CBCS SCHEME

USN | 1 | CR | 19 | MCA | 08 | 18MCA13

First Semester MCA Degree Examination, Dec.2019/Jan.2020
Web Technologies

Time: 3 hrs. Max. Marks: 100

Note: ***Answer any FIVE full questions, choosing ONE full question from each module.***

Module-1

1 a. What are the two phases of HTTP? Explain each of them in detail. **(10 Marks)**
b. Explain the following : i) Domain name ii) URL. **(10 Marks)**

OR

2 a. With an examples, explain the lists tags. **(06 Marks)**
b. Write a HTML program to describe a table with the rows, cols, rowspan and colspan attributes. **(08 Marks)**
c. Explain the following tags with an example each : i) <input> ii) <select> iii) <textarea>. **(06 Marks)**

Module-2

3 a. Explain the tags nav, section, article, aside, footer. **(10 Marks)**
b. Explain the features of HTML5. **(10 Marks)**

OR

4 a. Explain different selector forms in CSS. **(10 Marks)**
b. Write a XHTML and CSS document to illustrate different font properties. **(10 Marks)**

Module-3

5 a. How arrays can be created in javascript? Explain all array methods with examples. **(10 Marks)**
b. How functions are declared in javascript? Write a Javascript program to calculate median of an array. **(10 Marks)**

OR

6 a. Write a javascript program to show handling events from textbox and password elements. **(10 Marks)**
b. Explain DOM2 Event Model. **(10 Marks)**

Module-4

7 a. Explain the ways of positioning elements in Dynamic Documents with Javascript with a suitable example. **(10 Marks)**
b. Write a program to implement stacking of elements. **(10 Marks)**

OR

8 a. Explain the syntax of XML. **(05 Marks)**
b. How XML document can be written with XSLT form explain. **(10 Marks)**
c. Differentiate between XML and HTML. **(05 Marks)**

Module-5

9 a. Explain different query wrapper. **(10 Marks)**
b. Explain the following commands : size( ), get( ), index( ), add( ), not( ). **(10 Marks)**

OR

10 a. How attr( ) commands can be used in different ways in manipulating element. **(10 Marks)**
b. Explain the following commands bind( ), one( ), unbind( ), trigger( ), eventName( ). **(10 Marks)**

* * * * *

Q1 a) What are the two phases of HTTP? Explain each of them in detail.

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

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.

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.

Q1b) Explain the following i)Domain name ii)URL

Domain Names

The IP addresses are numbers. Hence, it would be difficult for the users to remember IP address. To solve this problem, text based names were introduced. These are technically known as domain name system (DNS).

These names begin with the names of the host machine, followed by progressively larger enclosing collection of machines, called domains. There may be two, three or more domain names. DNS is of the form hostname.domainName.domainName.

Example: www.amazon.in

The steps for conversion from DNS to IP:

- The DNS has to be converted to IP address before destination is reached.
- This conversion is needed because computer understands only numbers.
- The conversion is done with the help of name server.
- As soon as domain name is provided, it will be sent across the internet to contact name servers.
- This name server is responsible for converting domain name to IP
- If one of the name servers is not able to convert DNS to IP, it contacts other name server.
- This process continues until IP address is generated.
- Once the IP address is generated, the host can be accessed.
- The hostname and all domain names form what is known as FULLY QUALIFIED DOMAIN NAME.
- This is as shown below:

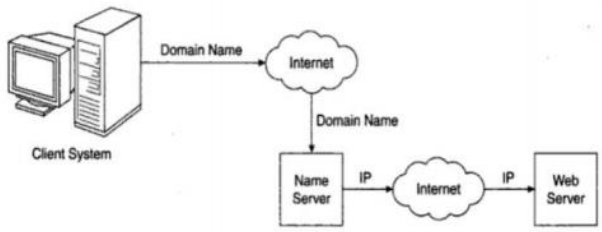

Figure 1.1 Domain name conversion

UNIFORM RESOURCE LOCATORS

- 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.html

Q2a) With an example explain list tags

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) Write and xhtml program to describe a table with rows. Col, colspan and rowspan attributes

```
<?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">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                <title>MCA Department</title>
        </head>
        <body bgcolor="AliceBlue">

                <h3 id="3">Time Table</h3>
                <table border="1" cellspacing="5" cellpadding="5">
                        <tr>
                        <td></td>
                        <td>8-9</td>
                        <td>9-10</td>
                        <td>10-10:30</td>
                        <td>10.30-11.30</td>
                        <td>11.30-12:30</td>
                        <td>12:30-1:30</td>
                        <td>1:30-2:30</td>
                        <td>2:30-3:30</td>
                        </tr>
                        <tr>
                                <td>Day 1</td>
                                <td>web</td>
                                <td>unix</td>
                                <td rowspan="3">Break</td>
                                <td>dms</td>
                                <td>co</td>
                                <td rowspan="3">Break</td>
```

```
                    <td>ds</td>
                    <td>unix</td>
                </tr>
                <tr>
                    <td>Day 2</td>
                    <td>web</td>
                    <td>unix</td>
                    <td>dms</td>
                    <td>co</td>
                    <td>ds</td>
                    <td>unix</td>
                </tr>
                <tr>
                    <td>Day 3</td>
                    <td>web</td>
                    <td>unix</td>
                    <td>dms</td>
                    <td>co</td>
                    <td colspan="2">Lab</td>
                </tr>
            </table>
        </body>
</html>
```

Q3a) Explain the tags nav, section, article, aside, footer

1) Nav

The <nav> tag is a new element in HTML5. It is used to define a block of navigation links, either within the current document or to other documents. Examples of navigation blocks are menus, tables of contents, and indexes.

One HTML document may contain several <nav> tags, for example, one for site navigation and one for intra-page navigation.

Note that not all links in the HTML document can be placed inside the <nav> element; it can only include major navigation blocks. For example, the <nav> tag is not placed in the <footer> tag for defining links in the footer of the website.

Example:

```
<nav>
<a href="/learn-html.html">HTML</a> | <a href="/learn-css.html">CSS</a> | <a href="/learn-javascript.html">JavaScript</a> | <a href="/learn-php.html">PHP</a> |
</nav>
```

2) Section

The HTML <section> tag specifies a section in a document.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Section Tag</title>
```

```
</head>
<body>
<section>
<h1>Java</h1>

<h3>Inheritance</h3>
<p>Inheritance defines the relationship between superclass and subclass.</p>
</section>
</body>
</html>
```

3) Article

The <article> tag 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 site.

Potential sources for the <article> element:

Forum post

Blog post

News story

Comment

```
<article>
<h1>Google Chrome</h1>
<p>Google Chrome is a free, open-source web browser developed by Google, released in 2008.</p>
</article>
```

4) Aside

The <aside> tag defines some content aside from the content it is placed in.

The aside content should be related to the surrounding content.

```
<p>My family and I visited The Epcot center this summer.</p>
<aside>
<h4>Epcot Center</h4>
<p>The Epcot Center is a theme park in Disney World, Florida.</p>
</aside>
```

5) Footer

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

A <footer> element should contain information about its containing element.

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.

Q3 b) Explain the feature of HTML5

HTML 5 adds a lot of new features to the HTML specification, and it is easy it is to implement.

You use the HTML 5 doctype, which is simple and streamlined:

<!doctype html>

HTML 5 is no longer part of SGML but is instead a markup language all on its own.

The character set for HTML 5 is streamlined as well. It uses UTF-8, and you define it with just one meta tag:

<meta charset="UTF-8">

HTML 5 New Structure

HTML 5 recognizes that webpages have a structure, just like books and other XML documents have a structure. In general, webpages have navigation, body content, sidebar content, headers, footers, and other features. HTML 5 has tags to support those elements of the page. They are:

- <section> defines sections of pages.
- <header> defines the header of a page.
- <footer> defines the footer of a page.
- <nav> defines the navigation on a page.
- <article> defines the article or primary content on a page.
- <aside> defines extra content like a sidebar on a page.
- <figure> defines images that annotate an article.
-

HTML 5 New Inline Elements

The new inline elements define some basic concepts and keep them semantically marked up:

- <mark> indicates content that is marked in some fashion.
- <time> indicates content that is a time or date.
- <meter> indicates content that is a fraction of a known range such as disk usage.
- <progress> indicates the progress of a task towards completion.
-

HTML 5 New Dynamic Pages Support

HTML 5 was developed to help web application developers, so there are a lot of new features that make it easy to create dynamic HTML pages:

- Context menus – HTML 5 supports the creation and use of context menus within webpages and applications.
- href is not required on a tag. This allows you to use a tag with scripts and in web applications without needing a place to send that anchor.
- async attribute – This is added to the script tag to tell the browser that the script should be loaded asynchronously so that it doesn't slow down the load and display of the rest of the page.
- <details> – This provides details about an element. This would be like tooltips in non-web applications
- <datagrid> creates a table that is built from a database or other dynamic source.
- <menu> is an old tag brought back and given new life allowing you to create a menu system on your webpages.
- <command> defines actions that should happen when a dynamic element is activated.

HTML 5 New Form Types

HTML 5 supports all the standard form input types, but it adds a few more:

- datetime
- datetime-local
- date
- month
- week
- time
- number
- range
- email
- url

HTML 5 New Elements

There are a few exciting new elements in HTML 5:

- <canvas> – This element gives you a drawing space in JavaScript on your webpages. It can add images or graphs to tooltips or create dynamic graphs on your webpages, built on the fly.
- <video> – Add video to your webpages with this simple tag.
- <audio> – Add sound to your webpages with this simple tag.

HTML 5 Removes Some Elements

Some elements in HTML 4 are no longer be supported by HTML 5. Most are already

deprecated and shouldn't be surprising. They are:

acronym,applet, basefont, big, center, dir, font, frame, frameset, isindex, noframes, noscript, s, strike, tt, u

Q4 a) Write different selector forms in CSS

SELECTOR FORMS

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>
```

6) Pseudo Classes:

Pseudo class selectors are used if the properties are to be changed dynamically. For example: when mouse movement happens, in other words, hover happens or focus happens.

```
<head>
<style type = "text/css">
input:focus { font-family: 'lucida calligraphy'; color: purple; font-size:100; }
input:hover { font-family: 'lucida handwriting';color: violet; font-size:40; }
</style>
</head>
<body>
<form action = " ">
<p><label> NAME: <input type = "text" /></label></p>
</form>
</body>
```

Q4 b) Write XHTML and CSS document to illustrate different font properties

```
<?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">

<!-- fonts.html
     An example to illustrate font properties
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Font properties </title>
    <style type = "text/css">
      p.major {font-size: 14pt;
               font-style: italic;
               font-family: 'Times New Roman';
              }
      p.minor {font: 10pt bold 'Courier New';}
      h2 {font-family: 'Times New Roman';
          font-size: 24pt; font-weight: bold}
      h3 {font-family: 'Courier New'; font-size: 18pt}
    </style>
  </head>
  <body>
    <p class = "major">
      If a job is worth doing, it's worth doing right.
    </p>
    <p class = "minor">
      Two wrongs don't make a right, but they certainly
      can get you in a lot of trouble.
    </p>
    <h2> Chapter 1 Introduction </h2>
    <h3> 1.1 The Basics of Computer Networks </h3>
  </body>
</html>
```

```
<?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">

<!-- decoration.html
     An example that illustrates several of the
     possible text decoration values
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Text decoration </title>
    <style type = "text/css">
      p.delete {text-decoration: line-through}
      p.cap {text-decoration: overline}
      p.attention {text-decoration: underline}
    </style>
  </head>
  <body>
    <p class = "delete">
      This illustrates line-through
    </p>
    <p class= "cap">
      This illustrates overline
    </p>
    <p class = "attention">
      This illustrates underline
    </p>
  </body>
</html>
```

Q5 a) How arrays can be created in javascript ? Explain all array methods with examples.

Array OBJECT CREATION

The usual way to create any object is with the `new` operator and a call to a constructor. In the case of arrays, the constructor is named `Array`:

```
var my_list = new Array(1, 2, "three", "four");
var your_list = new Array(100);
```

The second way to create an `Array` object is with a literal array value, which is a list of values enclosed in brackets: **`var my_list_2 = [1, 2, "three", "four"];`**

Array METHODS

Array objects have a collection of useful methods, most of which are described in this section.

- The **join** method converts all of the elements of an array to strings and catenates them into a single string. If no parameter is provided to join, the values in the new string are separated by commas. If a string parameter is provided, it is used as the element separator. Consider the following example:

```
var names = new Array["Mary", "Murray",
                      "Murphy", "Max"];
...
var name_string = names.join(" : ");
```

The value of `name_string` is now `"Mary : Murray : Murphy : Max"`.

- The **reverse** method reverses the order of the elements of the Array object through which it is called.

- The **sort** method coerces the elements of the array to become strings if they are not already strings and sorts them alphabetically
- The **concat** method catenates its actual parameters to the end of the Array object on which it is called.

```
var names = new Array["Mary", "Murray",
                      "Murphy", "Max"];
...
var new_names = names.concat("Moo", "Meow");
```

The `new_names` array now has length 6, with the elements of `names`, along with `"Moo"` and `"Meow"` as its fifth and sixth elements.

- The **`slice`** method does for arrays what the `substring` method does for strings, returning the part of the `Array` object specified by its parameters, which are used as subscripts. The array returned has the elements of the `Array` object through which it is called, from the first parameter up to, but not including, the second parameter.

```
var list = [2, 4, 6, 8, 10];
...
var list2 = list.slice(1, 3);
```

- The value of `list2` is now `[4, 6]`. If `slice` is given just one parameter, the array that is returned has all of the elements of the object, starting with the specified index.

- When the **`toString`** method is called through an `Array` object, each of the elements of the object is converted (if necessary) to a string. These strings are catenated, separated by commas. So, for `Array` objects, the `toString` method behaves much like `join`.
- The **`push`**, **`pop`**, **`unshift`**, and **`shift`** methods of `Array` allow the easy implementation of stacks and queues in arrays. The `pop` and `push` methods respectively remove and add an element to the high end of an array, as in the following code:

```
var list = ["Dasher", "Dancer", "Donner", "Blitzen"];
var deer = list.pop();    // deer is "Blitzen"
list.push("Blitzen");
  // This puts "Blitzen" back on list
```

The `shift` and `unshift` methods respectively remove and add an element to the beginning of an array.

```
var deer = list.shift(); // deer is now "Dasher"

list.unshift("Dasher"); // This puts "Dasher" back on list
```

```
// nested_arrays.js
//   An example illustrating an array of arrays

// Create an array object with three arrays as its elements
var nested_array = [[2, 4, 6], [1, 3, 5], [10, 20, 30]
                   ];

// Display the elements of nested_list
for (var row = 0; row <= 2; row++) {
  document.write("Row ", row, ":  ");

  for (var col = 0; col <=2; col++)
    document.write(nested_array[row][col], " ");

  document.write("<br />");
}
```

Q5 b) How functions are declared in javascript

- A function definition consists of the function's header and a compound statement that describes the actions of the function. This compound statement is called the body of the function.
- A function header consists of the reserved word function, the function's name, and a parenthesized list of parameters if there are any.
- A return statement returns control from the function in which it appears to the function's caller. A function body may include one or more return statements. If there are no return statements in a function or if the specific return that is executed does not include an expression, the value returned is undefined.
- JavaScript functions are objects, so variables that reference them can be treated as are other object references—they can be passed as parameters, be assigned to other variables, and be the elements of an array. The following example is illustrative: Because JavaScript functions are objects, their references can be properties in other objects, in which case they act as methods.

- The following example illustrates a variable number of function parameters:

```
// params.js
//   The params function and a test driver for it.
//   This example illustrates a variable number of
//   function parameters

// Function params
// Parameters: A variable number of parameters
// Returns: nothing
// Displays its parameters
function params(a, b) {
  document.write("Function params was passed ",
      arguments.length, " parameter(s) <br />");
  document.write("Parameter values are: <br />");

  for (var arg = 0; arg < arguments.length; arg++)
    document.write(arguments[arg], "<br />");

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

// A test driver for function params
params("Mozart");
params("Mozart", "Beethoven");
params("Mozart", "Beethoven", "Tchaikowsky");
```

OUTPUT

Q6 a) Write a javascript program to show handling events from textbox and password elements

```
<?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">

<!-- nochange.html
     A document for nochange.js
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> nochange.html </title>

<!-- Script for the event handlers -->
    <script type = "text/javascript"  src = "nochange.js" >
    </script>

  </head>
  <body>
    <form action = "">
      <h3> Coffee Order Form </h3>

<!-- A bordered table for item orders -->
      <table border = "border">

<!-- First, the column headings -->
        <tr>
```

```
            <th> Product Name </th>
            <th> Price </th>
            <th> Quantity </th>
          </tr>

<!-- Now, the table data entries -->
          <tr>
            <th> French Vanilla (1 lb.) </th>
            <td> $3.49 </td>
            <td> <input type = "text"  id = "french"
                        size ="2" /> </td>
          </tr>
          <tr>
            <th> Hazlenut Cream (1 lb.) </th>
            <td> $3.95 </td>
            <td> <input type = "text"  id = "hazlenut"
                  size = "2" /> </td>
          </tr>
          <tr>
            <th> Colombian (1 lb.) </th>
            <td> $4.59 </td>
            <td> <input type = "text"  id = "colombian"
                  size = "2" /></td>
          </tr>
        </table>

<!-- Button for precomputation of the total cost -->
        <p>
          <input type = "button"  value = "Total Cost"
                 onclick = "computeCost();" />
          <input type = "text"  size = "5"  id = "cost"
                 onfocus = "this.blur();" />
        </p>

<!-- The submit and reset buttons -->
        <p>
          <input type = "submit"  value = "Submit Order" />
          <input type = "reset"  value = "Clear Order Form" />
        </p>
      </form>
    </body>
</html>
```

Q6 b) Explain DOM2 event model

The DOM 2 model is a modularized interface. One of the DOM 2 modules is Events, which includes several sub-modules. The ones most commonly used are HTMLEvents and MouseEvents. The interfaces and events defined by these modules are as follows:

Module	Event Interface	Event Types
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	click, mousedown, mousemove, mouseout, mouseover. mouseup

EVENT PROPAGATION:

- A browser which understands DOM, on receiving the XHTML document from the server, creates a tree known as document tree.
- The tree constructed consists of elements of the document except the HTML
- The root of the document tree is document object itself
- The other elements will form the node of the tree
- In case of DOM2, the node which generates an event is known as target node

- Once the event is generated, it starts the propagation from root node
- During the propagation, if there are any event handlers on any node and if it is enabled then event handler is executed
- The event further propagates and reaches the target node.
- When the event handler reaches the target node, the event handler gets executed
- After this execution, the event is again re-propagated in backward direction
- During this propagation, if there are any event handlers which are enabled, will be executed.
- The propagation of the even from the root node towards the leaf node or the target node is known as capturing phase.
- The execution of the event handler on the target node is known as execution phase.
- This phase is similar to event handling mechanism in DOM – 0
- The propagation of the event from the leaf or from the target node is known as bubbling phase
- All events cannot be bubbled for ex: load and unload event
- If user wants to stop the propagation of an event, then stop propagation has to be executed.

EVENT REGISTRATION:

- In case of DOM2, the events get registered using an API known as addEventListener
- The first arg is the eventName. Ex: click, change, blur, focus
- The second arg is the event handler function that has to be executed when there is an event
- The third arg is a Boolean argument that can either take a true or false value
- If the value is true, it means event handler is enabled in capturing phase
- If the event value if off (false), then event handler is enabled at target node
- The addEventListener method will return event object to eventhandler function. The event object can be accessed using the keyword “Event”
- The address of the node that generated event will be stored in current target, which is property of event object
-

Q7 a) Explain the ways of positing the elements in dynamic document with javascript with suitable example

The position property has three possible values: absolute, relative, and static.

1)ABSOLUTE POSITIONING

- The absolute value is specified for position when the element is to be placed at a specific place in the document display without regard to the positions of other elements.
- One use of absolute positioning is to superimpose special text over a paragraph of ordinary text to create an effect similar to a watermark on paper.
- A larger italicized font, in a light-gray color and with space between the letters, could be used for the special text, allowing both the ordinary text and the special text to be legible.

Example

```
<?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">

<!-- absPos.html
     Illustrates absolute positioning of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Absolute positioning </title>
    <style type = "text/css">
```

```
/* A style for a paragraph of text */
      .regtext {font-family: Times; font-size: 14pt; width: 600px}

/* A style for the text to be absolutely positioned */
     .abstext {position: absolute; top: 25px; left: 50px;
               font-family: Times; font-size: 24pt;
               font-style: italic; letter-spacing: 1em;
               color: rgb(102,102,102); width: 500px}
    </style>
  </head>
  <body>
    <p class = "regtext">
      Apple is the common name for any tree of the genus Malus, of
      the family Rosaceae. Apple trees grow in any of the temperate
      areas of the world. Some apple blossoms are white, but most
      have stripes or tints of rose. Some apple blossoms are bright
      red. Apples have a firm and fleshy structure that grows from
      the blossom. The colors of apples range from green to very
      dark red. The wood of apple trees is fine-grained and hard.
      It is, therefore, good for furniture construction. Apple trees
      have been grown for many centuries. They are propagated by
      grafting because they do not reproduce themselves.
    </p>
    <p class = "abstext">
      APPLES ARE GOOD FOR YOU
    </p>
  </body>
</html>
```

2)RELATIVE POSITIONING

- An element that has the position property set to relative, but does not specify top and left property values, is placed in the document as if the position attribute were
- not set at all.
- However, such an element can be moved later.
- If the top and left properties are given values, they displace the element by the specified amount from the position where it would have been placed.
- In both the case of an absolutely positioned element inside another element and the case of a relatively positioned element, negative values of top and left displace the element upward and to the left, respectively.
- Relative positioning can be used for a variety of special effects in placing elements.

Example

```
<?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">

<!-- relPos.html
     Illustrates relative positioning of elements
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Relative positioning </title>
  </head>
  <body style = "font-family: Times; font-size: 24pt;">
    <p>
      Apples are <span style =
                "position: relative; top: 10px;
                 font-family: Times; font-size: 48pt;
                 font-style: italic; color: red;">
      GOOD </span> for you.
    </p>
  </body>
</html>
```

3)STATIC POSITIONING

- The default value for the position property is static.
- A statically positioned element is placed in the document as if it had the position value of relative but no values for top or left were given.
- The difference is that a statically positioned element cannot have its top or left properties initially set or changed later.
- Therefore, a statically placed element cannot be displaced from its normal position and cannot be moved from that position later.

Q7 b) write a program to implement stacking of elements

```
<?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">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head><title>Program 7a</title>
        <script type="text/javascript">
                var oldid="p3";
                function toTop(topid){
                        domTop=document.getElementById(oldid).style;
                        domNew=document.getElementById(topid).style;
                        domTop.zIndex="0";
                        domNew.zIndex="10";
                        oldid=topid;
                }
                function toDown(downid){
                        document.getElementById(downid).style.zIndex="0";
                }
        </script>
<style type="text/css">
.para1{
        position:absolute;
        top:0px;
```

```
        left:0px;
        z-index:0;
        width:200px;
        height:200px;
        background-color:red;
}
.para2{
        position:absolute;
        top:100px;
        left:100px;
        z-index:0;
        width:200px;
        height:200px;
        background-color:blue;
}
.para3{
        position:absolute;
        top:200px;
        left:200px;
        z-index:0;
        width:200px;
        height:200px;
        background-color:green;
}
</style>
</head>
<body>
<p class="para1" id="p1" onmouseover="toTop('p1')" onmouseout="toDown('p1')"> Frame One
</p>
<p class="para2" id="p2" onmouseover="toTop('p2')" onmouseout="toDown('p2')"> Frame Three
</p>
<p class="para3" id="p3" onmouseover="toTop('p3')" onmouseout="toDown('p3')"> Frame Two
</p>
</body>
</html>
```

Q8 a) Explain the syntax of XML

- XML imposes two distinct levels of syntax:
- There is a general low level syntax that is appreciable on all XML documents
- The other syntactic level is specified by DTD (Document Type Definition) or XML schemas.
- The DTDs and XML schemas specify a set of tag and attribute that can appear in a particular document or collection of documents.
- They also specify the order of occurrence in the document.
- The XML documents consists of data elements which form the statements of XML document.

- The XML document might also consists of markup declaration, which act as instructions to the XML parser
- All XML documents begin with an XML declaration. This declaration identifies that the document is a XML document and also specifies version number of XML standard.
- It also specifies encoding standard.
  <?xml version = “1.0” encoding = “utf-8”?>
- Comments in XML is similar to HTML
- XML names are used to name elements and attributes.
- XML names are case-sensitive.
- There is no limitation on the length of the names.
- All XML document contains a single root element whose opening tag appears on first line of the code
- All other tags must be nested inside the root element
- As in case of XHTML, XML tags can also have attributes
- The values for the attributes must be in single or double quotation
  Example:
  <?xml version = “1.0” encoding = “utf-8”?>
  <student>
  <name>XYZ</name>
  <usn>1CY17MCA01</usn>
  </student>

  Tags with attributes

  The above code can be also written as
  <student name = “XYZ” usn = “1CY17MCA01”> </student>

Q8 b) How XML documents can be written with XSLT form explain

An XML document that is to be used as data to an XSLT style sheet must include a processing instruction to inform the XSLT processor that the style sheet is to be used. The form of this instruction is as follows:

<?xml-stylesheet type=”text/xsl” href=”XSL_stylesheet_name”?>

An XSLT style sheet is an XML document whose root element is the special-purpose element stylesheet. The stylesheet tag defines namespaces as its attributes and encloses the collection of elements that defines its transformations. It also identifies the document as an XSLT document.

<xsl:stylesheet version="1.0"

xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>

In many XSLT documents, a template is included to match the root node of the XML document.
<xsl:template match="/">

In many cases, the content of an element of the XML document is to be copied to the output document.

This is done with the value-of element, which uses a select attribute to specify the element of the XML document whose contents are to be copied. <xsl:value-of select="name"/> The select attribute can specify any node of the XML document. This is an advantage of XSLT formatting over CSS, in which the order of data as stored is the only possible order of display.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplane2.xsl
     An XSLT Stylesheet for xslplane.xml using implicit templates
     -->
<xsl:stylesheet version = "1.0"
                xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
                xmlns = "http://www.w3.org/1999/xhtml">

<!-- The template for the whole document (the plane element) -->
     <xsl:template match = "/">
     <html><head><title> Style sheet for xslplane.xml </title>
     </head><body>
     <h2> Airplane Description </h2>
     <span style = "font-style: italic; color: blue;"> Year:
     </span>
     <xsl:value-of select = "year" /> <br />
     <span style = "font-style: italic; color: blue;"> Make:
     </span>
     <xsl:value-of select = "make" /> <br />
     <span style = "font-style: italic; color: blue;"> Model:
     </span>
     <xsl:value-of select = "model" /> <br />
     <span style = "font-style: italic; color: blue;"> Color:
     </span>
     <xsl:value-of select = "color" /> <br />
     </body></html>
   </xsl:template>
</xsl:stylesheet>
```

Q8 c) Differentiate between XML and HTML

Parameter	XML	HTML
Type of language	XML is a framework for specifying markup languages.	HTML is predefined markup language.
Language type	Case sensitive	Case insensitive
Structural details	It is provided	It is not provided.
Purpose	Transfer of data	Presentation of the data
Coding Errors	No coding errors are allowed.	Small errors are ignored.
Whitespace	You can use whitespaces in your code.	You can't use white spaces in your code.
Nesting	Should be done appropriately.	Does not have any effect on the code.
Driven by	XML is content driven	HTML is format driven
End of tags	The closing tag is essential in a well-formed XML document.	The closing tag is not always required. <HTML> tag needs an equivalent </HTML> tag but tag does not require </br> tag
Quotes	Quotes required around XML attribute values.	Quotes are not required for the values of attributes.
Object support	Objects have to be expressed by conventions. Mostly using attributes and elements.	Offers native object support

Null support	Need to use xsi:nil on elements in an XML instance document and also need to import the corresponding namespace.	Natively recognizes the null value.
Namespaces	XML provides support for namespaces. It helps you to remove the risk of name collisions when combining with other documents.	Does not support the concept of namespaces. Naming collisions can be avoided either using a prefix in an object member name or by nesting objects.
Formatting decisions	Require more significant effort to map application types to XML elements and attributes.	Provides direct mapping for application data.
Size	Documents are mostly lengthy in size, especially when an element-centric approach used in formatting.	The syntax is very brief and yields formatted text.
Parsing in Javascript	Requires an XML DOM implementation and application code to map text back into JavaScript objects.	No extra application code required to parse text. For this purpose, you can use the eval function of JavaScript.
Learning curve	Very hard as you need to learn technologies like XPath, XML Schema, DOM, etc.	HTML is a simple technology stack that is familiar to developers.

Q9 a) Explain different jquery wrapper

When CSS was introduced to web technologies in order to separate design from content, a way was needed to refer to groups of page elements from external style sheets. The method developed was through the use of selectors, which concisely represent elements based upon their attributes or position within the HTML document.

For example, the selector
p a

refers to the group of all links (<a> elements) that are nested inside a <p> element. jQuery makes use of the same selectors, supporting not only the common selectors currently used in CSS, but also the more powerful ones not yet fully implemented by most browsers.
To collect a group of elements, we use the simple syntax

$(selector)
or
jQuery(selector)

For example, to retrieve the group of links nested inside a <p> element, we use the following

$("p a")

The $() function (an alias for the jQuery() function) returns a special Java-Script object containing an array of the DOM elements that match the selector.
This object possesses a large number of useful predefined methods that can act on the group of elements.
In programming parlance, this type of construct is termed a wrapper because it wraps the matching

element(s) with extended functionality. The term jQuery wrapper or wrapped set to refer to this set of matched elements that can be operated on with the methods defined by jQuery.

Example
$("div.notLongForThisWorld").fadeOut();
$("div.notLongForThisWorld").fadeOut().addClass("removed");

Q9 b) Explain the following commands: size(), get(), index(), not(), add()

I. Determining the size of the wrapped set

Command syntax: size

size()

Returns the count of elements in the wrapped set

Parameters
none

Returns
The element count

Consider the following statement:

```
$('#someDiv') .html('There are '+$('a').size()+' link(s) on this page.');
```

The inner jQuery wrapper matches all elements of type <a> and returns the number of matched elements using the size() method.

II. Obtaining elements from the wrapped set

Because jQuery allows us to treat the wrapped set as a JavaScript array, we can use simple array indexing to obtain any element in the wrapped list by position.
If we prefer to use a method rather than array indexing, jQuery defines the **get() method** for that purpose.

Command syntax: get

get(index)

Obtains one or all of the matched elements in the wrapped set. If no parameter is specified, all elements in the wrapped set are returned in a JavaScript array. If an index parameter is provided, the indexed element is returned.

Parameters
index (Number) The index of the single element to return. If omitted, the entire set is returned in an array.

Returns
A DOM element or an array of DOM elements.

The get() method can also be used to obtain a plain JavaScript array of all the wrapped elements. Consider:

```
var allLabeledButtons = $('label+button').get();
```

index() command is used to find the index of a particular element in the wrapped set.

Command syntax: index

index(element)

Finds the passed element in the wrapped set and returns its ordinal index within the set. If the element isn't resident in the set, the value -1 is returned.

Parameters

`element` (Element) A reference to the element whose ordinal value is to be determined.

Returns

The ordinal value of the passed element within the wrapped set or -1 if not found.

Slicing and dicing the wrapped element set

Once we have a wrapped element set, we may want to augment that set by adding to it or by reducing the set to a subset of the originally matched elements. jQuery gives us a large collection of methods to manage the set of wrapped elements.

Adding more elements to the wrapped set

Often, we may find ourselves in a situation where we want to add more elements to an existing wrapped set. This capability is most useful when we want to add more elements after applying some command to the original set

```
$('img[alt]').add('img[title]')
```

Using the add() method in this fashion allows us to chain a bunch of selectors together into an or relationship, creating the union of the elements that satisfy both of the selectors. Methods such as add() can also be useful in place of selectors in that the end() method can be used to back out of the elements added via add().

Command syntax: add

add(expression)

Adds elements, specified by the `expression` parameter, to the wrapped set. The expression can be a selector, an HTML fragment, a DOM element, or an array of DOM elements.

Parameters

`expression` (String|Element|Array) Specifies what is to be added to the matched set. This parameter can be a jQuery selector, in which case any matched elements are added to the set. If an HTML fragment, the appropriate elements are created and added to the set. If a DOM element or an array of DOM elements, they are added to the set.

Returns

The wrapped set.

Example :

```
$('img[alt]').addClass('thickBorder').add('img[title]').addClass('seeThrough')
```

The add() method can also be used to add elements to an existing wrapped set given references to those elements. Passing an element reference, or an array of

element references, to the add() method adds the elements to the wrapped set.

```
$('img[alt]').add(someElement)
```

The add() method not only allows us to add existing elements to the wrapped set, we can also use it to add new elements by passing it a string containing HTML markup.

```
$('p').add('<div>Hi there!</div>')
```

Honing the contents of the wrapped set

By using the **not() method**, which removes any elements from a wrapped set that match the passed selector expression, we can express an except type of relationship.

Command syntax: not

`not(expression)`

Removes elements from the matched set according to the value of the `expression` parameter. If the parameter is a jQuery filter selector, any matching elements are removed. If an element reference is passed, that element is removed from the set.

Parameters

`expression` (String|Element|Array) A jQuery filter expression, element reference, or array of element references defining what is to be removed from the wrapped set.

Returns

The wrapped set.

As with add(), the not() method can also be used to remove individual elements from the wrapped set by passing a reference to an element or an array of element references.

At times, we may want to filter the wrapped set in ways that are difficult or impossible to express with a selector expression. In such cases, we may need to resort to programmatic filtering of the wrapped set items.

The **filter() method**, when passed a function, invokes that function for each wrapped element and removes any element whose function invocation returns the value false. Each invocation has access to the current wrapped element via the function context (this) in the body of the filtering function.

Command syntax: filter

`filter(expression)`

Filters out elements from the wrapped set using a passed selector expression, or a filtering function.

Parameters

`expression` (String|Function) Specifies a jQuery selector used to remove all elements that do not match from the wrapped set, or a function that makes the filtering decision. This function is invoked for each element in the set, with the current element set as the function context for that invocation. Any element that returns an invocation of `false` is removed from the set.

Returns

The wrapped set.

Q10 a) How attr() command can be used in a different ways in manipulating element.

There are two ways to set attributes onto elements in the wrapped set with jQuery.

Command syntax: attr

`attr(name,value)`

Sets the named attribute onto all elements in the wrapped set using the passed value.

Parameters

`name`	(String) The name of the attribute to be set.
`value`	(String|Object|Function) Specifies the value of the attribute. This can be any JavaScript expression that results in a value, or it can be a function. See the following discussion for how this parameter is handled.

Returns

The wrapped set.

Command syntax: attr

`attr(attributes)`

Sets the attributes and values specified by the passed object onto all elements of the matched set

Parameters

`attributes`	(Object) An object whose properties are copied as attributes to all elements in the wrapped set

Returns

The wrapped set

Q10 b) Explain the following commands bind(), one(), unbind(), trigger(), eventName()

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.

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.

I. 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.

V. Triggering event handlers

jQuery has provided means to assist us in avoiding top-level functions by defining a series of methods that will automatically trigger event handlers on our behalf under

script control. The most general of these commands is trigger(), whose syntax is as follows:

Command syntax: trigger

`trigger(eventType)`

Invokes any event handlers established for the passed event type for all matched elements

Parameters

`eventType`	(String) Specifies the name of the event type for handlers which are to be invoked

Returns

The wrapped set

The trigger() command does not cause an event to be triggered and to propagate through the DOM hierarchy. As there's no dependable cross-browser means to generate an event, jQuery calls the handlers as normal functions.
Each handler called is passed a minimally populated instance of Event.
Because there's no event, properties that report values, such as the location of a mouse event, have no value. The target property is set to reference the element of the matched set to which the handler was bound.
Also because there's no event, no event propagation takes place. The handlers bound to the matched elements will be called, but no handlers on the ancestors of those elements will be invoked.

In addition to the trigger() command, jQuery provides convenience commands for most of the event types. The syntax for all these commands is exactly the same except for the command name, and that syntax is described as follows:

Command syntax: *eventName*

`eventName``()`

Invokes any event handlers established for the named event type for all matched elements.
The supported commands are as follows:

- `blur`
- `click`
- `focus`
- `select`
- `submit`

Parameters

`none`

Returns

The wrapped set.

In addition to binding, unbinding, and triggering event handlers, jQuery offers high-level functions that further make dealing with events on our pages as easy as possible.