js on a Windows machine. It's time to start exploring!

Creating Web Server with HTTP (Request & Response)

The Node.js framework is mostly used to create server-based applications. The framework can easily be used to create web servers which can serve content to users.

There are a variety of modules such as the “http” and “request” module, which helps in processing server related requests in the webserver space. We will have a look at how we can create a basic web server application using Node js.
Node as a web server using HTTP
Let’s look at an example of how to create and run our first Node js application.
Our application is going to create a simple server module which will listen on port no 7000. If a request is made through the browser on this port no, then server application will send a ‘Hello World’ response to the client.



```
var http=require('http')
var server=http.createServer((function(request,response)
{
  response.writeHead(200,
      {"Content-Type": "text/plain"});
  response.end("Hello World\n");
}));
server.listen(7000);
```

1
calls the http library
2
Create the server using the http library
3
Set the content header
4
Send the string to the response
5
Make the server listen on port 7000

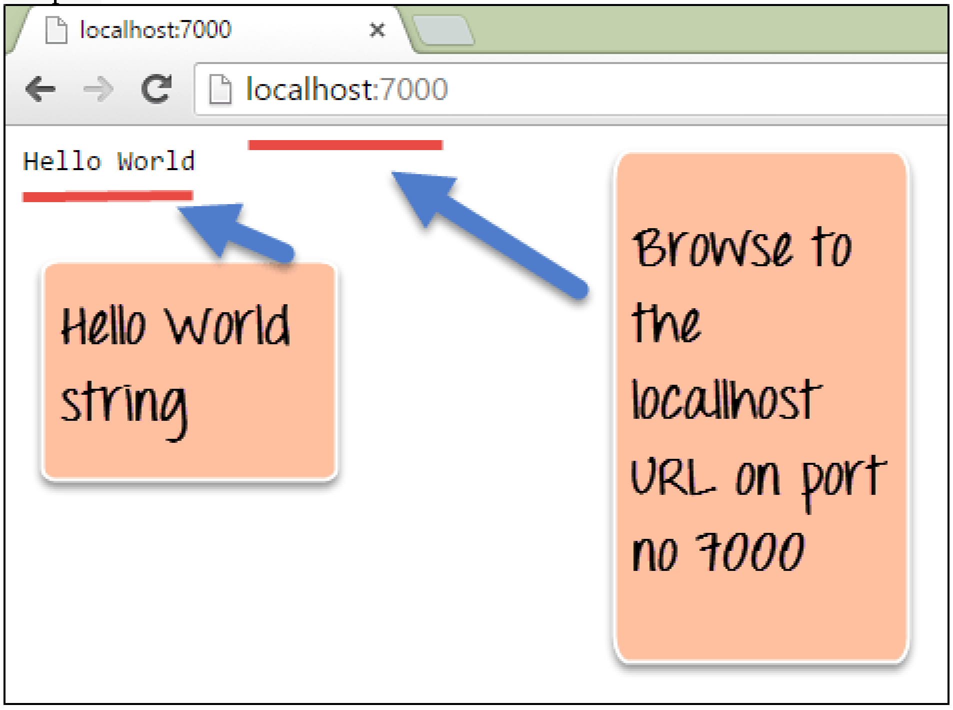
Fig 4.2 Request Response Process

Code Explanation:

1. The basic functionality of the require function is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object. So in our case, since we want to use the functionality of the http module, we use the require function to get the desired functions from the http module so that it can be used in our application.
2. In this line of code, we are creating a server application which is based on a simple function. This function is called whenever a request is made to our server application.
3. When a request is received, we are saying to send a response with a header type of ‘200.’ This number is the normal response which is sent in an http header when a successful response is sent to the client.
4. In the response itself, we are sending the string ‘Hello World.’
5. We are then using the server.listen function to make our server application listen to client requests on port no 7000. You can specify any available port over here.

If the command is executed successfully, the following Output will be shown when you run your code in the browser.

Output:



```
Hello World
```

Fig 4.3 Output

From the output,

- You can clearly see that if we browse to the URL of localhost on port 7000, you will see the string ‘Hello World’ displayed in the page.
- Because in our code we have mentioned specifically for the server to listen on port no 7000, we are able to view the output when browsing to this url.

Here is the code for your reference

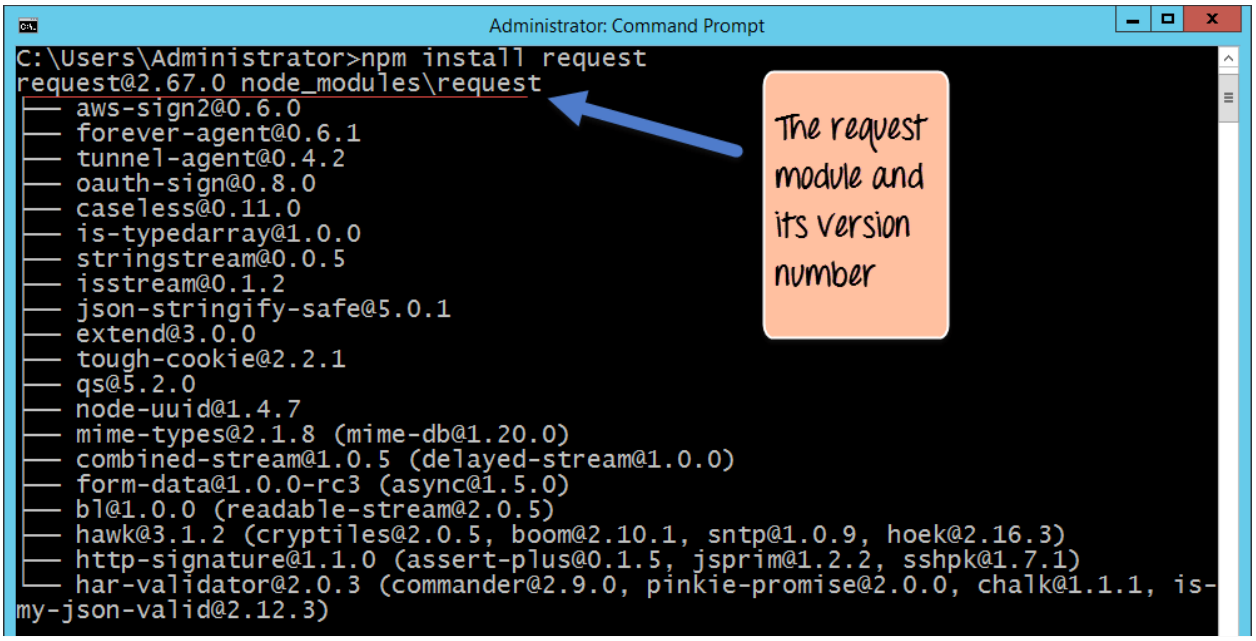
```
var http=require('http')
var server=http.createServer((function(request,response)
{
        response.writeHead(200,
        {"Content-Type" : "text/plain"});
        response.end("Hello World\n");
}));
server.listen(7000);
```

Handling GET Requests in Node.js

Making a GET Request to get the data from another site is relatively very simple in Node.js. To make a Get request in the node, we need to first have the request module installed. This can be done by executing the following line in the command line

npm install request

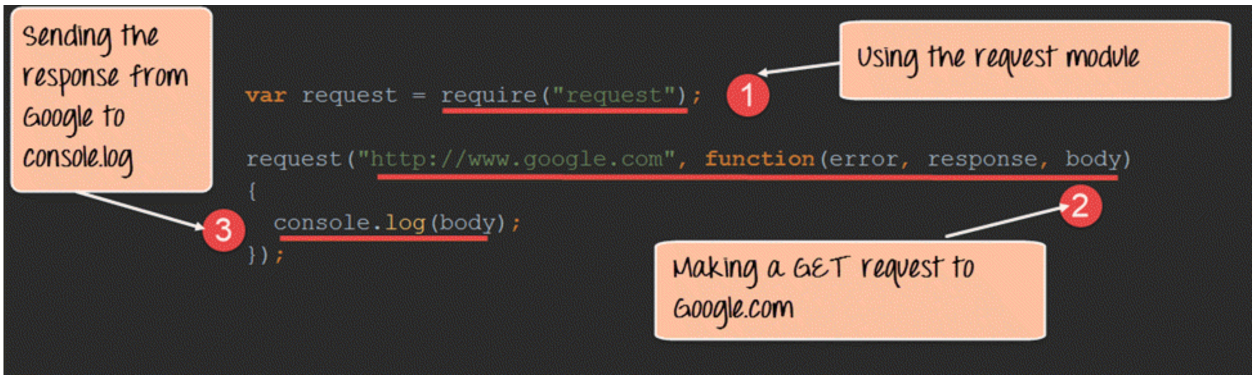
The above command requests the Node package manager to download the required request modules and install them accordingly. When your npm module has been installed successfully, the command line will show the installed module name and version: <name>@<version>.


Administrator: Command Prompt
The request module and its version number

```
C:\Users\Administrator>npm install request
request@2.67.0 node_modules\request
├── aws-sign2@0.6.0
├── forever-agent@0.6.1
├── tunnel-agent@0.4.2
├── oauth-sign@0.8.0
├── caseless@0.11.0
├── is-typedarray@1.0.0
├── stringstream@0.0.5
├── isstream@0.1.2
├── json-stringify-safe@5.0.1
├── extend@3.0.0
├── tough-cookie@2.2.1
├── qs@5.2.0
├── node-uuid@1.4.7
├── mime-types@2.1.8 (mime-db@1.20.0)
├── combined-stream@1.0.5 (delayed-stream@1.0.0)
├── form-data@1.0.0-rc3 (async@1.5.0)
├── bl@1.0.0 (readable-stream@2.0.5)
├── hawk@3.1.2 (cryptiles@2.0.5, boom@2.10.1, sntp@1.0.9, hoek@2.16.3)
├── http-signature@1.1.0 (assert-plus@0.1.5, jsprim@1.2.2, sshpk@1.7.1)
└── har-validator@2.0.3 (commander@2.9.0, pinkie-promise@2.0.0, chalk@1.1.1, is-
my-json-valid@2.12.3)
```

Fig 4.4 Snapshot

In the above snapshot, you can see that the ‘request’ module along with the version number 2.67.0 was downloaded and installed.
Now let’s see the code which can make use of this ‘request’ command.


Sending the response from Google to console.log
Using the request module
1
2
3
Making a GET request to Google.com

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

request("http://www.google.com", function(error, response, body)
{
  console.log(body);
});
```

Fig 4.5 Code Explanation

Code Explanation:

1. We are using the ‘request’ module which was installed in the last step. This module has the necessary functions which can be used to make GET requests to websites.
2. We are making a GET Request to www.google.com and subsequently calling a function when a response is received. When a response is received the parameters(error, response, and body) will have the following values
   1. Error – In case there is any error received when using the GET request, it will be recorded here.
   2. Response- The response will have the http headers which are sent back in the response.
   3. Body- The body will contain the entire content of the response sent by Google.
3. In this, we are just writing the content received in the body parameter to the console.log file. So basically, whatever we get by going to **www.google.com** will be written to the console.log.

Here is the code for your reference
var request = require("request");

```
        request("http://www.google.com",function(error,response,body)
        {
                console.log(body);
        });
```

- The Node.js framework can be used to develop web servers using the ‘http’ module. The application can be made to listen on a particular port and send a response to the client whenever a request is made to the application.
- The ‘request’ module can be used to get information from web sites. The information would contain the entire content of the web page requested from the relevant web site.

Event Handling

Node.js is perfect for event-driven applications.

Events in Node.js

Every action on a computer is an event. Like when a connection is made or a file is opened.
Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
console.log('The file is open');
});
```

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. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

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

The EventEmitter Object

You can assign event handlers to your own events with the EventEmitter object.
In the example below we have created a function that will be executed when a "scream" event is fired.
To fire an event, use the emit() method.

MySQL Database

To be able to experiment with the code examples, you should have MySQL installed on your computer.
You can download a free MySQL database at https://www.mysql.com/downloads/.

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.
To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.
To download and install the "mysql" module, open the Command Terminal and execute the following:

```
C:\Users\Your Name>npm install mysql
```

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database:

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

Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database.

demo_db_connection.js

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

var con = mysql.createConnection({
host: "localhost",
user: "yourusername",
password: "yourpassword"
        });

con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
});
```

Save the code above in a file called "demo_db_connection.js" and run the file:

Run "demo_db_connection.js"

```
C:\Users\Your Name>node demo_db_connection.js
```

Which will give you this result:

```
Connected!
```

Now you can start querying the database using SQL statements.

Query a Database

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database.

The connection object created in the example above, has a method for querying the database:

```
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
con.query(sql, function (err, result) {
if (err) throw err;
console.log("Result: " + result);
});
});
```

The query method takes an sql statements as a parameter and returns the result.

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Create a database named "mydb":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
```

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
      console.log("Database created");
  });
});
```

Save the code above in a file called "demo_create_db.js" and run the file:

```
Run "demo_create_db.js"
C:\Users\Your Name>node demo_create_db.js
```

Which will give you this result:

```
Connected!
Database created
```

Implementation of CRUD Operation

What is CRUD?

CRUD Meaning: *CRUD* is an acronym that comes from the world of computer programming and refers to the four functions that are considered necessary to implement a persistent storage application: *create, read, update* and *delete*. Persistent storage refers to any data storage device that retains power after the device is powered off, such as a hard disk or a solid-state drive. In contrast, random access memory and internal caching are two examples of volatile memory - they contain data that will be erased when they lose power.

A relational database consists of tables with rows and columns. In a relational database, each row of a table is known as a tuple or a record. Each column of the table represents a specific attribute or field. The four CRUD functions can be called by users to perform different types of operations on selected data within the database. This could be accomplished using code or through a graphical user interface. Let's review each of the four components in-depth to fully appreciate their collective importance of facilitating database interactions.

Create

The create function allows users to create a new record in the database. In the SQL relational database application, the Create function is called INSERT. In Oracle HCM Cloud, it is called create. Remember that a record is a row and that columns are termed attributes. A user can create a new row and populate it with data that corresponds to each attribute, but only an administrator might be able to add new attributes to the table itself.

Read

The read function is similar to a search function. It allows users to search and retrieve specific records in the table and read their values. Users may be able to find desired records using keywords, or by filtering the data based on customized criteria. For example, a database of cars might enable users to type in "1996 Toyota Corolla", or it might provide options to filter search results by make, model and year.

Update

The update function is used to modify existing records that exist in the database. To fully change a record, users may have to modify information in multiple fields. For example, a restaurant that stores recipes for menu items in a database might have a table whose attributes are "dish", "cooking time", "cost" and "price". One day, the chef decides to replace an ingredient in the dish with something different. As a result, the existing record in the database must be changed and all of the attribute values changed to reflect the characteristics of the new dish. In both SQL and Oracle HCM cloud, the update function is simply called "Update".

Delete

The delete function allows users to remove records from a database that is no longer needed. Both SQL and Oracle HCM Cloud have a delete function that allows users to delete one or more records from the database. Some relational database applications may permit users to perform either a hard delete or a soft delete. A hard delete permanently removes records from the database, while a soft delete might simply update the status of a row to indicate that it has been deleted while leaving the data present and intact.

TEXT / REFERENCE BOOKS

1.Web link for Node.js Introduction - W3Schoolshttps://www.w3schools.com › nodejs › nodejs_intro

Question Bank

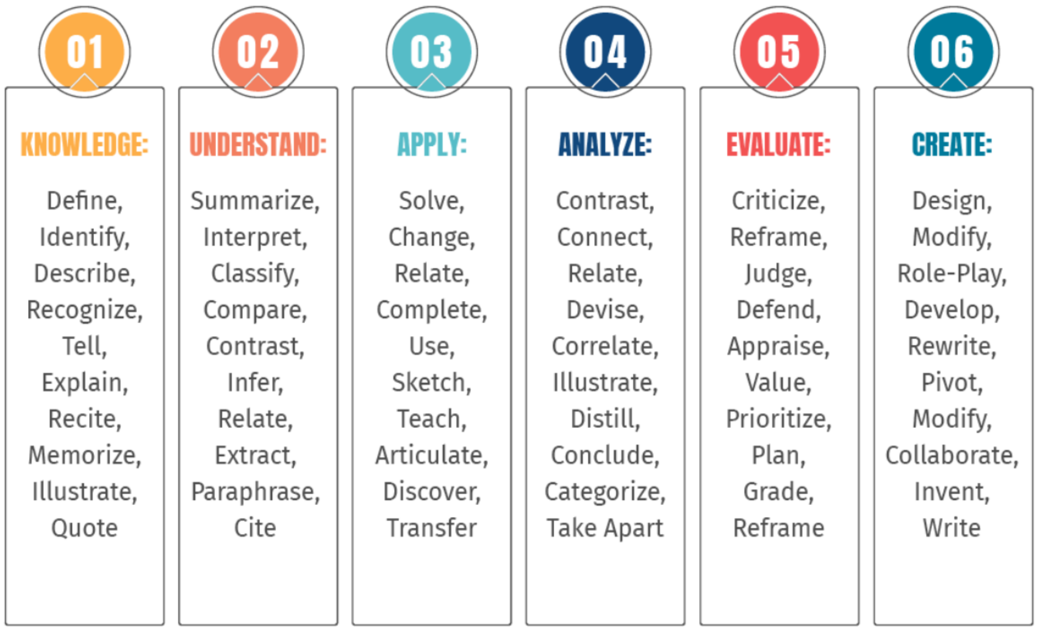
Part-A			
Q.No	**Questions (2 Marks)**	**Competence**	**BT Level**
1.	If Node.js is single-threaded, then how does it handle concurrency?	Understand	BTL2
2.	How do you create a simple server in Node.js that returns Hello World?	Knowledge	BTL1
3.	For Node.js, why Google uses V8 engine?	Understand	BTL2
4.	If we want to return the character from a specific index which method is used?	Knowledge	BTL1
5.	How to secure a Website hosted on Apache Web Server?	Knowledge	BTL1
6.	How to configure Apache log, so it captures the time taken to serve a request?	Create	BTL6
7.	How to verify httpd.conf file to ensure no configuration syntax error?	Create	BTL6
8.	How to ensure the webserver is getting started after a server reboot?	Knowledge	BTL1
9.	What is meant by event-driven programming in Node.js?	Knowledge	BTL1
10.	What the difference is between spawn and fork methods in Node.js?	Understand	BTL2
11.	Do you have any certification to boost your candidature for this Node.js role?	Understand	BTL2
12.	How to kill child processes that spawn their own child	Analyze	BTL4

	processes in Node.js?		
13.	How to use JSON Web Token (JWT) for authentication in node js?	Create	BTL6
14.	How to build a microservices architecture with node js?	Create	BTL6
15.	How to generate and verify checksum of the given string in Nodejs	Create	BTL6

Part-B

Q.No	Questions (16 Marks)	Competence	BT Level
1.	Build a Simple static file web server in Node	Understand	BTL2
2.	How to Implement CRUD operations in your application Using mysql	Create	BTL6
3.	Describe how are Event Handlers used in JavaScript?	Create	BTL6
4.	Demonstrate an event takes place and is conveyed to the Web browser and thence to the document. Describe what it means to handle an event.	Apply	BTL3
5.	What is the special keyword that acts like a variable that you use to refer to an object in its event handler?	Analyze	BTL4
6.	What is the special keyword that acts like a variable that you use to refer to an object in its event handler?	Understand	BTL2
7.	List and Explain CRUD Operations in MySQL with syntax	Apply	BTL3
8.	Explain how to execute queries with the help of a developer – Testers can begin with verifying the user interface of the application and get queries from the developer.	Evaluate	BTL5
9.	Create a Repository interface. We have created a repository interface with the name BooksRepository in the package com.javatpoint.repository. It extends the Crud Repository interface.	Create	BTL6
10.	Who is accessing the CRUD functionality? Does the system set different privileges for different users?	Create	BTL3

Bloom's Taxonomy



01	02	03	04	05	06
KNOWLEDGE:	UNDERSTAND:	APPLY:	ANALYZE:	EVALUATE:	CREATE:
Define, Identify, Describe, Recognize, Tell, Explain, Recite, Memorize, Illustrate, Quote	Summarize, Interpret, Classify, Compare, Contrast, Infer, Relate, Extract, Paraphrase, Cite	Solve, Change, Relate, Complete, Use, Sketch, Teach, Articulate, Discover, Transfer	Contrast, Connect, Relate, Devise, Correlate, Illustrate, Distill, Conclude, Categorize, Take Apart	Criticize, Reframe, Judge, Defend, Appraise, Value, Prioritize, Plan, Grade, Reframe	Design, Modify, Role-Play, Develop, Rewrite, Pivot, Modify, Collaborate, Invent, Write

UNIT – V – INTRODUCTION TO CLIENT SIDE FRAMEWORK – SITA3004

V INTRODUCTION TO CLIENT SIDE FRAMEWORK

Introduction to Angular 4.0 - Needs & Evolution – Features – Setup and Configuration – Components and Modules – Templates – Change Detection – Directives – Data Binding - Pipes – Nested Components. Template - Model Driven Forms or Reactive Forms - Custom Valuators. Introduction to ReactJS - React Components- Build a simple React component- React internals -Component inter communication- Component composition- Component styling.

AngularJS Introduction

- AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.
- AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.
- AngularJS is a JavaScript Framework
- AngularJS is a JavaScript framework written in JavaScript.

AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

AngularJS Extends HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

Example explained:

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application**.
The **ng-model** directive binds the value of the input field to the application variable **name**.
The **ng-bind** directive binds the content of the <p> element to the application variable **name**.
AngularJS Directives
As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.
The **ng-init** directive initializes AngularJS application variables.
AngularJS Example

```
<div ng-app="" ng-init="firstName='John'">

<p>The name is <span ng-bind="firstName"></span></p>

</div>
```

Needs & Evolution Angular 4

Angular 4.0.0 was released in March 2017. It was the first major version after Angular 2 and considering that, many developer focused features and enhancements were included. However, most of these changes were non-breaking.

A major version update between 1.x to 2.x was an architecture shift in building rich UI applications with Angular. Many applications were rewritten in Angular 2. Organizations, developers and projects spent a lot of time and resources in upgrading their apps from Angular 1.x to 2.x.

However Angular 4 was a totally different scenario. It had incremental changes, and improvements to the framework and applications that were written using the framework.

No major version 3.x

- **MonoRepo:** Angular 2 has been a single repository, with individual packages downloadable through npm with the @angular/package-name convention. For example @angular/core, @angular/http, @angular/router so on.

Considering this approach, it was important to have a consistent version numbering among various packages. Hence, the Angular team skipped a major version 3. It was to keep up the framework with Angular Router's version. Doing so would help avoid confusions with certain parts of the framework on version 4, while the others on version 3.

Following are some more improvements in Angular 4.x

- **Move animations out of @angular/core**

One of the problems Angular had been facing is with size of the bundles. For projects, the framework and all vendor dependencies could grow pretty big. In an effort to reduce the bundle footprint, animations were moved out. Any project migrating from Angular 2 to 4 most likely would install and import animations package explicitly.

import BrowserAnimationsModule from @angular/platform-browser/animations.

Note: reference the module in imports array of @NgModule

- **Improved View Engine**

Under the hood, AOT compilation process improved with version 4. It reduced size of the compiled component by almost half. A smaller bundle size was achieved with this change.

- **Angular Universal**

Server-side rendering capabilities were made mainstream with version 4. Angular Universal allowed server-side rendering for better search engine crawling/indexing capabilities.

- **Improved conditional statements in the template**

ng-if supported else conditions with Angular v4 onwards. The ng-if and else statements supported conditional rendering in the view. It was an enhancement to this feature.

AngularJS Modules

- An AngularJS module defines an application.
- The module is a container for the different parts of an application.
- The module is a container for the application controllers.
- Controllers always belong to a module.

Creating a Module

A module is created by using the AngularJS function angular.module

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

<script>

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

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.
Now you can add controllers, directives, filters, and more, to your AngularJS application.

Adding a Controller

Add a controller to your application, and refer to the controller with the ng-controller directive:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
   $scope.firstName = "John";
   $scope.lastName = "Doe";
});
</script>
</body>
</html>
```

Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" w3-test-directive></div>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
   return {
      template : "I was made in a directive constructor!"
   };
});
</script>
</body>
</html>
```

Modules and Controllers in Files

It is common in AngularJS applications to put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{firstName+""+lastName}}
</div>

<script src="myApp.js"></script>
<script src="myCtrl.js"></script>

</body>
</html>
```

myApp.js

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

myCtrl.js

```
app.controller("myCtrl",function($scope){
 $scope.firstName="John";
  $scope.lastName="Doe";
});
```

Functions can Pollute the Global Namespace

Global functions should be avoided in JavaScript. They can easily be overwritten or destroyed by other scripts.

AngularJS modules reduces this problem, by keeping all functions local to the module.

When to Load the Library
While it is common in HTML applications to place scripts at the end of the <body> element, it is recommended that you load the AngularJS library either in the <head> or at the start of the <body>. This is because calls to angular.module can only be compiled after the library has been loaded.
Example
```
<!DOCTYPE html>
<html>
<body>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">
{{firstName+""+lastName}}
</div>

<script>
var app=angular.module("myApp", []);
app.controller("myCtrl", function($scope){
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>
</body>
</html>
```
AngularJS Directives

- AngularJS lets you extend HTML with new attributes called **Directives**.
- AngularJS has a set of built-in directives which offers functionality to your applications.
- AngularJS also lets you define your own directives.
- AngularJS Directives
- AngularJS directives are extended HTML attributes with the prefix ng-.
- The ng-app directive initializes an AngularJS application.
- The ng-init directive initializes application data.
- The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

Example
```
<div ng-app="" ng-init="firstName='John'">

<p>Name: <input type="text" ng-model="firstName"></p>
<p>Youwrote: {{firstName}}</p>

</div>
```
The ng-app directive also tells AngularJS that the <div> element is the "owner" of the AngularJS application.
Data Binding
The {{ firstName }} expression, in the example above, is an AngularJS data binding expression.
Data binding in AngularJS binds AngularJS expressions with AngularJS data.
{{ firstName }} is bound with ng-model="firstName".

In the next example two text fields are bound together with two ng-model directives:
Example

```
<div ng-app="" ng-init="quantity=1;price=5">

Quantity: <input type="number" ng-model="quantity">
Costs:    <input type="number" ng-model="price">

Totalindollar: {{quantity*price}}

</div>
```

Repeating HTML Elements

The ng-repeat directive repeats an HTML element:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="" ng-init="names=['Jani','Hege','Kai']">
  <p>Looping with ng-repeat:</p>
  <ul>
    <li ng-repeat="x in names">
      {{ x }}
    </li>
  </ul>
</div>
</body>
</html>
```

The ng-repeat directive actually **clones HTML elements** once for each item in a collection.
The ng-repeat directive used on an array of objects.

AngularJS Data Binding

Data binding in AngularJS is the synchronization between the model and the view.

Data Model

AngularJS applications usually have a data model. The data model is a collection of data available for the application.
Example

```
var app=angular.module('myApp',[]);
app.controller('myCtrl', function($scope){
 $scope.firstname = "John";
  $scope.lastname = "Doe";
});
```

HTML View

The HTML container where the AngularJS application is displayed, is called the view.
The view has access to the model, and there are several ways of displaying model data in the view.
You can use the ng-bind directive, which will bind the innerHTML of the element to the specified model property:
Example

```
<p ng-bind="firstname"></p>
```

You can also use double braces {{ }} to display content from the model:

Example

```
<p>First name: {{firstname}}</p>
```

The ng-model Directive

Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, textarea)

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
   <input ng-model="firstname">
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstname = "John";
   $scope.lastname = "Doe";
});
</script>
<p>Use the ng-model directive on HTML controls (input, select, textarea) to bind data between the view and the data model.</p>
</body>
</html>
```

The ng-model Directive

Use the ng-model directive to bind data from the model to the view on HTML controls (input, select, textarea)

Two-way Binding

Data binding in AngularJS is the synchronization between the model and the view.

When data in the *model* changes, the *view* reflects the change, and when data in the *view* changes, the *model* is updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
   Name: <input ng-model="firstname">
   <h1>{{firstname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstname = "John";
   $scope.lastname = "Doe";
});
</script>
<p>Change the name inside the input field, and the model data will change automatically, and therefore also the header will change its value.</p>
```

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

AngularJS Controller

Applications in AngularJS are controlled by controllers. Read about controllers in the AngularJS Controllers chapter.

Because of the immediate synchronization of the model and the view, the controller can be completely separated from the view, and simply concentrate on the model data. Thanks to the data binding in AngularJS, the view will reflect any changes made in the controller.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
  <h1 ng-click="changeName()">{{firstname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "John";
  $scope.changeName = function() {
    $scope.firstname = "Nelly";
  }
});
</script>
<p>Click on the header to run the "changeName" function.</p>
<p>This example demonstrates how to use the controller to change model data.</p>
</body>
</html>
```

Angular Components Overview

Components are the main building block for Angular applications. Each component consists of:

- An HTML template that declares what renders on the page
- A Typescript class that defines behavior
- A CSS selector that defines how the component is used in a template
- Optionally, CSS styles applied to the template

This topic describes how to create and configure an Angular component.Creating a component

The best way to create a component is with the Angular CLI. You can also create a component manually.

Creating a component using the Angular CLI

To create a component using the Angular CLI:

1. From a terminal window, navigate to the directory containing your application.
2. Run the ng generate component <component-name> command, where <component-name> is the name of your new component.

By default, this command creates the following:

- A folder named after the component
- A component file, <component-name>.component.ts
- A template file, <component-name>.component.html
- A CSS file, <component-name>.component.css
- A testing specification file, <component-name>.component.spec.ts

Where <component-name> is the name of your component.

You can change how ng generate component creates new components. For more information, see ng generate component in the Angular CLI documentation.

Creating a component manually

Although the Angular CLI is the best way to create an Angular component, you can also create a component manually. This section describes how to create the core component file within an existing Angular project.

To create a new component manually:

1. Navigate to your Angular project directory.
2. Create a new file, <component-name>.component.ts.
3. At the top of the file, add the following import statement.
```
import { Component } from '@angular/core';
```
4. After the import statement, add a @Component decorator.
```
@Component({
})
```
5. Choose a CSS selector for the component.
```
@Component({
  selector: 'app-component-overview',
})
```
6. Define the HTML template that the component uses to display information. In most cases, this template is a separate HTML file.
```
@Component({
  selector: 'app-component-overview',
  templateUrl: './component-overview.component.html',
})
```
7. Select the styles for the component's template. In most cases, you define the styles for your component's template in a separate file.
```
@Component({
  selector: 'app-component-overview',
  templateUrl: './component-overview.component.html',
  styleUrls: ['./component-overview.component.css']
})
```
8. Add a class statement that includes the code for the component.
```
export class ComponentOverviewComponent {
}
```

Specifying a component's CSS selector

Every component requires a CSS *selector*. A selector instructs Angular to instantiate this component wherever it finds the corresponding tag in template HTML. For example, consider a component hello-world.component.ts that defines its selector as app-hello-world. This selector instructs Angular to instantiate this component any time the tag <app-hello-world> appears in a template.

Specify a component's selector by adding a selector statement to the @Component decorator.
```
@Component({
  selector: 'app-component-overview',
})
```

Defining a component's template

A template is a block of HTML that tells Angular how to render the component in your application. Define a template for your component in one of two ways: by referencing an external file, or directly within the component.

To define a template as an external file, add a templateUrl property to the @Component decorator.
```
@Component({
  selector: 'app-component-overview',
  templateUrl: './component-overview.component.html',
})
```

To define a template within the component, add a template property to the @Component decorator that contains the HTML you want to use.

```
@Component({
  selector: 'app-component-overview',
  template: '<h1>Hello World!</h1>',
})
```

If you want your template to span multiple lines, use backticks (`). For example:

```
@Component({
  selector: 'app-component-overview',
  template: `
    <h1>Hello World!</h1>
    <p>This template definition spans multiple lines.</p>
  `
})
```

An Angular component requires a template defined using template or templateUrl. You cannot have both statements in a component.

Declaring a component's styles

Declare component styles uses for its template in one of two ways: by referencing an external file, or directly within the component.

To declare the styles for a component in a separate file, add a styleUrls property to the @Component decorator.

```
@Component({
  selector: 'app-component-overview',
  templateUrl: './component-overview.component.html',
  styleUrls: ['./component-overview.component.css']
})
```

To declare the styles within the component, add a styles property to the @Component decorator that contains the styles you want to use.

```
@Component({
  selector: 'app-component-overview',
  template: '<h1>Hello World!</h1>',
  styles: ['h1 { font-weight: normal; }']
})
```

The styles property takes an array of strings that contain the CSS rule declarations.

Template syntax

In Angular, a *template* is a chunk of HTML. Use special syntax within a template to build on many of Angular's features.

Each Angular template in your application is a section of HTML to include as a part of the page that the browser displays. An Angular HTML template renders a view, or user interface, in the browser, just like regular HTML, but with a lot more functionality.

When you generate an Angular application with the Angular CLI, the app.component.html file is the default template containing placeholder HTML.

The template syntax guides show you how to control the UX/UI by coordinating data between the class and the template.

Empower your HTML

Extend the HTML vocabulary of your applications With special Angular syntax in your templates. For example, Angular helps you get and set DOM (Document Object Model) values dynamically with features such as built-in template functions, variables, event listening, and data binding.

Almost all HTML syntax is valid template syntax. However, because an Angular template is part of an overall webpage, and not the entire page, you don't need to include elements such as <html>, <body>, or <base>, and can focus exclusively on the part of the page you are developing. To eliminate the risk of script injection attacks, Angular does not support the <script> element in templates. Angular ignores the <script> tag and outputs a warning to the browser console.

Template statements

Template statements are methods or properties that you can use in your HTML to respond to user events. With template statements, your application can engage users through actions such as displaying dynamic content or submitting forms.

In the following example, the template statement deleteHero() appears in quotes to the right of the = symbol as in (event)="statement".

src/app/app.component.html

```
    <button (click)="deleteHero()">Delete hero</button>
```

When the user clicks the Delete hero button, Angular calls the deleteHero() method in the component class.

Use template statements with elements, components, or directives in response to events.

Syntax

Like template expressions, template statements use a language that looks like JavaScript. However, the parser for template statements differs from the parser for template expressions. In addition, the template statements parser specifically supports both basic assignment, =, and chaining expressions with semicolons ;.

The following JavaScript and template expression syntax is not allowed:

- new
- increment and decrement operators, ++ and --
- operator assignment, such as += and -=
- the bitwise operators, such as | and &
- the pipe operator

Statement context

Statements have a context—a particular part of the application to which the statement belongs.

Statements can refer only to what's in the statement context, which is typically the component instance. For example, deleteHero() of (click)="deleteHero()" is a method of the component in the following snippet.

src/app/app.component.html

```
<button (click)="deleteHero()">Delete hero</button>
```

The statement context may also refer to properties of the template's own context. In the following example, the component's event handling method, onSave() takes the template's own $event object as an argument. On the next two lines, the deleteHero() method takes a template input variable, hero, and onSubmit() takes a template reference variable, #heroForm.

src/app/app.component.html

```
    <button (click)="onSave($event)">Save</button>
    <button *ngFor="let hero of heroes" (click)="deleteHero(hero)">{{hero.name}}</button>
    <form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form>
```

In this example, the context of the $event object, hero, and #heroForm is the template.

Template context names take precedence over component context names. In the preceding deleteHero(hero), the hero is the template input variable, not the component's hero property.

Transforming Data Using Pipes

Use pipes to transform strings, currency amounts, dates, and other data for display. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe

once. For example, you would use a pipe to show a date as April 15, 1988 rather than the raw string format.
Angular provides built-in pipes for typical data transformations, including transformations for internationalization (i18n), which use locale information to format data. The following are commonly used built-in pipes for data formatting:

- DatePipe: Formats a date value according to locale rules.
- UpperCasePipe: Transforms text to all upper case.
- LowerCasePipe: Transforms text to all lower case.
- CurrencyPipe: Transforms a number to a currency string, formatted according to locale rules.
- DecimalPipe: Transforms a number into a string with a decimal point, formatted according to locale rules.
- PercentPipe: Transforms a number to a percentage string, formatted according to locale rules.

Using a pipe in a template

To apply a pipe, use the pipe operator (|) within a template expression as shown in the following code example, along with the *name* of the pipe, which is date for the built-in DatePipe. The tabs in the example show the following:

- app.component.html uses date in a separate template to display a birthday.
- hero-birthday1.component.ts uses the same pipe as part of an in-line template in a component that also sets the birthday value.

```
<p>The hero's birthday is {{ birthday | date }}</p>
```

Transforming data with parameters and chained pipes

Use optional parameters to fine-tune a pipe's output. For example, use the CurrencyPipe with a country code such as EUR as a parameter. The template expression {{ amount | currency:'EUR' }} transforms the amount to currency in euros. Follow the pipe name (currency) with a colon (:) and the parameter value ('EUR').
If the pipe accepts multiple parameters, separate the values with colons. For example, {{ amount | currency:'EUR':'Euros '}} adds the second parameter, the string literal 'Euros ', to the output string.
Use any valid template expression as a parameter, such as a string literal or a component property.
Some pipes require at least one parameter and allow more optional parameters, such as SlicePipe. For example, {{ slice:1:5 }} creates a new array or string containing a subset of the elements starting with element 1 and ending with element 5.
Example: Formatting a date
The tabs in the following example demonstrates toggling between two different formats ('shortDate' and 'fullDate'):

- The app.component.html template uses a format parameter for the DatePipe (named date) to show the date as 04/15/88.
- The hero-birthday2.component.ts component binds the pipe's format parameter to the component's format property in the template section, and adds a button for a click event bound to the component's toggleFormat() method.
- The hero-birthday2.component.ts component's toggleFormat() method toggles the component's format property between a short form ('shortDate') and a longer form ('fullDate').

```
content_copy<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>
```

Example: Applying two formats by chaining pipes
Chain pipes so that the output of one pipe becomes the input to the next.
In the following example, chained pipes first apply a format to a date value, then convert the formatted date to uppercase characters. The first tab for the src/app/app.component.html template chains DatePipe and UpperCasePipe to display the birthday as APR 15, 1988. The second tab for the src/app/app.component.html template passes the fullDate parameter to date before chaining to uppercase, which produces FRIDAY, APRIL 15, 1988.

```
The chained hero's birthday is
```

```
{{ birthday | date | uppercase}}
```

Creating pipes for custom data transformations

Create custom pipes to encapsulate transformations that are not provided with the built-in pipes. Then, use your custom pipe in template expressions, the same way you use built-in pipes—to transform input values to output values for display.

Marking a class as a pipe

To mark a class as a pipe and supply configuration metadata, apply the @Pipe decorator to the class. Use UpperCamelCase (the general convention for class names) for the pipe class name, and camelCase for the corresponding name string. Do not use hyphens in the name. For details and more examples, see Pipe names.

Use name in template expressions as you would for a built-in pipe.

Using the PipeTransform interface

Implement the PipeTransform interface in your custom pipe class to perform the transformation.

Angular invokes the transform method with the value of a binding as the first argument, and any parameters as the second argument in list form, and returns the transformed value.

Example: Transforming a value exponentially

In a game, you might want to implement a transformation that raises a value exponentially to increase a hero's power. For example, if the hero's score is 2, boosting the hero's power exponentially by 10 produces a score of 1024. Use a custom pipe for this transformation.

The following code example shows two component definitions:

- The exponential-strength.pipe.ts component defines a custom pipe named exponentialStrength with the transform method that performs the transformation. It defines an argument to the transform method (exponent) for a parameter passed to the pipe.
- The power-booster.component.ts component demonstrates how to use the pipe, specifying a value (2) and the exponent parameter (10).

```
import { Pipe, PipeTransform } from '@angular/core';
/*
 * Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage:
 *   value | exponentialStrength:exponent
 * Example:
 *   {{ 2 | exponentialStrength:10 }}
 *   formats to: 1024
*/
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent = 1): number {
    return Math.pow(value, exponent);
  }
}
```

The browser displays the following:

```
content_copyPower Booster
Superpower boost: 1024
```

Detecting changes with data binding in pipes

You use data binding with a pipe to display values and respond to user actions. If the data is a primitive input value, such as String or Number, or an object reference as input, such as Date or Array, Angular executes the pipe whenever it detects a change for the input value or reference.

For example, you could change the previous custom pipe example to use two-way data binding with ngModel to input the amount and boost factor, as shown in the following code example.
src/app/power-boost-calculator.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-power-boost-calculator',
  template: `
    <h2>Power Boost Calculator</h2>
    <label for="power-input">Normal power: </label>
    <input id="power-input" type="text" [(ngModel)]="power">
    <label for="boost-input">Boost factor: </label>
    <input id="boost-input" type="text" [(ngModel)]="factor">
    <p>
      Super Hero Power: {{power | exponentialStrength: factor}}
    </p>
  `,
  styles: ['input {margin: .5rem 0;}']
})
export class PowerBoostCalculatorComponent {
  power = 5;
  factor = 1;
}
```

The exponentialStrength pipe executes every time the user changes the "normal power" value or the "boost factor".
Angular detects each change and immediately runs the pipe. This is fine for primitive input values. However, if you change something *inside* a composite object (such as the month of a date, an element of an array, or an object property), you need to understand how change detection works, and how to use an impure pipe.

How change detection works

Angular looks for changes to data-bound values in a change detection process that runs after every DOM event: every keystroke, mouse move, timer tick, and server response. The following example, which doesn't use a pipe, demonstrates how Angular uses its default change detection strategy to monitor and update its display of every hero in the heroes array. The example tabs show the following:

- In the flying-heroes.component.html (v1) template, the *ngFor repeater displays the hero names.
- Its companion component class flying-heroes.component.ts (v1) provides heroes, adds heroes into the array, and resets the array.

```
<label for="hero-name">New hero name: </label>
<input type="text" #box
       id="hero-name"
       (keyup.enter)="addHero(box.value); box.value=""
       placeholder="hero name">
<button (click)="reset()">Reset list of heroes</button>
  <div *ngFor="let hero of heroes">
    {{hero.name}}
  </div>
```

Angular updates the display every time the user adds a hero. If the user clicks the Reset button, Angular replaces heroes with a new array of the original heroes and updates the display. If you add

the ability to remove or change a hero, Angular would detect those changes and update the display as well.
However, executing a pipe to update the display with every change would slow down your application's performance. So Angular uses a faster change-detection algorithm for executing a pipe, as described in the next section.
Detecting pure changes to primitives and object references
By default, pipes are defined as *pure* so that Angular executes the pipe only when it detects a *pure change* to the input value. A pure change is either a change to a primitive input value (such as String, Number, Boolean, or Symbol), or a changed object reference (such as Date, Array, Function, or Object).
A pure pipe must use a pure function, which is one that processes inputs and returns values without side effects. In other words, given the same input, a pure function should always return the same output.
With a pure pipe, Angular ignores changes within composite objects, such as a newly added element of an existing array, because checking a primitive value or object reference is much faster than performing a deep check for differences within objects. Angular can quickly determine if it can skip executing the pipe and updating the view.
However, a pure pipe with an array as input might not work the way you want. To demonstrate this issue, change the previous example to filter the list of heroes to just those heroes who can fly. Use the FlyingHeroesPipe in the *ngFor repeater as shown in the following code. The tabs for the example show the following:

- The template (flying-heroes.component.html (flyers)) with the new pipe.
- The FlyingHeroesPipe custom pipe implementation (flying-heroes.pipe.ts).

```
<div *ngFor="let hero of (heroes | flyingHeroes)">
  {{hero.name}}
</div>
```

The application now shows unexpected behavior: When the user adds flying heroes, none of them appear under "Heroes who fly." This happens because the code that adds a hero does so by pushing it onto the heroes array:
src/app/flying-heroes.component.ts

```
this.heroes.push(hero);
```

The change detector ignores changes to elements of an array, so the pipe doesn't run.
The reason Angular ignores the changed array element is that the *reference* to the array hasn't changed. Because the array is the same, Angular does not update the display.
One way to get the behavior you want is to change the object reference itself. Replace the array with a new array containing the newly changed elements, and then input the new array to the pipe. In the preceding example, create an array with the new hero appended, and assign that to heroes. Angular detects the change in the array reference and executes the pipe.
To summarize, if you mutate the input array, the pure pipe doesn't execute. If you *replace* the input array, the pipe executes and the display is updated.
The preceding example demonstrates changing a component's code to accommodate a pipe.
To keep your component independent of HTML templates that use pipes, you can, as an alternative, use an *impure* pipe to detect changes within composite objects such as arrays, as described in the next section.
Detecting impure changes within composite objects
To execute a custom pipe after a change *within* a composite object, such as a change to an element of an array, you need to define your pipe as impure to detect impure changes. Angular executes an impure pipe every time it detects a change with every keystroke or mouse movement.
Make a pipe impure by setting its pure flag to false:
src/app/flying-heroes.pipe.ts

```
@Pipe({
  name: 'flyingHeroesImpure',
  pure: false
})
```
The following code shows the complete implementation of FlyingHeroesImpurePipe, which extends FlyingHeroesPipe to inherit its characteristics. The example shows that you don't have to change anything else—the only difference is setting the pure flag as false in the pipe metadata.
```
@Pipe({
  name: 'flyingHeroesImpure',
  pure: false
})
export class FlyingHeroesImpurePipe extends FlyingHeroesPipe {}
```
FlyingHeroesImpurePipe is a good candidate for an impure pipe because the transform function is trivial and fast:

src/app/flying-heroes.pipe.ts (filter)
```
return allHeroes.filter(hero => hero.canFly);
```
You can derive a FlyingHeroesImpureComponent from FlyingHeroesComponent. As shown in the following code, only the pipe in the template changes.

src/app/flying-heroes-impure.component.html (excerpt)
```
<div *ngFor="let hero of (heroes | flyingHeroesImpure)">
  {{hero.name}}
</div>
```
AngularJS Forms

Forms in AngularJS provides data-binding and validation of input control

Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

Input controls provides data-binding by using the ng-model directive.
```
<input type="text" ng-model="firstname">
```
The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application.

The property firstname, can be referred to in a controller:

Example
```
<script>
var app=angular.module('myApp',[]);
app.controller('formCtrl', function($scope){
 $scope.firstname = "John";
});
</script>
```
It can also be referred to elsewhere in the application:

Example
```
<form>
 FirstName: <input type="text" ng-model="firstname">
```

```
</form>

<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

Example

Show the header if the checkbox is checked:

```
 <form>
 Check to show a header:
  <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Header</h1>
```

Radiobuttons

Bind radio buttons to your application with the ng-model directive.

Radio buttons with the same ng-model can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
 Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs">Dogs
  <input type="radio" ng-model="myVar" value="tuts">Tutorials
  <input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

The value of myVar will be either dogs, tuts, or cars

Selectbox

Bind select boxes to your application with the ng-model directive.

The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

Example

Display some text, based on the value of the selected option:

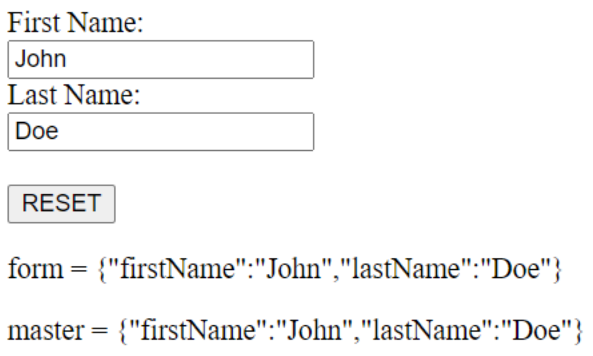
```
<form>
 Select a topic:
  <select ng-model="myVar">
    <option value="">
    <option value="dogs">Dogs
    <option value="tuts">Tutorials
    <option value="cars">Cars
  </select>
</form>
```

Application Code

```
<!DOCTYPE html>
<html lang="en">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="formCtrl">
```

```
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
    $scope.master = {firstName:"John", lastName:"Doe"};
    $scope.reset = function() {
        $scope.user = angular.copy($scope.master);
    };
    $scope.reset();
});
</script>
</body>
</html>
```

OUTPUT:



```
First Name:
John
Last Name:
Doe

RESET

form = {"firstName":"John","lastName":"Doe"}

master = {"firstName":"John","lastName":"Doe"}
```

Example Explained

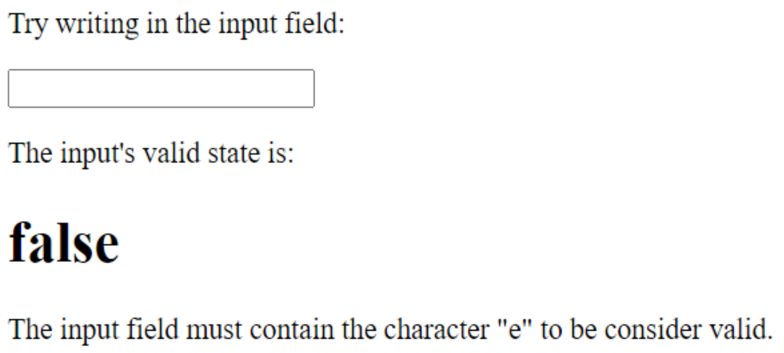
- The **ng-app** directive defines the AngularJS application.
- The **ng-controller** directive defines the application controller.
- The **ng-model** directive binds two input elements to the **user** object in the model.
- The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.
- The **reset()** method sets the **user** object equal to the **master** object.
- The **ng-click** directive invokes the **reset()** method, only if the button is clicked.
- The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

Custom Validation

To create your own validation function is a bit more tricky; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
```

```
<body ng-app="myApp">
<p>Try writing in the input field:</p>
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
<script>
var app = angular.module('myApp', []);
app.directive('myDirective', function() {
   return {
      require: 'ngModel',
      link: function(scope, element, attr, mCtrl) {
         function myValidation(value) {
            if (value.indexOf("e") > -1) {
               mCtrl.$setValidity('charE', true);
            } else {
               mCtrl.$setValidity('charE', false);
            }
            return value;
         }
         mCtrl.$parsers.push(myValidation);
      }
   };
});
</script>

<p>The input field must contain the character "e" to be consider valid.</p>
</body>
</html>
```
OUTPUT:


Try writing in the input field:

The input's valid state is:

false

The input field must contain the character "e" to be consider valid.

Example Explained:

- In HTML, the new directive will be referred to by using the attribute my-directive.
- In the JavaScript we start by adding a new directive named myDirective.
- Remember, when naming a directive, you must use a camel case name, myDirective, but when invoking it, you must use - separated name, my-directive.
- Then, return an object where you specify that we require ngModel, which is the ngModelController.
- Make a linking function which takes some arguments, where the fourth argument, mCtrl, is the ngModelController,

- Then specify a function, in this case named myValidation, which takes one argument, this argument is the value of the input element.
- Test if the value contains the letter "e", and set the validity of the model controller to either true or false.
- At last, mCtrl.$parsers.push(myValidation); will add the myValidation function to an array of other functions, which will be executed every time the input value changes.

React Introduction

What is React?

React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.React is a tool for building UI components.

How does React Work?

React creates a VIRTUAL DOM in memory.
Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM. React only changes what needs to be changed!
React finds out what changes have been made, and changes **only** what needs to be changed.

React.JS History

- Current version of React.JS is V17.0.2 (August 2021).
- Initial Release to the Public (V0.3.0) was in July 2013.
- React.JS was first used in 2011 for Facebook's Newsfeed feature.
- Facebook Software Engineer, Jordan Walke, created it.
- Current version of create-react-app is v4.0.3 (August 2021).
- create-react-app includes built tools such as webpack, Babel, and ESLint.

React Components

Components are like functions that return HTML elements.

React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

Create Your First Component

When creating a React component, the component's name *MUST* start with an upper case letter.

Class Component

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
The component also requires a render() method, this method returns HTML.
Example
Create a Class component called Car

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Function Component

Here is the same example as above, but created using a Function component instead.
A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

Example

Create a Function component called Car

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

React Internals

What is the DOM?

The DOM or Document Object Model is a tree data structure that is used by the browser. It is a representation of the UI in the form of a tree data structure. Any updates to the DOM results in re-rendering or re-painting of the UI.

What is the Virtual DOM?

The Virtual DOM is a programming concept where a clone of the UI is kept in memory. Changes to this clone do not cause a re-render directly. It is synced with the real DOM in the browser by a library such as React DOM.

Why does React use the Virtual DOM?

The DOM model in the browser is a tree data structure which makes updating and searching of nodes easy and fast. The re-rendering of the UI is a performance bottleneck. The more UI components there are, the more expensive the DOM update will be.

The Virtual DOM is a clone of the DOM. No re-rendering takes place when the Virtual DOM changes. A library like React DOM can calculate the difference between the virtual DOM and real DOM and apply the minimum set of changes.

How is an Element represented in memory in React?

When an element’s type is a string, it represents a DOM node with that tag name, and props correspond to its attributes. This is what React will render. For example:

```
{
  type: 'button',
  props: {
    className: 'button',
    children: {
      type: 'b',
      props: {
        children: 'Hello World!'
      }
    }
  }
}
```

This is just another representation of this:

```
<button class='button'>
  <b>
    Hello World!
  </b>
</button>
```

Component elements

When the type of an element is a function or a class, React calls that components render function. For example:

```
{
  type: Button,
  props: {
    color: 'blue',
    children: 'Hello World!'
  }
}
```
React will call the render() method of the Button component
The result of the above will be
```
{
  type: 'button',
  props: {
    className: 'button',
    children: {
      type: 'b',
      props: {
        children: 'Hello World!'
      }
    }
  }
}
```
Wrapping up

React does not do a full re-render everytime the state of one of the components changes. Although, the whole tree will be re-generated if the root component changes. We will look more into this in the next chapter of this series.

Component inter communication

Introduction

React is a component-based UI library. When the UI is split into small, focused components, they can do one job and do it well. But in order to build up a system into something that can accomplish an interesting task, multiple components are needed. These components often need to work in coordination together and, thus, must be able to communicate with each other. Data must flow between them. React components are composed in a hierarchy that mimics the DOM tree hierarchy that they are used to create. There are those components that are higher (parents) and those components that are lower (children) in the hierarchy. Let's take a look at the directional communication and data flow that React enables between components.

From Parent to Child with Props

The simplest data flow direction is down the hierarchy, from parent to child. React's mechanism for accomplishing this is called props. A React component is a function that receives a parameter called props. Props is a bag of data, an object that can contain any number of fields.
If a parent component wants to feed data to a child component, it simply passes it via props. Let's say that we have a BookList component that contains data for a list of books. As it iterates through the book list at render time, it wants to pass the details of each book in its list to the child Book component. It can do that through props. These props are passed to the child component as attributes in JSX:
```
  function BookList() {
   const list = [
     { title: 'A Christmas Carol', author: 'Charles Dickens' },
     { title: 'The Mansion', author: 'Henry Van Dyke' },
     // ...
    ]
    return (
      <ul>
```

```
        {list.map((book, i) => <Book title={book.title} author={book.author} key={i} />)}
      </ul>
    )
  }
```

Then the Book component can receive and use those fields as contained in the props parameter to its function:

```
function Book(props) {
  return (
    <li>
      <h2>{props.title</h2>
      <div>{props.author}</div>
    </li>
  )
}
```

Favor this simplest form of data passing whenever it makes sense.

There is a limitation here, however, because props are immutable. Data that is passed in props should never be changed. But then how does a child communicate back to its parent component? One answer is callbacks.

From Child to Parent with Callbacks

For a child to talk back to a parent (unacceptable, I know!), it must first receive a mechanism to communicate back from its parent. As we learned, parents pass data to children through props. A "special" prop of type function can be passed down to a child. At the time of a relevant event (eg, user interaction) the child can then call this function as a callback.

Let's say that a book can be edited from a BookTitle component:

```
function BookTitle(props) {
  return (
    <label>
      Title:
      <input onChange={props.onTitleChange} value={props.title} />
    </label>
  )
}
js
```

It receives a onTitleChange function in the props, sent from its parent. It binds this function to the onChange event on the <input /> field. When the input changes, it will call the onTitleChange callback, passing the change Event object. Because the parent, BookEditForm, has reference to this function, it can receive the arguments that are passed to the function:

```
import React, { useState } from 'react'
function BookEditForm(props) {
  const [title, setTitle] = useState(props.book.title)
  function handleTitleChange(evt) {
    setTitle(evt.target.value)
  }
  return (
    <form>
      <BookTitle onTitleChange={handleTitleChange} title={title} />
    </form>
  )
}
```

In this case, the parent passed handleTitleChange, and when it's called, it sets the internal state based on the value of evt.target.value -- a value that has come as a callback argument from the child component. There are some cases, however, when data sent through props might not be the best option for communicating between components. For these cases, React provides a mechanism called context.

From Parent to Child with Context

If we desire something to be globally available -- in many components and levels in the hierarchy -- props passing has the potential to be cumbersome. Think of some data that we might like to broadcast to all child components that they react to wherever they are, such as theming data. Instead of passing theme props to every component down the tree or a subtree in the hierarchy, we can define a theme context to be provided at the top and then consume it in whichever child needs it down the line.

Let's say we went back to the example of a list of books in BookList and had a parent component above that called BookPage. In that component we could provide a context for the theme:

```
const ThemeContext = React.createContext('dark')

function BookPage() {
  return (
    <ThemeContext.Provider value="light">
      <BookList />
    </ThemeContext.Provider>
  )
}
```

The ThemeContext need only be created once and, thus, is created outside the component function. It is given a default of "dark" as a fallback theme name. The context object contains a Provider function which we wrap our rendered child component in. We can specify a value to override the default theme. Here, we are saying our BookPage will always show the "light" theme. Note also that BookList does not receive any theme props. We can leave its implementation as-is. But let's say that we want our Book component to respond to theming. We could adjust it to something like:

```
import React, { useContext } from 'react'
function Book(props) {
  const theme = useContext(ThemeContext)
  const styles = {
    dark: { background: 'black', color: 'white' },
    light: { background: 'white', color: 'black' }
  }
  return (
    <li style={styles[theme]}>
      <h2>{props.title</h2>
      <div>{props.author}</div>
   </li>
  )
}
```

Book needs to have access to the ThemeContext object created next to BookPage, and that context object is fed to the useContext hook. From there, we create a simple styles map and select the appropriate styling based on the value of theme. Based on the value of theme provided in BookPage, we'll have a black color on white background styling shown here. The thing that's special about context is that theme did not come from props but rather was simply available because a parent component provided it to any and all children which used it. As with most global code patterns, use context sparingly. It creates coupling between components that can lead to less-reusable code and relationships or between components that are less clear. If the context value was a callback function, we could see this being used for child to parent communication as well.

Sideways With Non-React Options

React provides patterns for communicating up and down the component hierarchy. Since all components exist in this hierarchy, this is natural and effective. What if, however, we want to communicate "sideways" where data doesn't come from a parent or back up from a child? React can accomplish this by a combination of passing data up the hierarchy, then back down taking a different path to sibling components. But if we really want the data to not flow through a parent or child relationship, we have to step outside of React. When we step outside React, the data is not going to come from props, context, or React-passed callbacks. It's going to come from vanilla JavaScript-type sources such as a module we import or a JavaScript object we observe. There are some libraries that have formalized patterns for working with data flow outside of React but that work well with React. Redux is a common example of this, where a single state tree is maintained outside the component hierarchy but which is designed to connect easily to your components, allowing sideways-access to data. If parent-child communication doesn't make sense for some reason, keep this non-React set of options in mind.

Component Composition in React

Introduction

When building an application using React, it is usually the case that developers will want to re-use component code. Developers new to React will often, instinctively, try to achieve this using inheritance. However, it is nearly always better to use React's composition model. A good example of components that often need to re-use component code is tabs which show different content when selected. This guide will start with a set of tabs that re-use no code whatsoever and build specialized components, container components, and a combination of both to achieve re-use through composition.

Tabs

A simple way to build a component containing tabs is to do something like this:

```
<ul className="nav nav-tabs">
  <li className="nav-item">
    <button
      className={`nav-link${selectedTabIndex === 0 ? " active" : ""}`}
      onClick={() => setSelectedTabIndex(0)}
    >
      {"Tab 1"}
    </button>
  </li>
  <li className="nav-item">
    <button
      className={`nav-link${selectedTabIndex === 1 ? " active" : ""}`}
      onClick={() => setSelectedTabIndex(1)}
    >
      {"Tab 2"}
    </button>
  </li>
</ul>
<div className="tab-content">
  {selectedTabIndex === 0 && (
    <>
      <h2>{"Tab 1"}</h2>
      <p>{"Some content for the first tab"}</p>
    </>)}
  {selectedTabIndex === 1 && (
    <>
      <h2>{"Tab 2"}</h2>
      <p>{"Some content for the second tab"}</p>
    </>)}
</div>
```

The ul element contains the UI for each tab separately and the content is displayed in the div element at the bottom. This is OK; however, when new tabs are required, code will need to be copy/pasted and if any of the elements that are common across tabs or content need to be changed then they will have to be changed for every tab, which is not ideal. To make this easier, the tab components can be composed using one or a combination of both of the following patterns.

Specialized Components

A specialized component is a generic component that accepts props that are used to render a specialized version. A specialized component for the tab UI looks like this:

```
const TabSpecialized = props => (
  <li className="nav-item">
    <button
      className={`nav-link${props.selected ? " active" : ""}`}
      onClick={props.onSelect}>
      {props.text}
    </button>
  </li>);
```

This component takes props of text, selected, and onSelect to define the text to show on the tab: a boolean indicating whether the tab is selected and a function to call when the tab is clicked. This tab component can then be consumed like this:

```
<TabSpecialized
  text="Tab 3"
  selected={selectedTabIndex === 2}
  onSelect={() => setSelectedTabIndex(2)}
/>
```

Therefore, if the display of the tab needs to change, this change is made only once in the SpecializedTab component which will then be reflected in all of the tabs.

Similarly, a specialized component can be used for the tab content:

```
const TabContentSpecialized = props => (
  <>
    <h2>{props.header}</h2>
    <p>{props.paragraph}</p>
  </>);
```

This component takes props of header for the header to display and a paragraph of text. The component is consumed like this:

```
<TabContentSpecialized
  header="Tab 3"
  paragraph="Some content for the third tab"
/>
```

This will render the header inside an h2 element, followed by the paragraph.

As with the tab component, if any changes need to be made to how either the header or paragraph are displayed then this change need only be made in the TabContentSpecialized component.

Container Components

In many cases, a specialized component will suffice, however, a drawback to using this pattern with these tab components is that it assumes that all components will be displayed in the same format and, indeed, that it is known how the content for every tab in the application will be displayed. For the actual tab components, this may well be correct. However, for the content, it probably will not be. The specialized content component takes props of strings for a header and a paragraph; if anything else needs to be rendered then either optional props can be added to the component to cover all cases that are known, or a container component can be used. Adding props for all cases, again, assumes that all cases are known and can also make the component code very complex; therefore a container component is the better option. All React components have a special children prop so that consumers can pass components directly by nesting them inside the jsx. This prop can be used by a tab content component to accept the actual content without needing to know anything else about it.
A version of the tab content component as a container looks like this:

```
const TabContentContainer = props => (
  <div className="tab-content">
    {props.children}
  <div />);
```

This component simply renders the child components inside a div which is used to ensure that content is shown with a standard style and is consumed like this:

```
<TabContentContainer>
  <div className="tab4-content">
    <img src={logo} className="App-logo" alt="logo" />
    <p>This tab can contain anything.</p>
  </div>
</TabContentContainer>
```

Everything nested inside the TabContentContainer element is passed as the children prop and is rendered by the content component meaning that, as the text says, the tab can now contain anything.

Combining Specialized and Container Components

In the case of the tab components, a combination of both specialization and containers is probably the best pattern to use. This way, each tab and content can show text for a header in the same style but also can use the children prop to define specific content.

The code for a combined tab component looks like this:

```
const Tab = props => (
  <li className="nav-item">
    <button
      className={`nav-link${props.selected ? " active" : ""}`}
      onClick={props.onSelect}
    >
      {props.text}
      {props.children}
    </button>
  </li>);
```

In addition to the props for the specialized tab component, there is now a children prop which allows the tab to display any extra content. So, using this component to add an image to the tab can be done like this:

```
<Tab
  text="Tab 4"
  selected={selectedTabIndex === 3}
  onSelect={() => setSelectedTabIndex(3)}
>
  <img src={logo} className="App-logo" alt="logo" />
</Tab>
```

The content component looks like this:

```
const TabContent = props => (
  <>
    <h2>{props.header}</h2>
    {props.children}
  </>);
```

This component has added a header prop to the TabContentContainer component above and can be consumed like this:

```
<TabContent header="Tab 4">
  ...
</TabContent>
```

The header will then be displayed followed by the children.

Component styling

Pass a string as the className prop:

```
render() {
  return <span className="menu navigation-menu">Menu</span>
}
```

It is common for CSS classes to depend on the component props or state:

```
render() {
  let className = 'menu';
  if (this.props.isActive) {
    className += ' menu-active';
  }
  return <span className={className}>Menu</span>
}
```

“CSS-in-JS” refers to a pattern where CSS is composed using JavaScript instead of defined in external files.

Note that this functionality is not a part of React, but provided by third-party libraries. React does not have an opinion about how styles are defined; if in doubt, a good starting point is to define your styles in a separate *.css file as usual and refer to them using className.

TEXT / REFERENCE BOOKS

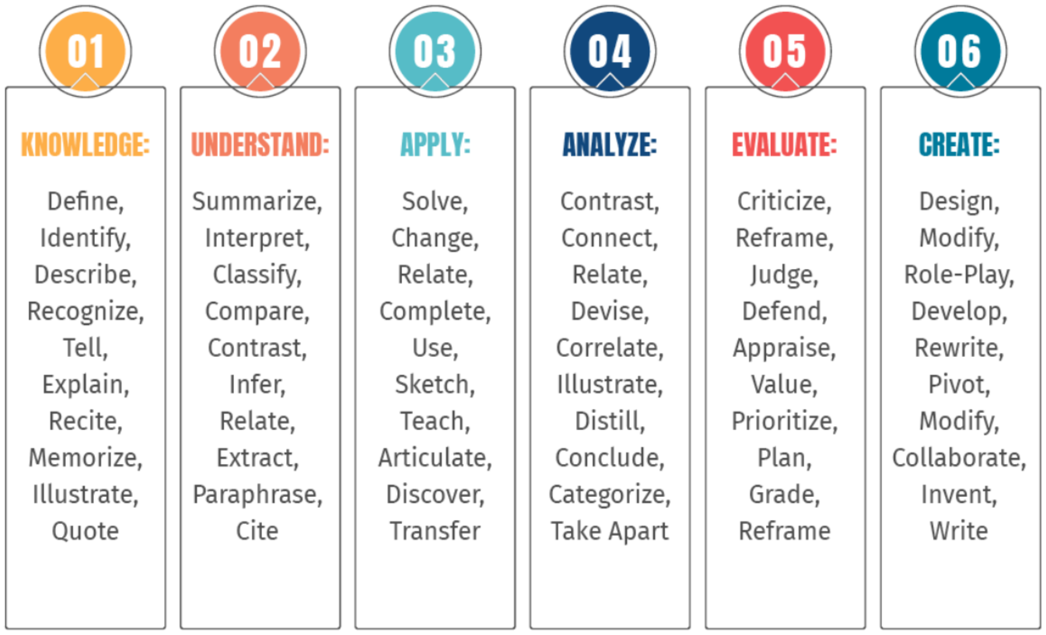
1.Web link for AngularJS Tutorial - W3Schoolshttps://www.w3schools.com › angular

Question Bank

Part-A			
Q.No	**Questions (2 Marks)**	**Competence**	**BT Level**
1.	Explain the differences between Angular and jQuery	Understand	BTL 2
2.	Identify what is the Ahead of Time Compilation?	Analyze	BTL 4
3.	List out the difference between Angular Component andDirective	Remember	BTL 1
4.	Categorize different phases of the AngularJS Scopelifecycle.	Create	BTL 5
5.	Determine the AngularJS components that can be injected asdependency	Create	BTL 5
6.	Brief the use of $routeProvider	Create	BTL 5
7.	Distinguish between AngularJS and backbone.js	Understand	BTL 2
8.	Differentiate between a link and compile in Angular.js?	Analyze	BTL 4

9.	Brief the linking function and its type	Create	BTL 5
10.	Explain directives and their types	Analyze	BTL 4
11.	Define $rootscope and how do we use it?	Apply	BTL 3
12.	Organize the step to initialize a select box with options onpage load.	Analyze	BTL 4
13.	Define Dependency Injection.	Remember	BTL 1
14.	Create a program for to bootstrap process in Angular	Create	BTL 6
15.	Compare and Contrast prop() and attr().	Analyze	BTL 4
Part-B			
Q.No	**Questions (16 Marks)**	**Competence**	**BT Level**
1.	Illustrate the Sample Angular Powered View with example	Analyze	BTL 4
2.	Develop a simple code fragment using NgFor. Create a file called mock-heroes.ts in the user defined folder and Define a HEROES constant as an array of ten heroes and export it and list the name of the heros.	Evaluate	BTL 5
3.	List and Explain the Different data binding methods inangularjs with syntax.	Apply	BTL 3
4.	Demonstrate passing data to directrive type scope withsample coding fragment	Apply	BTL 3
5.	Develop a code for printing the fees using filters	Create	BTL 6
6.	Explain function scope and service in AngularJS	Understand	BTL 2
7.	Write a program to hide an HTML tag just by one buttonclick in angular	Evaluate	BTL 5
8.	Discuss the jQuery function used to provide effects.	Analyze	BTL 4
9.	List the different slide effects available in jQuery.	Remember	BTL 1
10.	Illustrate the various CSS classes used to manipulate inHTML using jQuery	Apply	BTL 3

Bloom's Taxonomy



01	02	03	04	05	06
KNOWLEDGE:	UNDERSTAND:	APPLY:	ANALYZE:	EVALUATE:	CREATE:
Define, Identify, Describe, Recognize, Tell, Explain, Recite, Memorize, Illustrate, Quote	Summarize, Interpret, Classify, Compare, Contrast, Infer, Relate, Extract, Paraphrase, Cite	Solve, Change, Relate, Complete, Use, Sketch, Teach, Articulate, Discover, Transfer	Contrast, Connect, Relate, Devise, Correlate, Illustrate, Distill, Conclude, Categorize, Take Apart	Criticize, Reframe, Judge, Defend, Appraise, Value, Prioritize, Plan, Grade, Reframe	Design, Modify, Role-Play, Develop, Rewrite, Pivot, Modify, Collaborate, Invent, Write