CBCS SCHEME

USN | | | | | | | | | | |

22MCA24

Second Semester MCA Degree Examination, June/July 2024
Web Technologies

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Briefly explain the following : i) Web browsers ii) URL iii) MIME iv) Web servers	10	L2	CO6
	b.	Discuss in detail different phases of HTTP.	10	L2	CO6
		OR			
Q.2	a.	List and explain different types of lists in XHTML with an example.	10	L2	CO6
	b.	Create a Registration form to accept name, gender, date of birth, qualification, address and provide Reset and Submit buttons.	10	L2	CO4
		Module – 2			
Q.3	a.	Discuss various selectors of CSS. Explain its usage with an example.	10	L3	CO7
	b.	Explain box model in detail with an example.	10	L3	CO7
		OR			
Q.4	a.	Give example for pattern patching methods of strings in Java script.	10	L2	CO2
	b.	Write a Java script program to accept a number and display the reverse of a given number.	10	L3	CO4
		Module – 3			
Q.5	a.	List out the various bootstrap button classes for different styles of button. Explain their use with code snippet.	10	L3	CO2
	b.	Explain how to create custom forms in Bootstrap with an example.	10	L2	CO6
		OR			
Q.6	a.	With suitable code snippets explain Grid system of Bootstrap.	10	L3	CO6
	b.	Describe different containers in Bootstrap with example.	10	L3	CO6
		Module – 4			
Q.7	a.	Demonstrate the use of different methods for adding and removing classes with suitable code snippets.	10	L2	CO1
	b.	Explain the following : i) bind( ) ii) eventTypeName( ) iii) One( ) iv) Unbind( )	10	L3	CO7
		OR			
Q.8	a.	Discuss the commands for showing and hiding elements with an example.	10	L3	CO5
	b.	Explain how to use child, container and attribute selectors in JQuery.	10	L2	CO5
		Module – 5			
Q.9	a.	Discuss different types of expression in Angular JS with an example.	10	L2	CO5
	b.	What is $scope and explain how to use controllers with an example.	10	L2	CO5
		OR			
Q.10	a.	Explain the following directives with example: i) ng_app ii) ng_model iii) ng_init iv) ng_repeat	10	L2	CO5
	b.	What is filter? Explain uppercase, lower case, order by and currency with an example.	10	L2	CO5

* * * * *

Q1a) Briefly explain the following i) Web browsers ii) URL iii) MIME

i) Web browsers

Web server operations:

- All the communications between a web client and a web server use the HTTP
- When a web server begins execution, it informs the OS under which it is running & it runs as a background process
- A web client or browser, opens a network connection to a web server, sends information requests and possibly data to the server, receives information from the server and closes the connection.
- The primary task of web server is to monitor a communication port on host machine, accept HTTP commands through that port and perform the operations specified by the commands.
- When the URL is received, it is translated into either a filename or a program name

General characteristics of web server:

- The file structure of a web server has two separate directories
- The root of one of these is called document root which stores web documents
- The root of the other directory is called the server root which stores server and its support software's
- The files stored directly in the document root are those available to clients through top level URLs
- The secondary areas from which documents can be served are called virtual document trees.
- Many servers can support more than one site on a computer, potentially reducing the cost of each site and making their maintenance more convenient. Such secondary hosts are called virtual hosts.
- Some servers can serve documents that are in the document root of other machines on the web; in this case they are called as proxy servers

ii) URL

- Uniform Resource Locators (URLs) are used to identify different kinds of resources on Internet.
- If the web browser wants some document from web server, just giving domain name is not sufficient because domain name can only be used for locating the server.
- It does not have information about which document client needs. Therefore, URL should be provided.
- The general format of URL is: scheme: object-address
  Example: http: www.vtu.ac.in/results.php
- The scheme indicates protocols being used. (http, ftp, telnet...)
- In case of http, the full form of the object address of a URL is as follows:
  //fully-qualified-domain-name/path-to-document
- URLs can never have embedded spaces
- It cannot use special characters like semicolons, ampersands and colons
- The path to the document for http protocol is a sequence of directory names and a filename, all separated by whatever special character the OS uses. (Forward or

backward slashes)

- The path in a URL can differ from a path to a file because a URL need not include all
- directories on the path
- A path that includes all directories along the way is called a complete path.
  Example: http://www.gumboco.com/files/f99/storefront.html
- In most cases, the path to the document is relative to some base path that is specified in the configuration files of the server. Such paths are called partial paths.
- Example: http://www.gumboco.com/storefront.htm

iii) MIME

- MIME stands for Multipurpose Internet Mail Extension.
- The server system apart from sending the requested document, it will also send MIME information.
- The MIME information is used by web browser for rendering the document properly.
- The format of MIME is: type/subtype
- Example: text/html , text/doc , image/jpeg , video/mpeg
- When the type is either text or image, the browser renders the document without any problem
- However, if the type is video or audio, it cannot render the document
- It has to take the help of other software like media player, win amp etc.,
- These softwares are called as helper applications or plugins
- These non-textual information are known as HYPER MEDIA
- Experimental document types are used when user wants to create a customized information & make it available in the internet
- The format of experimental document type is: type/x-subtype
  Example: database/x-xbase , video/x-msvideo
- Along with creating customized information, the user should also create helper applications.
- This helper application will be used for rendering the document by browser.
- The list of MIME specifications is stored in configuration file of web server.

Q1b) Explain request phase and response phase of HTTP.

Request Phase:

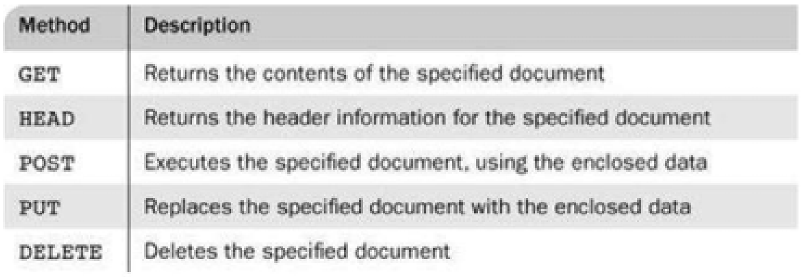
The general form of an HTTP request is as follows:

1. HTTP method Domain part of the URL HTTP version
2. Header fields
3. Blank line
4. Message body

The following is an example of the first line of an HTTP request:

GET /storefront.html HTTP/1.1

Table 1.1 HTTP request methods



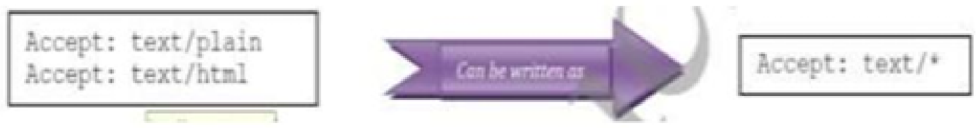
Method	Description
`GET`	Returns the contents of the specified document
`HEAD`	Returns the header information for the specified document
`POST`	Executes the specified document, using the enclosed data
`PUT`	Replaces the specified document with the enclosed data
`DELETE`	Deletes the specified document

The format of a header field is the field name followed by a colon and the value of the field.

There are four categories of header fields:

1. **General**: For general information, such as the date
2. **Request**: Included in request headers
3. **Response**: For response headers
4. **Entity**: Used in both request and response headers

A wildcard character, the asterisk (*), can be used to specify that part of a MIME type can be anything.


Accept: text/plain
Accept: text/html

Accept: text/*

The Host: host name request field gives the name of the host. The Host field is required for HTTP 1.1. The If-Modified-Since: date request field specifies that the requested file should be sent only if it has been modified since the given date. If the request has a body, the length of that body must be given with a Content-length field. The header of a request must be followed by a blank line, which is used to separate the header from the body of the request.

The Response Phase:

The general form of an HTTP response is as follows:

1. Status line
2. Response header fields
3. Blank line
4. Response body

The status line includes the HTTP version used, a three-digit status code for the response, and a short textual explanation of the status code.

For example, most responses begin with the following:

HTTP/1.1 200 OK

The status codes begin with 1, 2, 3, 4, or 5. The general meanings of the five categories specified by these first digits are shown in Table 1.2.

Table 1.2 First digits of HTTP status codes

First Digit	Category
1	Informational
2	Success
3	Redirection
4	Client error
5	Server error

One of the more common status codes is one user never want to see: 404 Not Found, which means the requested file could not be found.

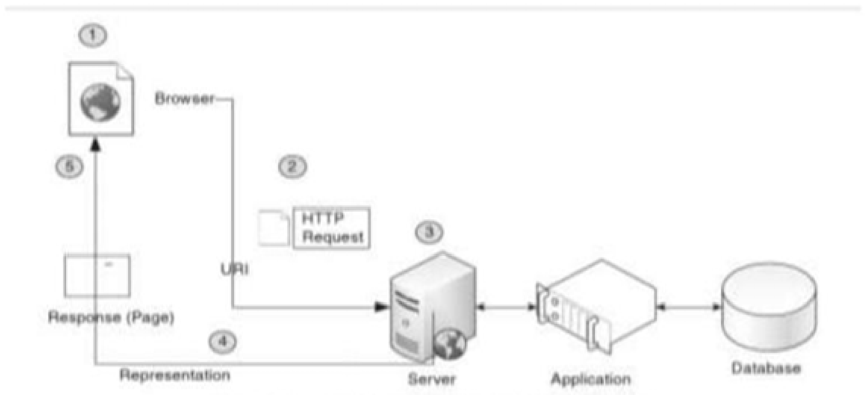


Fig. 1.7 *A simplified web communication scenario*

Q2a) List and Explain different types of lists in XHTML with an example

1) Unordered List
The <ul> tag, which is a block tag, creates an unordered list. Each item in a list is specified with an <li> tag (li is an acronym for list item). Any tags can appear in a list item, including nested lists. When displayed, each list item is implicitly preceded by a bullet

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- unordered.html
     An example to illustrate an unordered list
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Unordered list </title>
  </head>
  <body>
    <h3> Some Common Single-Engine Aircraft </h3>
    <ul>
      <li> Cessna Skyhawk </li>
      <li> Beechcraft Bonanza </li>
      <li> Piper Cherokee </li>
    </ul>
  </body>
</html>
```

2) Ordered List

Ordered lists are lists in which the order of items is important. This ordered-ness of a list is shown in the display of the list by the implicit attachment of a sequential value to the beginning of each item. The default sequential values are Arabic numerals, beginning with 1.
An ordered list is created within the block tag <ol>. The items are specified and displayed just as are those in unordered lists, except that the items in an ordered list are preceded by sequential values instead of bullets.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- ordered.html
     An example to illustrate an ordered list
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Ordered list </title>
  </head>
  <body>
    <h3> Cessna 210 Engine Starting Instructions </h3>
    <ol>
      <li> Set mixture to rich </li>
      <li> Set propeller to high RPM </li>
      <li> Set ignition switch to "BOTH" </li>
      <li> Set auxiliary fuel pump switch to "LOW PRIME" </li>
      <li> When fuel pressure reaches 2 to 2.5 PSI, push
           starter button
      </li>
    </ol>
  </body>
</html>
```

3) Definition List
As the name implies, definition lists are used to specify lists of terms and their definitions, as in glossaries. A definition list is given as the content of a <dl> tag, which is
a block tag. Each term to be defined in the definition list is given as the content of a
<dt>tag. The definitions themselves are specified as the content of <dd> tags. The defined terms of a definition list are usually displayed in the left margin; the definitions are usually shown indented on the line or lines following the term.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- definition.html
     An example to illustrate definition lists
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Definition lists </title>
  </head>
  <body>
    <h3> Single-Engine Cessna Airplanes </h3>
    <dl>
      <dt> 152 </dt>
      <dd> Two-place trainer </dd>
      <dt> 172 </dt>
      <dd> Smaller four-place airplane </dd>
      <dt> 182 </dt>
      <dd> Larger four-place airplane </dd>
      <dt> 210 </dt>
      <dd> Six-place airplane - high performance </dd>
    </dl>
  </body>
</html>
```

Q2b) Create a Registration form to accept name, gender, date of birth, qualification, address and provide Reset and Submit buttons.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>New document</title>
</head>
<body>
<form method=" " action=" "><br/>
<font color="blue"/>
        <h1 align="center">Registration Form</h1>
        <table align='center'>
                <tr>
                        <td>Enter Name</td>
                        <td><input type="text" name="name1"/></td>
                </tr>
                <tr>
                        <td>Enter Gender</td>
                        <td><input type="radio" name="gender"/>Male
                        < input type="radio" name="gender"/>Female</</td>
                </tr>
                <tr>
                        <td>Enter Date of Birth</td>
                        <td><input type="date" name="dob"/></td>
                </tr>
                <tr>
                        <td>Enter your highest qualification</td>
                        <td><input type="text" name="qualification"/></td>
                </tr>
                <tr>
                        <td>Enter Address</td>
                        <td><input type="text" name="add"/></td>
                </tr>


                <tr><td><input type="reset" value="CLEAR"/></td>
                        <td><input type="submit" value="INSERT</td>

                </tr>
                </table>
</form>
</body>
</html>
```

Q3a)Discuss various selectors of CSS Explain its usage with an example

1) Simple Selector Forms:

In case of simple selector, a tag is used. If the properties of the tag are changed, then it reflects at all the places when used in the program. The selector can be any tag. If the new properties for a tag are not mentioned within the rule list, then the browser uses default behaviour of a tag.

Eg:

```
h1 { font-size : 24pt; }

h2, h3{ font-size : 20pt; }

body b em { font-size : 14pt; }
```

Only applies to the content of ‘em’ elements that are descendent of bold element in the body of the document. This is a contextual selector

2) Class Selectors:

Class selectors are used to allow different occurrences of the same tag to use different style specifications.

Eg

```
<head>

<style type = "text/css">

p.one { font-family: 'Lucida Handwriting'; font-size: 25pt; color: Red; }

p.two{ font-family: 'Monotype Corsiva'; font-size: 50pt; color: green; }

</style>

</head>

<body>

<p class = "one">Web Technology</p>

<p class = "two">Web Technology</p>

</body>
```

3) Generic Selectors:

Sometimes it is convenient to have a class of Style specification that applies to the content of more than one kind of tag. This is done by using a generic class, which is defined without a tag name in its name. In place of the tag name, you use the name of the generic class, which must begin with a period.

Eg

```
<head>

<style type = "text/css">

.sale{ font-family: 'Monotype Corsiva'; color: green; }

</style>

</head>

<body>

<p class = "sale">Weekend Sale</p>

<h1 class = "sale">Weekend Sale</h1>

<h6 class = "sale"> Weekend Sale</h6>

</body>
```

4) id Selectors:

An id selector allows the application of a style to one specific element.

Eg:

```
<head>

<style type = "text/css">

#one { font-family: 'Lucida Handwriting'; font-size: 25pt; color: Red; }

#two { font-family: 'Monotype Corsiva'; font-size: 50pt; color: green; }

</style>

</head>

<body>
```

```
<p id = "one">Web Technology</p>

<p id = "two">Web Technology</p>

</body>
```

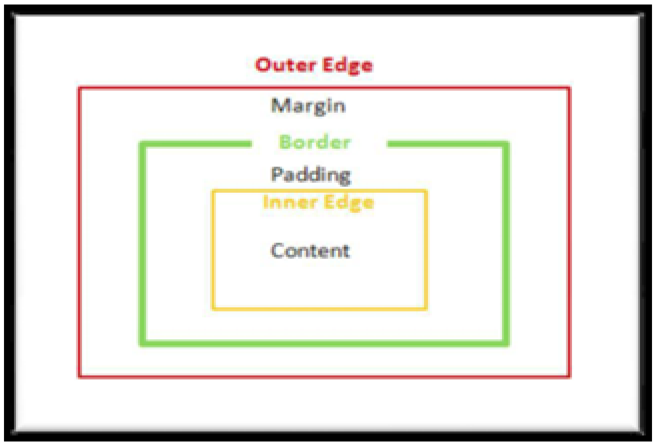
5) Universal Selectors:

The universal selector, denoted by an asterisk (*), applies its style to all elements in a document.

```
<head>

<style type = "text/css">

 { font-family: 'Lucida Handwriting'; font-size: 25pt; color: Red; }

</style>

</head>

<body>

<p>Web Technology</p>

<p>Web Technology</p>

</body>
```

Q3b) Explain box model in detail with an example

- On a given web page or a document, all the elements can have borders.
- The borders have various styles, color and width.
- The amount of space between the content of the element and its border is known as *padding.*
- The space between border and adjacent element is known as *margin.*



Borders:

1) **Border-style property** controls whether the elements content has a border, as well as the style of the border
   It can be dotted, dashed, double
   The styles of one of the four sides of an element can be set with
   - Border-top-style
   - Border-bottom-style
   - Border-left-style
   - Border-right-style

2) **Border-width** is used to specify the thickness of a border
   It can be thin, medium, thick or any length value
   The width of each of the four borders of an element specified with:
   - Border-top-width
   - Border-bottom-width
   - Border-left-width
   - Border-right-width

3) **Border-color** control color of a border
   The width of each of the four borders of an element specified with:
   - Border-top-color
   - Border-bottom-color
   - Border-left-color
   - Border-right-color

Margins and Padding:

The margin properties are named margin, which applies to all four sides of an element: margin-left, margin-right, margin-top, and margin-bottom.

The padding properties are named padding, which applies to all four sides: padding-left, padding-right, padding-top, and padding-bottom.

Q4a) Give example for pattern matching methods of strings in JavaScript

- JavaScript has powerful pattern-matching capabilities based on regular expressions.
- There are two approaches to pattern matching in JavaScript: one that is based on the `RegExp` object and one that is based on methods of the `String` object.
- The simplest pattern-matching method is `search`, which takes a pattern as a parameter.
- The `search` method returns the position in the `String` object (through which it is called) at which the pattern matched.
- If there is no match, `search` returns `-1`.
- Most characters are normal, which means that, in a pattern, they match themselves.
- The position of the first character in the string is 0.
- As an example, the following statements

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position >= 0)
    document.write("'bits' appears in position", position,
                   "<br />");
else
    document.write("'bits' does not appear in str <br />");
```

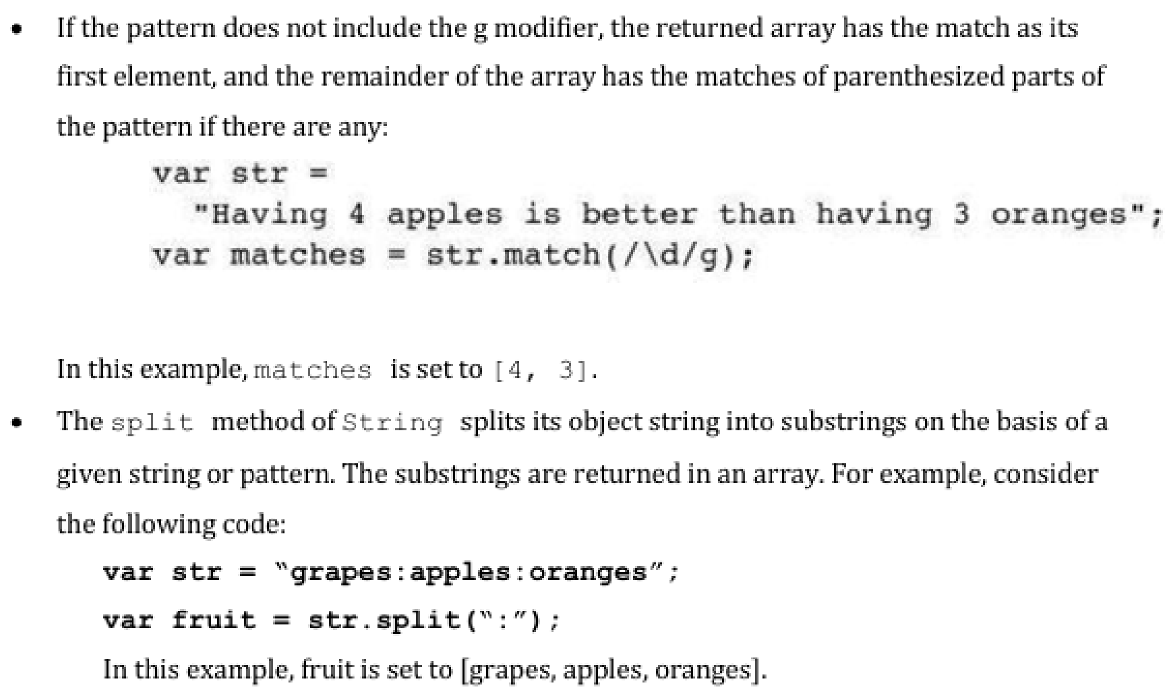
produce the following output:

```
'bits' appears in position 3
```

- The `replace` method is used to replace substrings of the `String` object that match the given pattern.
- The `replace` method takes two parameters: the pattern and the replacement string.
- The `g` modifier can be attached to the pattern if the replacement is to be global in the string, in which case the replacement is done for every match in the string.
- The matched substrings of the string are made available through the predefined variables `$1`, `$2`, and so on. For example, consider the following statements:

```
var str = "Fred, Freddie, and Frederica were siblings";
str.replace(/Fre/g, "Boy");
```

- In this example, str is set to "Boyd, Boyddie, and Boyderica were siblings", and $1, $2, and $3 are all set to "Fre".
- The match method is the most general of the String pattern-matching methods.
- The match method takes a single parameter: a pattern. It returns an array of the results of the pattern-matching operation.
- If the pattern has the g modifier, the returned array has all of the substrings of the string that matched.



- If the pattern does not include the g modifier, the returned array has the match as its first element, and the remainder of the array has the matches of parenthesized parts of the pattern if there are any:

```
var str =
  "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);
```

  In this example, `matches` is set to `[4, 3]`.
- The `split` method of `String` splits its object string into substrings on the basis of a given string or pattern. The substrings are returned in an array. For example, consider the following code:

```
var str = "grapes:apples:oranges";
var fruit = str.split(":");
```

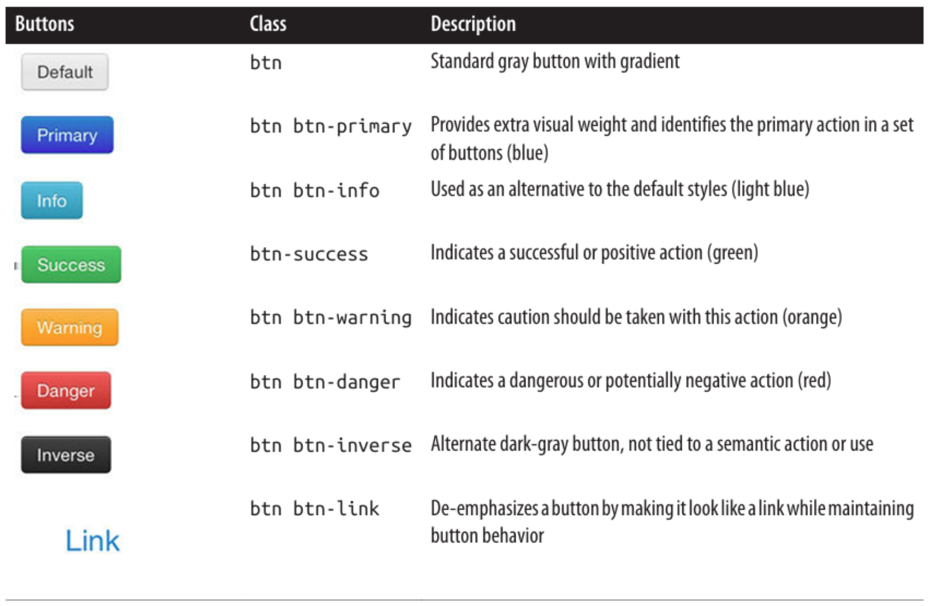
  In this example, fruit is set to [grapes, apples, oranges].

Q4b) Write a JavaScript program to accept a number and display the reserve of a given number

```
<!DOCTYPE HTML>
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
<script type ="text/javascript">
function rev_num()
{
        var num = prompt("Enter the number to be reveresed :", " ");
        var n= num;
        var rev = 0, rem;
        while (n>0)
        {
                rem = n % 10;
                rev = rev * 10 + rem ;
                n = Math.floor(n/10);
        }
        document.write("The given number is : " +num+ " <br/> The reversed number is : "
+rev+ "\n");
}
</script>
</head>
<body onload = "rev_num();">
</body>
</html>
```

Q5a) List out the various bootstrap button classes for different styles of button. Explain their use with code snippet.

Anything that is given a class of .btn will inherit the default look of a gray button with rounded corners. However, you can add color to the buttons by adding extra classes



Buttons	Class	Description
Default	btn	Standard gray button with gradient
Primary	btn btn-primary	Provides extra visual weight and identifies the primary action in a set of buttons (blue)
Info	btn btn-info	Used as an alternative to the default styles (light blue)
Success	btn-success	Indicates a successful or positive action (green)
Warning	btn btn-warning	Indicates caution should be taken with this action (orange)
Danger	btn btn-danger	Indicates a dangerous or potentially negative action (red)
Inverse	btn btn-inverse	Alternate dark-gray button, not tied to a semantic action or use
Link	btn btn-link	De-emphasizes a button by making it look like a link while maintaining button behavior

```
<button type="button" class="btn">Base class</button>

<button type="button" class="btn btn-primary">Primary</button>

<button type="button" class="btn btn-secondary">Secondary</button>
```

```
<button type="button" class="btn btn-success">Success</button>

<button type="button" class="btn btn-danger">Danger</button>

<button type="button" class="btn btn-warning">Warning</button>

<button type="button" class="btn btn-info">Info</button>

<button type="button" class="btn btn-light">Light</button>

<button type="button" class="btn btn-dark">Dark</button>



<input class="btn btn-primary" type="submit" value="Submit">
```

Button Sizes

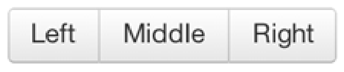
If you need larger or smaller buttons, simply add .btn-large, .btn-small, or .btn mini to links or buttons

```
<p>
  <button class="btn btn-large btn-primary" type="button">Large button</button>
  <button class="btn btn-large" type="button">Large button</button>
</p>
<p>
  <button class="btn btn-primary" type="button">Default button</button>
  <button class="btn" type="button">Default button</button>
</p>
<p>
  <button class="btn btn-small btn-primary" type="button">Small button</button>

  <button class="btn btn-small" type="button">Small button</button>
</p>
<p>
  <button class="btn btn-mini btn-primary" type="button">Mini button</button>
  <button class="btn btn-mini" type="button">Mini button</button>
</p>
```

Button Group

Button groups allow multiple buttons to be stacked together (see Figure 3-3). This is useful when you want to place items like alignment buttons together. To create a button group, simply wrap a series of anchors or buttons in a <div> that has .btn-group as a class:

```
<div class="btn-group">
  <button class="btn">1</button>
  <button class="btn">2</button>
  <button class="btn">3</button>
</div>
```



Buttons with Dropdowns

To add a dropdown to a button, simply wrap the button and dropdown menu in a .btn-group. You can also use to act as an indicator that the button is a dropdown

```
<div class="btn-group">
  <button class="btn btn-danger">Danger</button>
  <button class="btn btn-danger dropdown-toggle" data-toggle="dropdown">
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Action</a></li>
    <li><a href="#">Another action</a></li>
    <li><a href="#">Something else here</a></li>
    <li class="divider"></li>
    <li><a href="#">Separated link</a></li>
  </ul>
</div>
```

Btn checkbox

```
<div class="btn-group" >
  <input type="checkbox" class="btn-check" id="btncheck1">
  <label class="btn btn-outline-primary" for="btncheck1">Checkbox 1</label>

  <input type="checkbox" class="btn-check" id="btncheck2" >
  <label class="btn btn-outline-primary" for="btncheck2">Checkbox 2</label>

  <input type="checkbox" class="btn-check" id="btncheck3" >
  <label class="btn btn-outline-primary" for="btncheck3">Checkbox 3</label>
</div>
```

Btn radio

```
<div class="btn-group" >
  <input type="radio" class="btn-check" name="btnradio" id="btnradio1" checked>
  <label class="btn btn-outline-primary" for="btnradio1">Radio 1</label>

  <input type="radio" class="btn-check" name="btnradio" id="btnradio2" >
  <label class="btn btn-outline-primary" for="btnradio2">Radio 2</label>

  <input type="radio" class="btn-check" name="btnradio" id="btnradio3" >
  <label class="btn btn-outline-primary" for="btnradio3">Radio 3</label>
</div>
```

Q5b) Explain how to create custom forms in Bootstraps with an example

Bootstrap 4 enables to customize the browser's default form and control layouts. The customized form can be created by using Bootstrap 4 like checkbox, radio buttons, file inputs and more. Bootstrap simplifies the process of alignment and styling of web pages in many forms like label, input, field, textarea, button, checkbox, etc.

Custom Checkbox: The .custom-control and .custom-checkbox classes are used in <div> element to wrap the container element. The .custom-control-input class is used with input type="checkbox" to create custom input textbox

```
.<div class="custom-control custom-checkbox mb-3">

                <input type="checkbox" class="custom-control-input"
                    id="customCheckBox" name="checkbox1">
                <label class="custom-control-label" for="customCheckBox">
                    Custom checkbox
                </label>
            </div>
```

Custom switch: The .custom-control and .custom-switch classes are used to wrap the input checkbox. The .custom-control-input class is used with label tag. Bootstrap switch/toggle is a simple component used for activating one of two predefined options. Commonly used as an on/off button. A toggle button allows the user to change a setting between two states.

```
<div class="custom-control custom-switch">
                <input type="checkbox" class="custom-control-input"
                    id="customSwitch" name="switch">
                <label class="custom-control-label" for="customSwitch">
                    Toggle Off
                </label>
            </div>
```

Custom Radio button: It is the same as a checkbox. It uses .custom-radio instead of .custom-input on the label tag. Checkbox and radio buttons are made to support HTML-based form validation and give brief, friendly labels.

```
<div class="custom-control custom-radio">
                <input type="radio" class="custom-control-input"
```

```
            id="customRadio" name="radioButton">
        <label class="custom-control-label" for="customRadio">
            Radio Button Off
        </label>
    </div>
```

Q6a) With suitable code snippets explain Grid system of Bootstrap

Default Grid System

The default Bootstrap grid (see Figure 1-1) system utilizes 12 columns, making for a 940px-wide container without responsive features enabled. With the responsive CSS file added, the grid adapts to be 724px or 1170px wide, depending on your viewport. Below 767px viewports, such as the ones on tablets and smaller devices, the columns become fluid and stack vertically. At the default width, each column is 60 pixels wide and offset 20 pixels to the left. An example of the 12 possible columns is in Figure 1-1

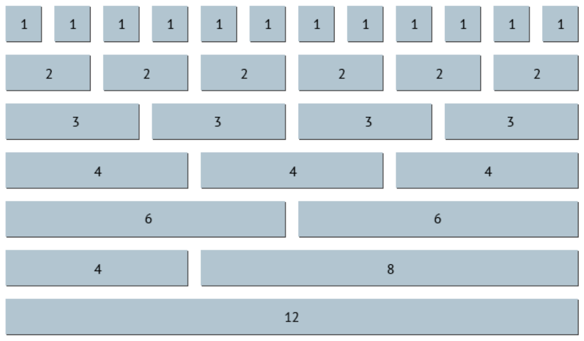


Figure 1-1. Default grid

Basic Grid HTML

To create a simple layout, create a container with a <div> that has a class of .row and add the appropriate amount of .span* columns. Since we have a 12-column grid, we just need the amount of .span* columns to equal 12. We could use a 3-6-3 layout, 4-8, 3-5-4, 2-8-2… we could go on and on, but I think you get the gist.
The following code shows .span8 and .span4, which adds up to 12:

```
<div class="row">
<div class="span8">...</div>
<div class="span4">...</div>
</div>
```

Offsetting Columns

You can move columns to the right using the .offset* class. Each class moves the span over that width. So an .offset2 would move a .span7 over two columns (see Figure 1-2):

```
<div class="row">
<div class="span2">...</div>
<div class="span7 offset2">...</div>
</div>
```

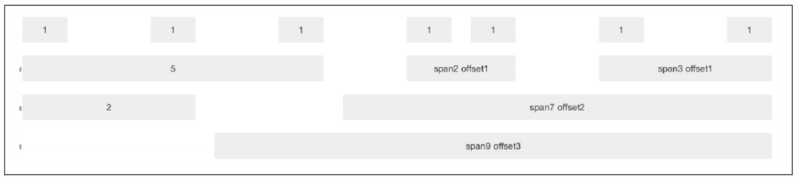


Figure 1-2. Offset grid

Nesting Columns

To nest your content with the default grid, inside of a .span*, simply add a new .row with enough .span* that it equals the number of spans of the parent container (see Figure 1-3):

```
<div class="row">
  <div class="span9">
    Level 1 of column
    <div class="row">
      <div class="span6">Level 2</div>
      <div class="span3">Level 2</div>
    </div>
  </div>
</div>
```

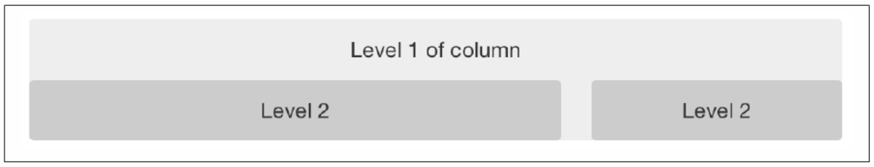


Figure 1-3. Nesting grid

Fluid Grid System

The fluid grid system uses percentages instead of pixels for column widths. It has the same responsive capabilities as our fixed grid system, ensuring proper proportions for key screen resolutions and devices. You can make any row “fluid” by changing .row to .row-fluid. The column classes stay exactly the same, making it easy to flip between fixed and fluid grids. To offset, you operate in the same way as the fixed grid system— add .offset* to any column to shift by your desired number of columns:

```
<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span8">...</div>
</div>

<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span4 offset2">...</div>
</div>
```

Nesting a fluid grid is a little different. Since we are using percentages, each .row resets the column count to 12. For example, if you were inside a .span8, instead of two .span4 elements to divide the content in half, you would use two .span6 divs (see Figure 1-4). This is the case for responsive content, as we want the content to fill 100% of the container:

```
<div class="row-fluid">
  <div class="span8">
              <div class="row">
                      <div class="span6">...</div>
                      <div class="span6">...</div>
              </div>
  </div>
</div>
```

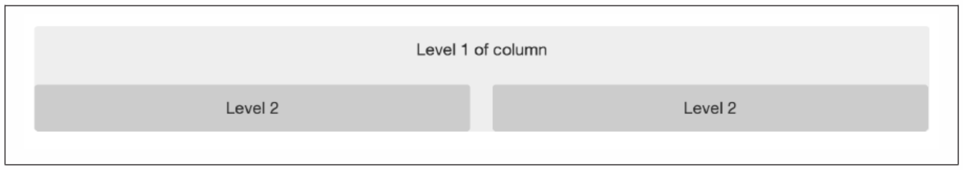


Figure 1-4. Nesting fluid grid

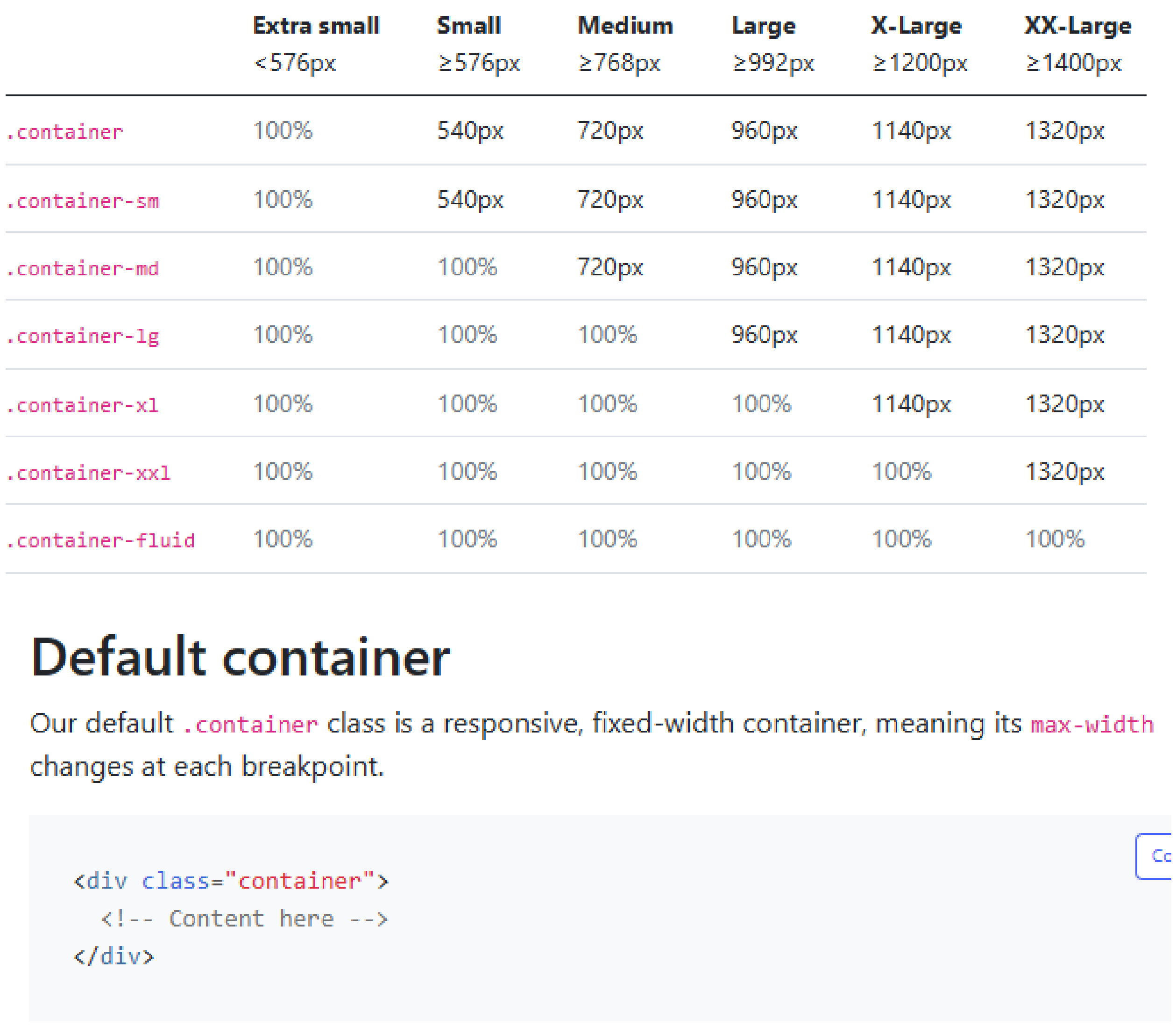
Q6b) Describe different containers in Bootstrap with example

Containers are the most basic layout element in Bootstrap and are **required when using our default grid system**. Containers are used to contain, pad, and (sometimes) center the content within them. While containers *can* be nested, most layouts do not require a nested container.

Bootstrap comes with three different containers:

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

The table below illustrates how each container's max-width compares to the original .container and .container-fluid across each breakpoint.

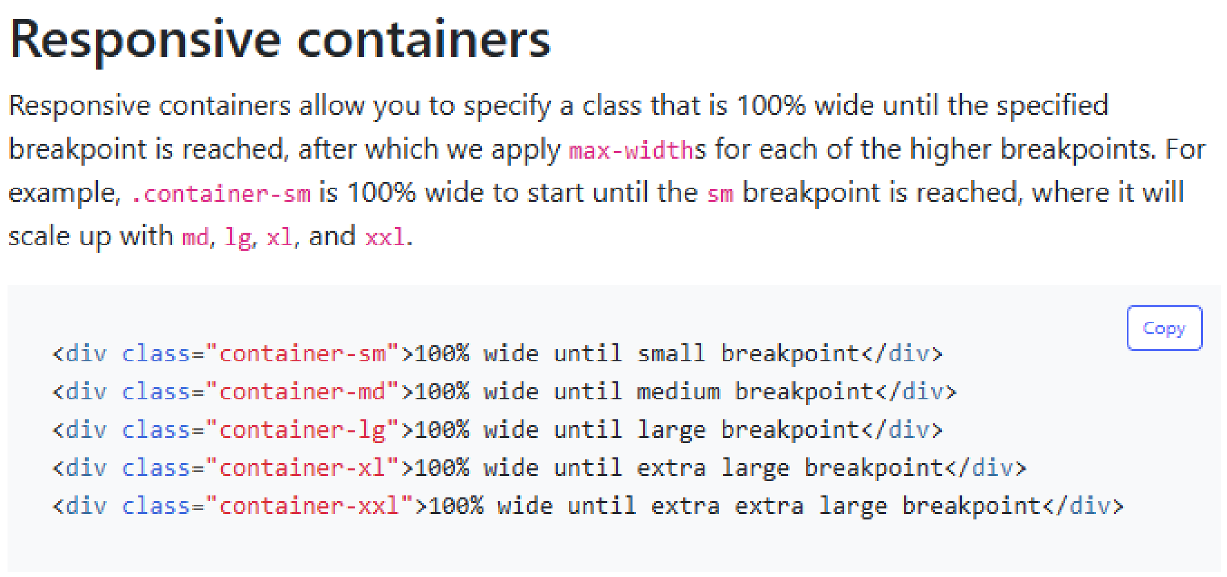


	Extra small <576px	**Small** ≥576px	**Medium** ≥768px	**Large** ≥992px	**X-Large** ≥1200px	**XX-Large** ≥1400px
`.container`	100%	540px	720px	960px	1140px	1320px
`.container-sm`	100%	540px	720px	960px	1140px	1320px
`.container-md`	100%	100%	720px	960px	1140px	1320px
`.container-lg`	100%	100%	100%	960px	1140px	1320px
`.container-xl`	100%	100%	100%	100%	1140px	1320px
`.container-xxl`	100%	100%	100%	100%	100%	1320px
`.container-fluid`	100%	100%	100%	100%	100%	100%

Default container

Our default `.container` class is a responsive, fixed-width container, meaning its `max-width` changes at each breakpoint.

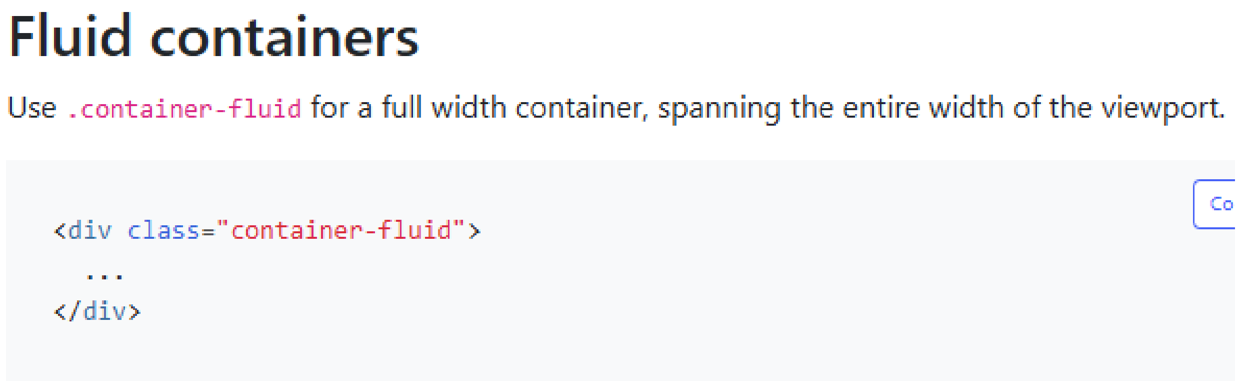
```
<div class="container">
  <!-- Content here -->
</div>
```



Responsive containers

Responsive containers allow you to specify a class that is 100% wide until the specified breakpoint is reached, after which we apply `max-width`s for each of the higher breakpoints. For example, `.container-sm` is 100% wide to start until the `sm` breakpoint is reached, where it will scale up with `md`, `lg`, `xl`, and `xxl`.

```
<div class="container-sm">100% wide until small breakpoint</div>
<div class="container-md">100% wide until medium breakpoint</div>
<div class="container-lg">100% wide until large breakpoint</div>
<div class="container-xl">100% wide until extra large breakpoint</div>
<div class="container-xxl">100% wide until extra extra large breakpoint</div>
```



Fluid containers

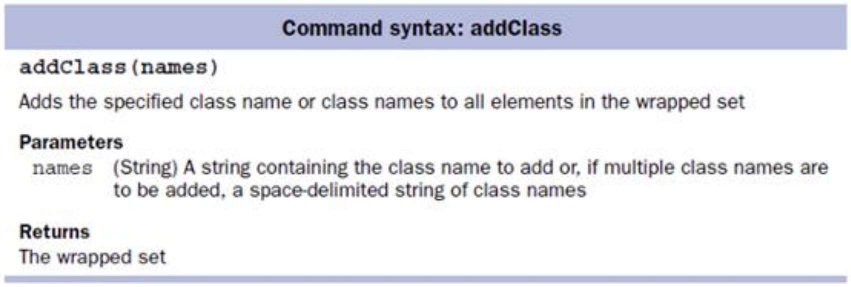
Use `.container-fluid` for a full width container, spanning the entire width of the viewport.

```
<div class="container-fluid">
  ...
</div>
```

Q7a) Demonstrate the use of different methods for adding and removing classes with suitable code example

I. Adding and removing class names

Adding class names to all the elements of a matched set is an easy operation with the following addClass() command:



Command syntax: addClass

`addClass(names)`

Adds the specified class name or class names to all elements in the wrapped set

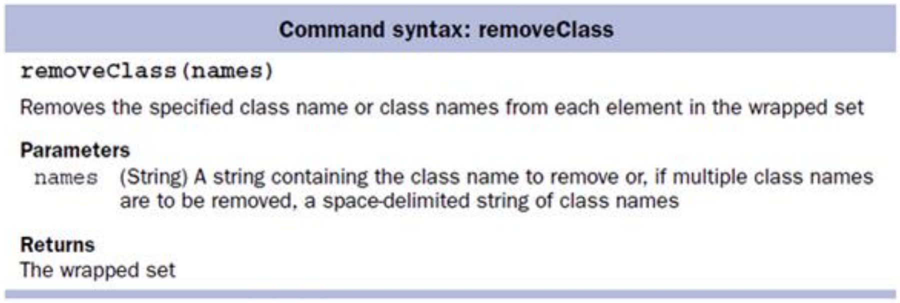
Parameters

`names` (String) A string containing the class name to add or, if multiple class names are to be added, a space-delimited string of class names

Returns

The wrapped set

Removing class names is as straightforward with the following removeClass() command:



Command syntax: removeClass

`removeClass(names)`

Removes the specified class name or class names from each element in the wrapped set

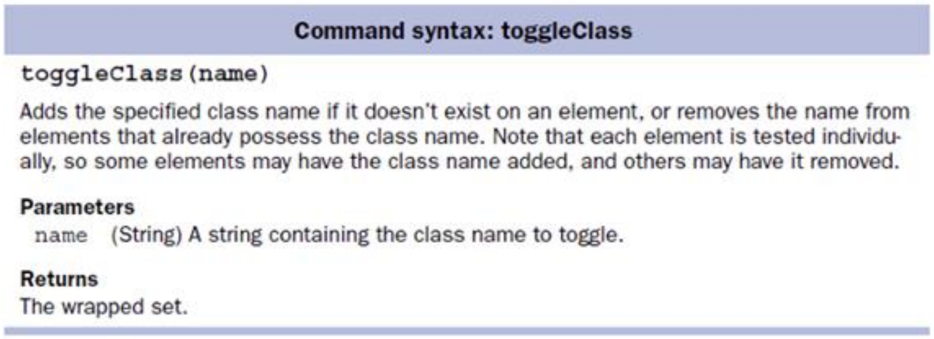
Parameters

`names` (String) A string containing the class name to remove or, if multiple class names are to be removed, a space-delimited string of class names

Returns

The wrapped set

Often, we may want to switch a set of styles back and forth, perhaps to indicate a change between two states or for any other reasons that make sense with our interface. jQuery makes it easy with the toggleClass() command.



Command syntax: toggleClass

`toggleClass(name)`

Adds the specified class name if it doesn't exist on an element, or removes the name from elements that already possess the class name. Note that each element is tested individually, so some elements may have the class name added, and others may have it removed.

Parameters

`name` (String) A string containing the class name to toggle.

Returns

The wrapped set.

One situation where the toggleClass() command is most useful is when we want to switch visual renditions between elements quickly and easily.

Example $('tr').toggleClass('striped');

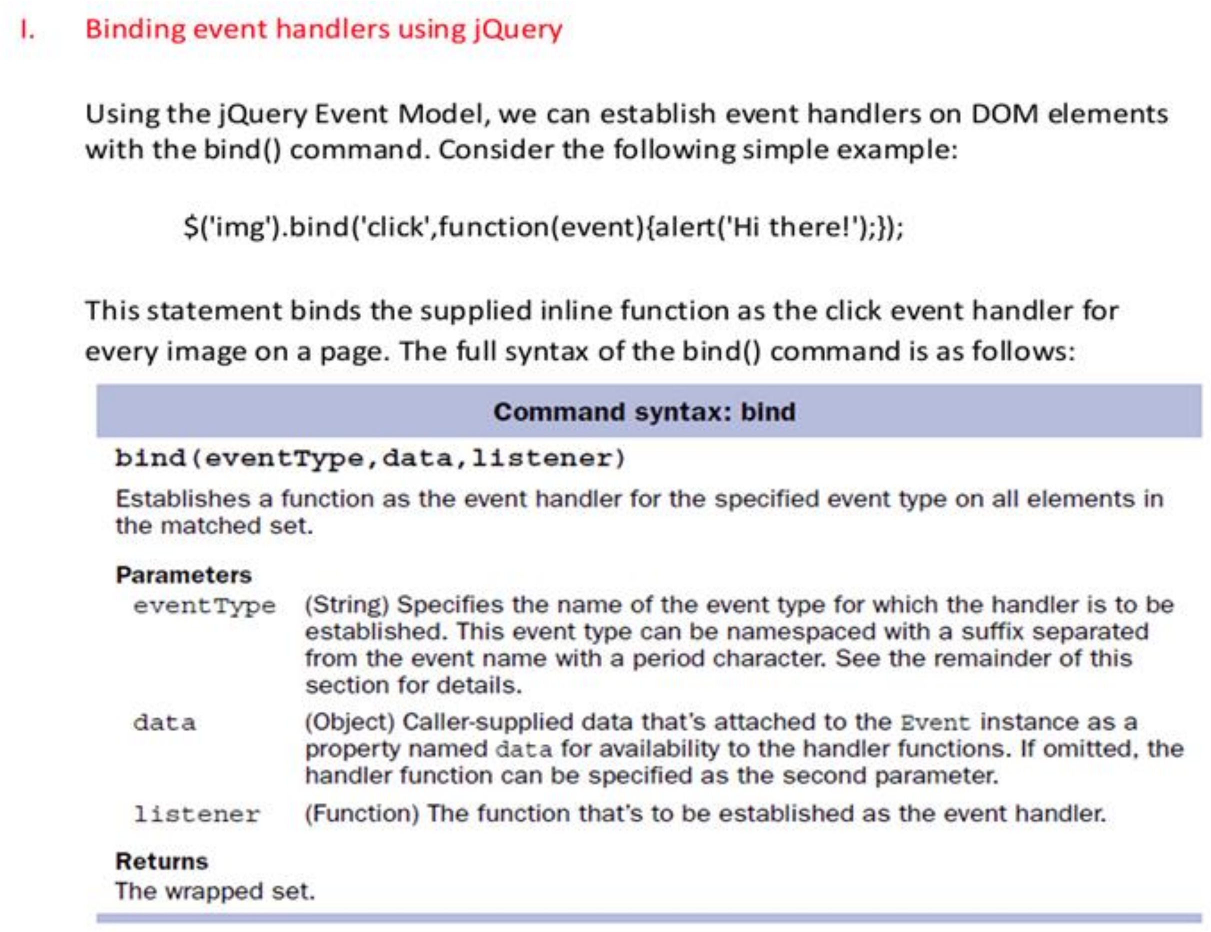
Q7b) Explain the following:

i)bind()

ii) evntTypeName()

iii) One()

iv) Unbind()



I. Binding event handlers using jQuery

Using the jQuery Event Model, we can establish event handlers on DOM elements with the bind() command. Consider the following simple example:

```
$('img').bind('click',function(event){alert('Hi there!');});
```

This statement binds the supplied inline function as the click event handler for every image on a page. The full syntax of the bind() command is as follows:

Command syntax: bind

`bind(eventType,data,listener)`

Establishes a function as the event handler for the specified event type on all elements in the matched set.

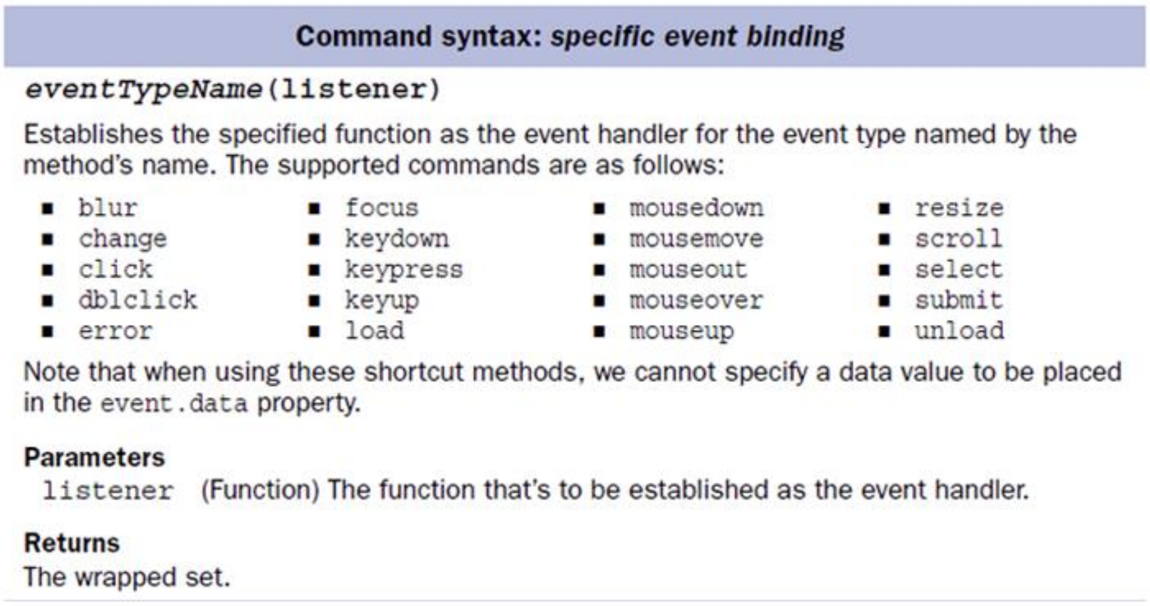
Parameters

`eventType`	(String) Specifies the name of the event type for which the handler is to be established. This event type can be namespaced with a suffix separated from the event name with a period character. See the remainder of this section for details.
`data`	(Object) Caller-supplied data that's attached to the `Event` instance as a property named `data` for availability to the handler functions. If omitted, the handler function can be specified as the second parameter.
`listener`	(Function) The function that's to be established as the event handler.

Returns

The wrapped set.

In addition to the bind() command, jQuery provides a handful of shortcut commands to establish specific event handlers.



Command syntax: *specific event binding*

`eventTypeName(listener)`

Establishes the specified function as the event handler for the event type named by the method's name. The supported commands are as follows:

- `blur`
- `change`
- `click`
- `dblclick`
- `error`
- `focus`
- `keydown`
- `keypress`
- `keyup`
- `load`
- `mousedown`
- `mousemove`
- `mouseout`
- `mouseover`
- `mouseup`
- `resize`
- `scroll`
- `select`
- `submit`
- `unload`

Note that when using these shortcut methods, we cannot specify a data value to be placed in the `event.data` property.

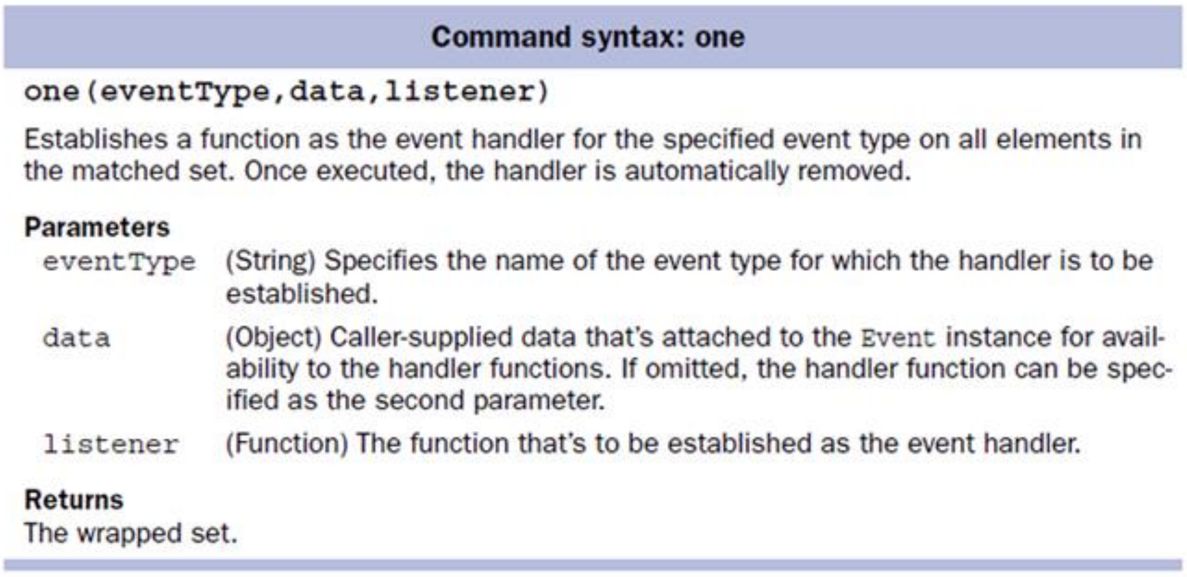
Parameters

`listener`	(Function) The function that's to be established as the event handler.

Returns

The wrapped set.

jQuery also provides a specialized version of the bind() command, named one(), that establishes an event handler as a one-shot deal. Once the event handler executes the first time, it's automatically removed as an event handler. Its syntax is similar to the bind() command and is as follows:



Command syntax: one

`one(eventType,data,listener)`

Establishes a function as the event handler for the specified event type on all elements in the matched set. Once executed, the handler is automatically removed.

Parameters

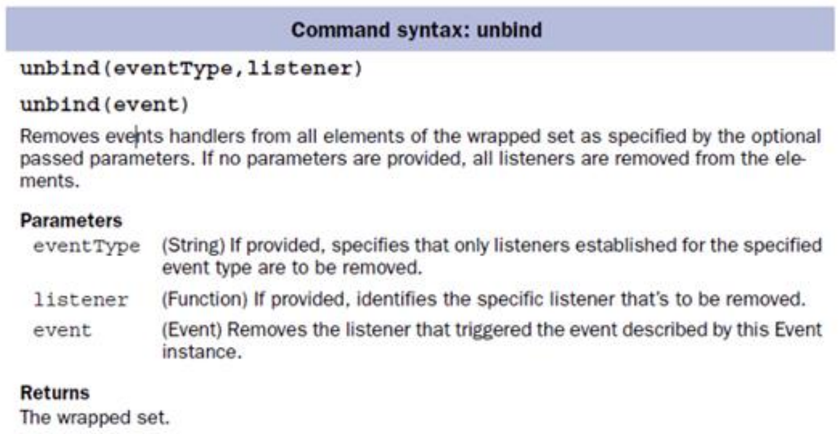
`eventType`	(String) Specifies the name of the event type for which the handler is to be established.
`data`	(Object) Caller-supplied data that's attached to the `Event` instance for availability to the handler functions. If omitted, the handler function can be specified as the second parameter.
`listener`	(Function) The function that's to be established as the event handler.

Returns

The wrapped set.

II. Removing event handlers

We've seen that the one() command can automatically remove a handler after it has completed its first (and only) execution, but for the more general case where we'd like to remove event handlers under our own control, jQuery provides the unbind() command.
The syntax of unbind() is as follows:



Command syntax: unbind

```
unbind(eventType,listener)
unbind(event)
```

Removes events handlers from all elements of the wrapped set as specified by the optional passed parameters. If no parameters are provided, all listeners are removed from the elements.

Parameters

`eventType`	(String) If provided, specifies that only listeners established for the specified event type are to be removed.
`listener`	(Function) If provided, identifies the specific listener that's to be removed.
`event`	(Event) Removes the listener that triggered the event described by this Event instance.

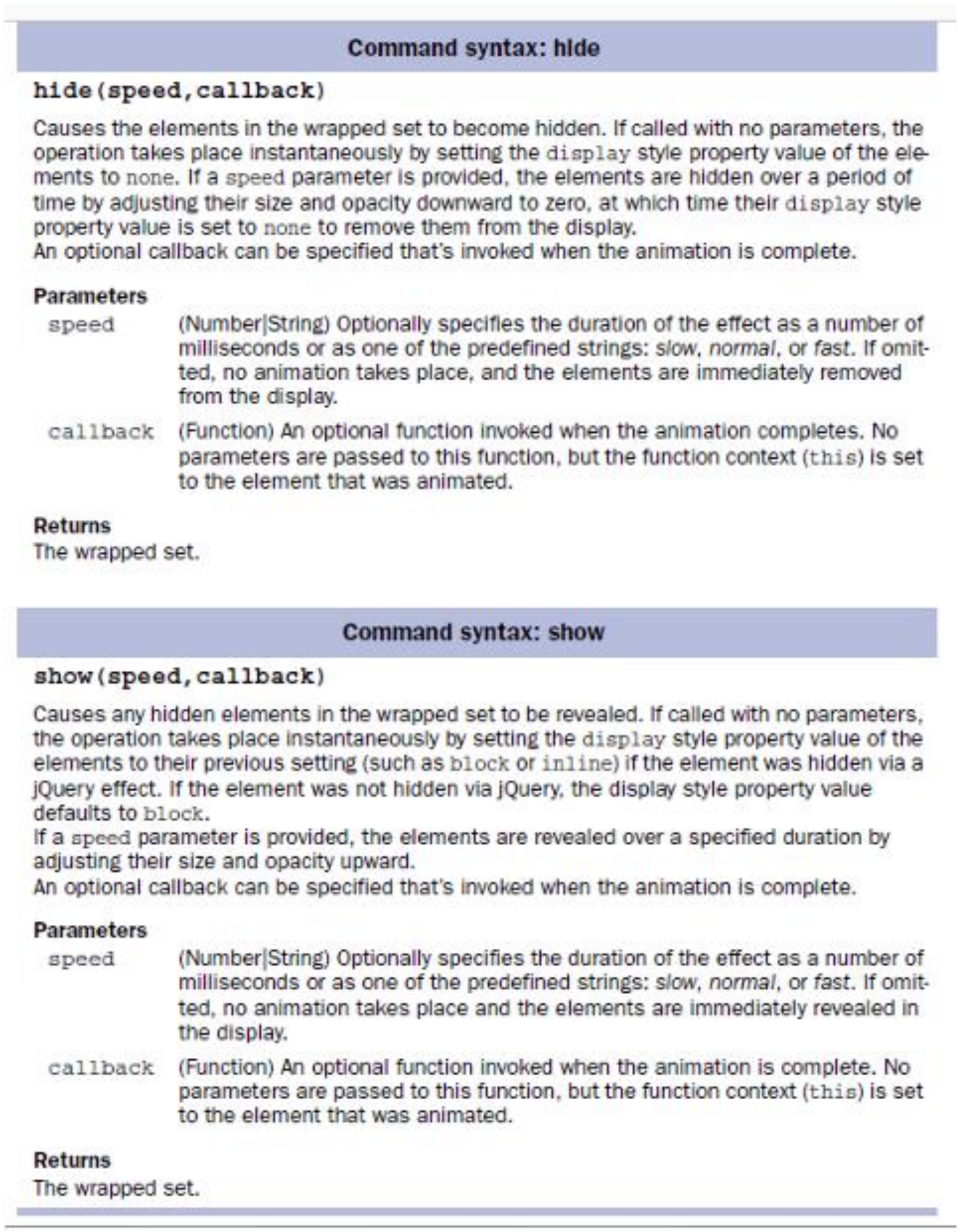
Returns
The wrapped set.

This command can be used to remove event handlers from the elements of the matched set at various levels of granularity. All listeners can be removed by omitting parameters, or listeners of a specific type can be removed by providing that event type.
Specific handlers can be removed by providing a reference to the function originally established as the listener.

Q8a) Discuss the commands for showing and hiding elements with an example

1) Showing and hiding elements gradually

The show(), hide(), and toggle() commands are more complex, when called with no parameters, these commands effect a simple manipulation of the display state of the wrapped elements, causing them to instantaneously be revealed or hidden from the display. But when passed parameters, these effects can be animated so that their changes in display status take place over a period of time.



Command syntax: hide

`hide(speed,callback)`

Causes the elements in the wrapped set to become hidden. If called with no parameters, the operation takes place instantaneously by setting the `display` style property value of the elements to `none`. If a `speed` parameter is provided, the elements are hidden over a period of time by adjusting their size and opacity downward to zero, at which time their `display` style property value is set to `none` to remove them from the display.
An optional callback can be specified that's invoked when the animation is complete.

Parameters

`speed`	(Number|String) Optionally specifies the duration of the effect as a number of milliseconds or as one of the predefined strings: *slow*, *normal*, or *fast*. If omitted, no animation takes place, and the elements are immediately removed from the display.
`callback`	(Function) An optional function invoked when the animation completes. No parameters are passed to this function, but the function context (`this`) is set to the element that was animated.

Returns

The wrapped set.

Command syntax: show

`show(speed,callback)`

Causes any hidden elements in the wrapped set to be revealed. If called with no parameters, the operation takes place instantaneously by setting the `display` style property value of the elements to their previous setting (such as `block` or `inline`) if the element was hidden via a jQuery effect. If the element was not hidden via jQuery, the display style property value defaults to `block`.
If a `speed` parameter is provided, the elements are revealed over a specified duration by adjusting their size and opacity upward.
An optional callback can be specified that's invoked when the animation is complete.

Parameters

`speed`	(Number|String) Optionally specifies the duration of the effect as a number of milliseconds or as one of the predefined strings: *slow*, *normal*, or *fast*. If omitted, no animation takes place and the elements are immediately revealed in the display.
`callback`	(Function) An optional function invoked when the animation is complete. No parameters are passed to this function, but the function context (`this`) is set to the element that was animated.

Returns

The wrapped set.

Q8b) Explain how to use child, container and attribute selectors in JQuery

2. Using child, container, and attribute selectors

Child selector

Consider the following HTML fragment:

```
<ul class="myList">
        <li><a href="http://jquery.com">jQuery supports</a>
                <ul>
                        <li><a href="css1">CSS1</a></li>
                        <li><a href="css2">CSS2</a></li>
                        <li><a href="css3">CSS3</a></li>
                        <li>Basic XPath</li>
                </ul>


        </li>
        <li>jQuery also supports
                <ul>
                        <li>Custom selectors</li>
                        <li>Form selectors</li>
                </ul>
        </li>
</ul>
```

Suppose we want to select the link to the remote jQuery site, but not the links to various local pages describing the different CSS specifications. Using basic CSS selectors, we might try something like ul.myList li a. Unfortunately, that selector would grab all links because they all descend from a list element.

A more advanced approach is to use child selectors, in which a parent and its direct child are separated by the right angle bracket character (>), as in

```
p > a
```

This selector matches only links that are direct children of a <p> element. If a link were further embedded, say within a <span> within the <p>, that link would not be selected. Going back to our example, consider a selector such as

```
ul.myList > li > a
```

This selector selects only links that are direct children of list elements, which are in turn direct children of <ul> elements that have the class myList. The links contained in the sublists are excluded because the <ul> elements serving as the parent of the sublists <li> elements don’t have the class myList,

Attribute selectors are also extremely powerful. Say we want to attach a special behavior only to links that point to locations outside our sites.

```
<li><a href="http://jquery.com">jQuery supports</a>
<ul>
<li><a href="css1">CSS1</a></li>
<li><a href="css2">CSS2</a></li>
<li><a href="css3">CSS3</a></li>
<li>Basic XPath</li>
</ul>
</li>
```

What makes the link pointing to an external site unique is the presence of the string http:// at the beginning of the value of the link's href attribute. We could select links with an href value starting with http:// with the following selector:

```
a[href^=http://]
```

This matches all links with a href value beginning with exactly http://. The caret character (^) is used to specify that the match is to occur at the beginning of a value. This is the same character used by most regular expression processors to signify matching at the beginning of a candidate string; it should be easy to remember.

There are other ways to use attribute selectors. To match an element that possesses a specific attribute, regardless of its value, we can use

```
form[method]
```

This matches any <form> element that has an explicit method attribute. To match a specific attribute value, we use something like

```
input[type=text]
```

This selector matches all input elements with a type of text.

```
div[title^=my]
```

This selects all <div> elements with title attributes whose value begins with my.
What about an “attribute ends with” selector?

```
a[href$=.pdf]
```

This is a useful selector for locating all links that reference PDF files.
And there’s a selector for locating elements whose attributes contain arbitrary strings anywhere in the attribute value:

```
a[href*=jquery.com]
```

As we would expect, this selector matches all <a> elements that reference the jQuery site.

container selector

li:has(a)

This selector matches all <li> elements that contain an <a> element. Note that this is not the same as a selector of li a, which matches all <a> elements contained within <li> elements.

Table shows the CSS selectors that we can use with jQuery.

Only a single level of nesting is supported. Although it's possible to nest one level, such as

    foo:not(bar:has(baz))

additional levels of nesting, such as

    foo:not(bar:has(baz:eq(2)))

aren't supported.

Selector	Description
`*`	Matches any element.
`E`	Matches all element with tag name `E`.
`E F`	Matches all elements with tag name `F` that are descendents of `E`.
`E>F`	Matches all elements with tag name `F` that are direct children of `E`.
`E+F`	Matches all elements `F` immediately preceded by sibling `E`.
`E~F`	Matches all elements `F` preceded by any sibling `E`.
`E:has(F)`	Matches all elements with tag name `E` that have at least one descendent with tag name `F`.
`E.C`	Matches all elements `E` with class name `C`. Omitting `E` is the same as `*.C`.
`E#I`	Matches element `E` with id of `I`. Omitting `E` is the same as `*#I`.
`E[A]`	Matches all elements `E` with attribute `A` of any value.
`E[A=V]`	Matches all elements `E` with attribute `A` whose value is exactly `V`.
`E[A^=V]`	Matches all elements `E` with attribute `A` whose value begins with `V`.
`E[A$=V]`	Matches all elements `E` with attribute `A` whose value ends with `V`.
E[A*=V]	Matches all elements `E` with attribute `A` whose value contains `V`.

Q9a) Discuss different types of expressions in Angular JS with an example

{{ Expression }}

{{Expression}} is used to bind the value with html element and displays the value. It works same as **ng-bind** directive. {{Expression}} is written within two curly brackets. The {{expression}} is basically pure JavaScript expression.

String Expression

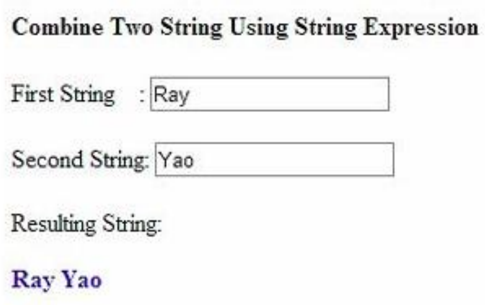
We know that string is collection of characters. In AngularJSthe string expression looks like this.

```
<element> {{First String + Second String}}
</element>
```

Example 6.1

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJS for beginners</title> <script src="js/angular.min.js">
</script> </head>
<body>
<h4>Combine Two String Using String Expression</h4> <div ng-app="" >
First String &nbsp;&nbsp;&nbsp;: <input type="text" ng-
model="firstString"/><br><br> Second String: <input ng-
model="secondString"/><br><br> Resulting String:<p
style="color:blue;font-weight:bold;">{{firstString +" "+secondString}}
</p> </div>
</body>
</html>
```

Output:



```
Combine Two String Using String Expression

First String   : Ray

Second String: Yao

Resulting String:

Ray Yao
```

Explanation:

"{{firstString +" "+secondString}}" joins two strings together.

{{ expression}} displays the value of expression.

In the above example, the text **Ray** is written in the first text box and **Yao** in the second text box, but in the resulting string area, Ray string and Yao string are combined due to use of string expression {{ }}. Note: The plus + sign is used for string concatenation.

Number Expression

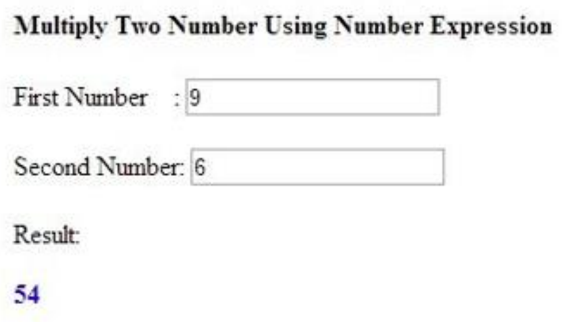
In AngularJSyou can perform different mathematic operation by using Number Expression.

```
<element> {{First Number + Second Number}}
</element>
```

Example 6.2

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJS for beginners</title> <script src="js/angular.min.js">
</script> </head>
<body>
<h4>Multiply Two Number Using Number Expression</h4> <div ng-app=""
ng-init="firstNumber=9;secondNumber=6"> First Number
&nbsp;&nbsp;&nbsp;: <input type="number" ng-model="firstNumber"/>
<br><br> Second Number: <input type="number" ng-
model="secondNumber"/><br><br> Result:<p style="color:blue;font-
weight:bold;">{{firstNumber * secondNumber}}</p> </div>
</body>
</html>
```

Output:



Multiply Two Number Using Number Expression

```
First Number   : 9
Second Number: 6
Result:
54
```

Explanation:

"{{firstNumber * secondNumber}}" multiplies the firstNumber and the secondNumber.

{{ expression}} displays the value of expression.

In the above example, the number **9** is written in the first text box and **6** in the second text box, and **54** is the result of multiplication of 9 and 6. You can perform any arithmetic operation by using Number Expression.

Object Expression

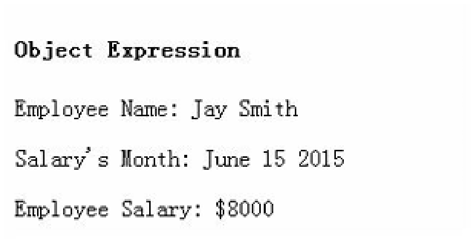
AngularJSobject works like a JavaScript object. The syntax looks like this:

```
object = {property: value}
```

Example 6.3

```
<html >
<script src= "js\angular.min.js"></script>
<body>
<h4>Object Expression</h4> <div ng-app=""  ng-init="EmployeeObject = {Emp_name: 'Jay Smith',     Emp_Month: 'June.15 2015',  Emp_salary: '$8000'}"> <p>Employee Name :  {{EmployeeObject.Emp_name}}</p> <p>Salary's Month:  {{EmployeeObject.Emp_Month}}</p> <p>Employee Salary:  {{EmployeeObject.Emp_salary}}</p> </div>
</body>
</html>
```

Output:



```
Object Expression

Employee Name: Jay Smith

Salary's Month: June 15 2015

Employee Salary: $8000
```

"Emp_salary" is a property.

{{ object.property }} displays the value of the property.

Array Expression

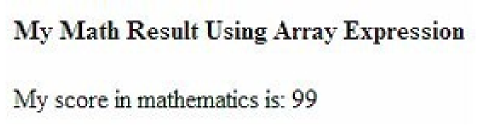
The array expression of AngularJS works like JavaScript array. The syntax looks like this:

```
Array=[val1, val2, val3,]
```

Example 6.4

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJS for beginners</title> <script src="js/angular.min.js">
</script> </head>
<body>
<h4>My Math Result Using Array Expression</h4> <div ng-app="" ng-
init="MyArray=[98,96,93,90,99]"> <p>My score in mathematics is:
{{MyArray[4]}}</p> </div>
</body>
</html>
```

Output:



My Math Result Using Array Expression

My score in mathematics is: 99

Explanation:

"MyArray=[98,96,93,90,99]" is an array.

"{{MyArray[4]}}" displays the value whose index is 4 in MyArray.

Q9b) What is $scope and explain how to use controllers with an example

What is Scope?

Scope is a JavaScript object which contains model data.

```
function ($scope) {  }
```

$scope is a parameter of JavaScript function which is called by a controller.
Let`s take an example for understanding.

Example 7.2

```
<script>
function ($scope) {
   $scope.firstNumber = 23;
   $scope.secondNumber = 63;
}
</script>
```

Explanation: In above example, the **$scope** is the parameter of the function ($scope) { }. $scope is an object in AngularJS.

$scope.**firstNumber and **$scope.secondNumber are models used in HTML. Model data is accessed by the $scope object. We assign values to the model with following formula: "$scope.property = value".

How to define Controller?

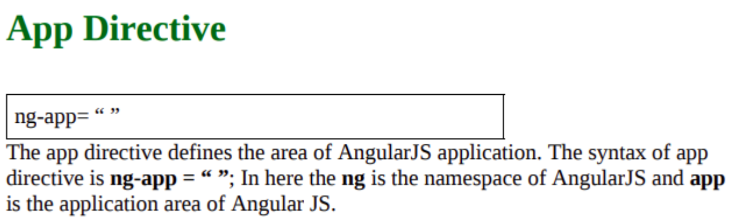
The **ng-controller** directive is used to define the Controller. We know that Controller is a JavaScript object which contains JavaScript function and properties. The syntax of Controller is as following:

```
<div ng-app=""  ng-controller="controllerName">
```

Example 7.1

```
<html>
<script src= "js\angular.min.js"></script> <body>
<div ng-app="Calculation" ng-controller="myController"> First Number:
<input type="number" ng-model="firstNumber"><br> Second Number:
<input type="number" ng-model="secondNumber"><br> <br>
Sum: {{firstNumber + secondNumber}}
</div>
<script>
var app = angular.module('Calculation', [ ]); app.controller('myController',
function($scope) {
   $scope.firstNumber = 4; $scope.secondNumber = 8; });
</script>
</body>
</html>
```

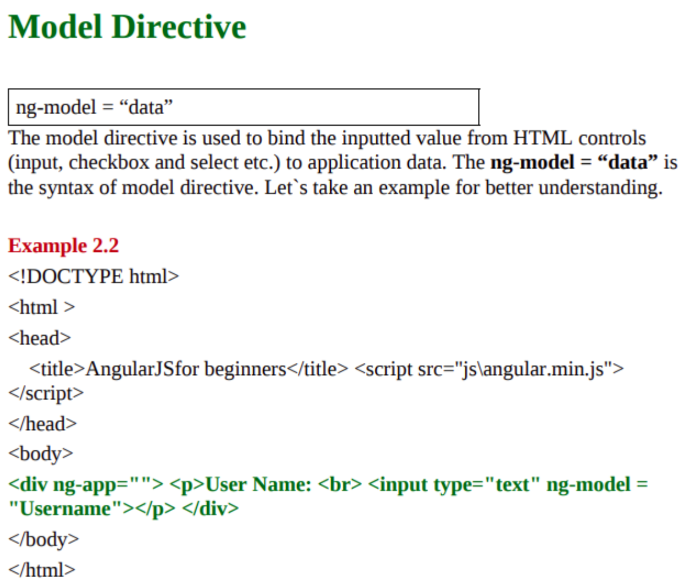
Q10a) Explain the following directives with example i)ng-app ii)ng-model iii)ng-init iv) ng-repeat



App Directive

```
ng-app= “ ”
```

The app directive defines the area of AngularJS application. The syntax of app directive is **ng-app = “ ”**; In here the **ng** is the namespace of AngularJS and **app** is the application area of Angular JS.



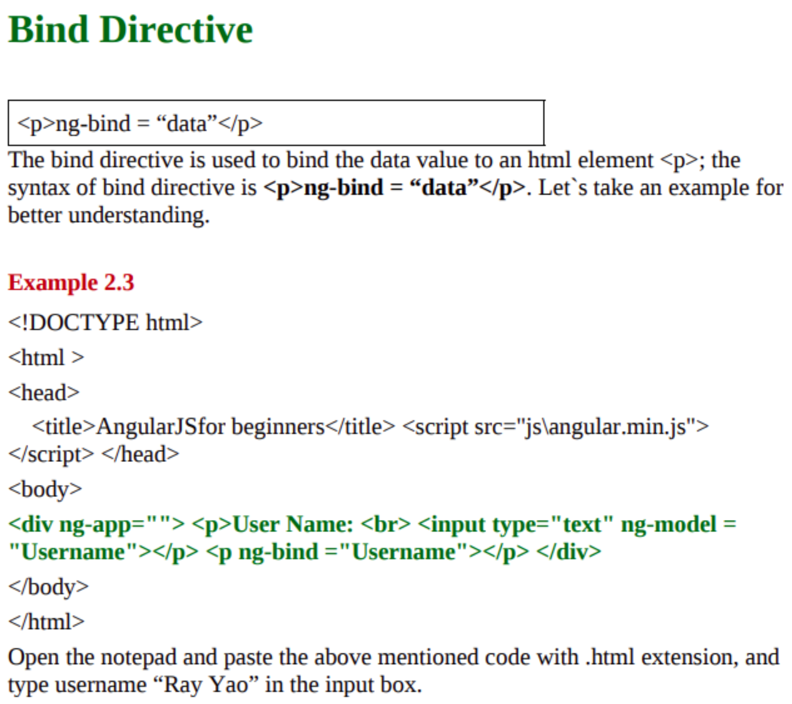
Model Directive

```
ng-model = “data”
```

The model directive is used to bind the inputted value from HTML controls (input, checkbox and select etc.) to application data. The **ng-model = “data”** is the syntax of model directive. Let`s take an example for better understanding.

Example 2.2

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script>
</head>
<body>
<div ng-app=""> <p>User Name: <br> <input type="text" ng-model =
"Username"></p> </div>
</body>
</html>
```



Bind Directive

```
<p>ng-bind = “data”</p>
```

The bind directive is used to bind the data value to an html element <p>; the syntax of bind directive is **<p>ng-bind = “data”</p>**. Let`s take an example for better understanding.

Example 2.3

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<div ng-app=""> <p>User Name: <br> <input type="text" ng-model =
"Username"></p> <p ng-bind ="Username"></p> </div>
</body>
</html>
```

Open the notepad and paste the above mentioned code with .html extension, and type username “Ray Yao” in the input box.

Init Directive

```
ng-init = "data = 'value'"
```

The init directive is used to initialize the data with a value. The syntax of init directive is **ng-init = "data = 'value'"**. Let`s take an example for better understanding.

Example 2.4

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<div ng-app=""    ng-init="Username= 'Andy Smith' "> <p>User Name:
<input type="text" ng-model = "Username"></p> <p ng-
bind="Username"></p> </div>
</body>
</html>
```

Repeat Directive

```
ng-repeat = "variable in array"
```

The repeat directive works like a loop. The **ng-repeat** directive repeats to get the value of an array.

Example 2.5

```
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
<div ng-app=""  ng-init = "ColorName = ['Pink', 'Red', 'Green', 'Blue',
'Black', 'White', 'Yellow', 'Gray']"> <p style="color:green; font-
weight:bold">Colours Name:</p> <ol>
    <li ng-repeat ="x in ColorName"> <p ng-bind="x"></p> </li>
</ol>
</div>
</body>
</html>
```

Q10b) What is filter? Explain uppercase, lowercase, order by and currency with an example

Uppercase filter

Value | uppercase

The uppercase filter changes the text to upper case. Suppose a user writes a text in lower case (e.g. ray) or title case (e.g. Ray) or in mixed case (e.g. rAy or RaY or rAY etc.), and you want the upper case result, then you will have to use upper case filter.

Example 3.1

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
<h3>Using Upper Case Filter</h3> <div ng-app=""  ng-init="Username=
'ray' "> <p>User Name: <input type="text" ng-model = "Username"></p>
<p style="color:red" ng-bind="Username | uppercase"></p> </div>
</body>
</html>
```

Lowercase filter

Value | lowercase

The lowercase filter changes the text to lower case. Suppose a user writes a text in upper case (e.g. RAY YAO) or title case (e.g. Ray Yao) or in mixed case (e.g. rAy or RaY or rAY etc.), and you want the lower case result, then you will have to use lower case filter.

Example 3.2

```
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<h3>Using Lower Case Filter</h3> <div ng-app=""  ng-init="Username=
'Ray YAO' "> <p>User Name: <input type="text" ng-model="Username">
</p> <p style="color:red" ng-bind="Username | lowercase"></p> </div>
</body>
</html>
```

OrderBy filter

OrderBy filer is used to display values in ascending order or descending order.
The syntax of “orderBy” looks like this:

```
Value | orderBy: ’value’    //for ascending order
Value | orderBy: ’-value’   //for descending order
```

Let`s take an example for better understanding.

Example 3.3

```
<!DOCTYPE html>
<html >
<head>
   <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<h1>Using OrderBy filter</h1> <div ng-app="" ng-init="StudentsResult=
[{name: 'Tienq', marks:81},            {name: 'Svbrf', marks:70},
{name: 'Yaito', marks:90},             {name: 'Pewfn', marks:63}, {name:
'Riet', marks:98}]"> <table border="1" > <tr>
<th>Student Name</th> <th>Mathematics’ Result</th> </tr>
<tr ng-repeat="x in StudentsResult | orderBy:'-marks' "> <td ng-
bind="x.name "></td> <td ng-bind="x.marks "></td> </tr>
</table>
</div>
</body>
</html>
```

Currency filter

```
Value | currency
```

The currency filter is used to display the result in currency format.

Example 3.5

```
<!DOCTYPE html>
<html >
<head>
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
<h1>Using Currency filter</h1> <div ng-app="" ng-init =
"Employees_Monthly_Salary=[{name: 'Jay', salary:8100}, {name: 'Sdwt',
salary:7000},              {name: 'Hao', salary:9000}, {name: 'Luoe',
salary:6300}, {name: 'Fin', salary:9800}]"> <table border="1" > <tr>
<th>Employee Name</th> <th>Employee Salary</th> </tr>
<tr ng-repeat="x in Employees_Monthly_Salary  "> <td ng-bind="x.name
"></td> <td ng-bind="x.salary | currency "></td> </tr>
</table>
</div>
</body>
</html>
```