# Building Responsive Websites with HTML5

# *Building Responsive Websites with HTML5*
# Learner's Guide

**© 2020 Aptech Limited**

All rights reserved.

No part of this book may be reproduced in any manner whatsoever, stored in a retrieval system or transmitted or translated in any form or manner or by any means, without the prior written permission of APTECH LIMITED.

All trademarks acknowledged.

**APTECH LIMITED**

Contact E-mail: ov-support@onlinevarsity.com

First Edition: February 2020

# Preface

HyperText Markup Language (HTML) is the markup language popularly used by developers to design Web sites. It is the language for the World Wide Web. It is used for building Web sites that can be as large as a corporate Web site or as small as a single page classroom project.

This book covers basic to advanced concepts of HTML5. The book begins with an explanation of basic HTML tags and attributes. It also explains the structure of Web pages using HTML. Then, it proceeds to explain the concept of Cascading Style Sheets (CSS). CSS is a technology that helps Web site designers to provide a consistent formatting across large Web sites by separating the content from its styles. Thus, CSS is used to control the look of the Web page by specifying the styles such as color, font, and font size for the HTML content. It can also be used to control the placement of items on a page. The book also explains JavaScript, which is a scripting language used for adding interactivity to Web pages. JavaScript allows programs in an HTML page to respond to user's actions. These responses could be validating the user's input, fetching and displaying the requested page, and so on. Finally, the book concludes with explanations of jQuery and HTML5 mobile application support. jQuery is a short and fast JavaScript library that simplifies the client side scripting of HTML, animation, event handling, traversing, and developing AJAX based Web applications.

The knowledge and information in this book is the result of the concentrated effort of the Design Team, which is continuously striving to bring to you the latest, the best and the most relevant subject matter in Information Technology. As a part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends and learner requirements.

We will be glad to receive your suggestions. Please send us your feedback, addressed to the Design Centre at Aptech's corporate office, Mumbai.

Design Team

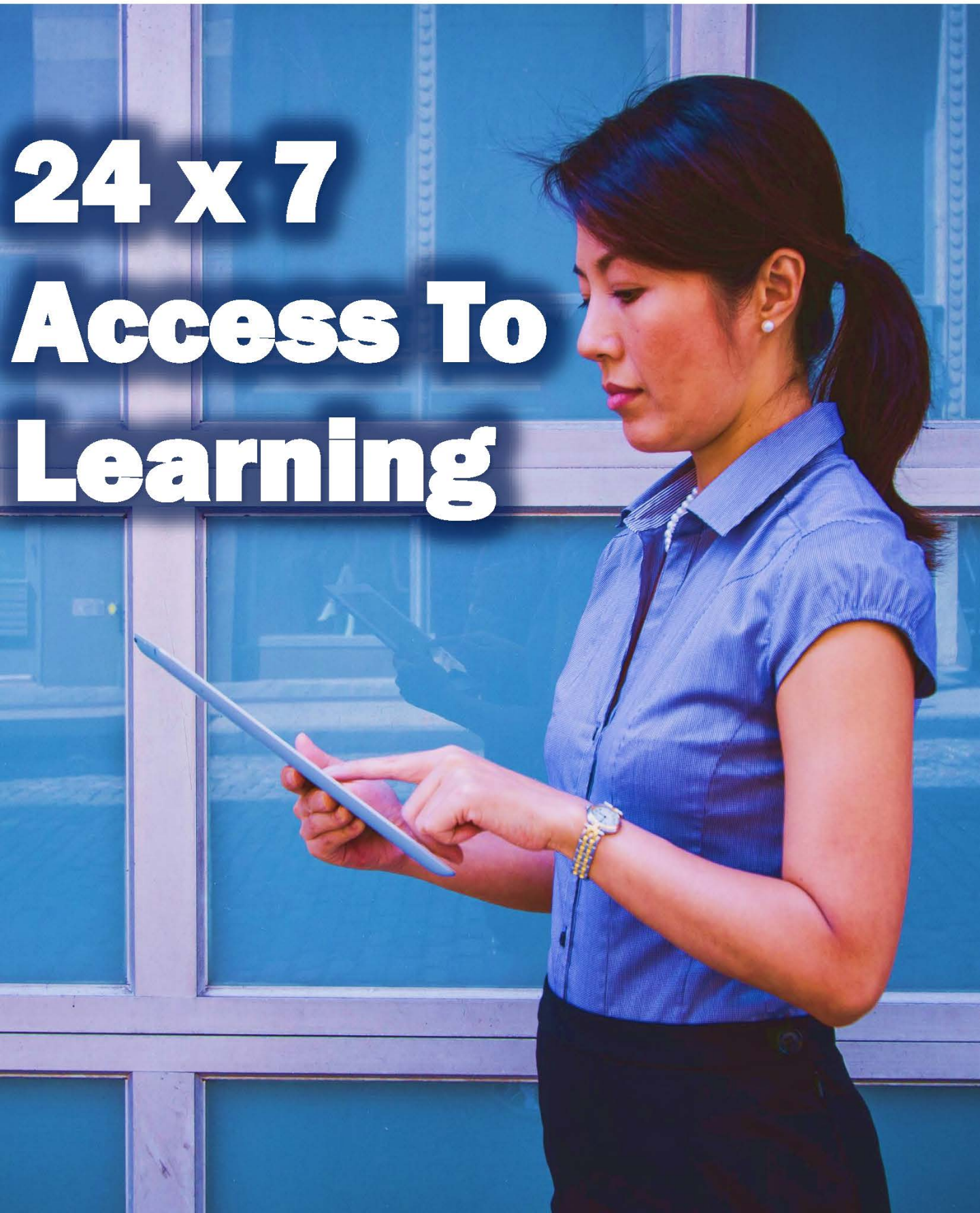
# MANY COURSES ONE PLATFORM

# Table of Contents

## Sessions


1. Introduction to the Web
2. Introduction to HTML5
3. Formatting Text Using Tags
4. Creating Hyperlinks and Anchors
5. Introduction to CSS3
6. Formatting Using Style Sheets
7. Displaying Graphics and CSS3 Animation
8. Creating Navigational Aids and Division-Based Layout
9. Creating Tables
10. HTML Forms
11. HTML5 Audio and Video
12. Introduction to JavaScript
13. Basics of jQuery
14. Functions, Widgets, and Plugins in jQuery
15. Operators and Statements
16. Loops and Arrays
17. Functions and Objects
18. Building a Mobile Web Application
19. Canvas and JavaScript
20. HTML5 Web Storage
21. HTML5 Geolocation and APIs

Onlinevarsity



Onlinevarsity

Onlinevarsity App

for Android devices

Download from Google Play Store

# Session - 1

# Introduction to the Web

Welcome to the Session, **Introduction to the Web**.

This session explains the evolution, page structure, and features of HTML5. This session also explains CSS, JavaScript, and jQuery. Finally, the session explains the browser support provided for HTML5.

In this Session, you will learn to:

- Explain the evolution of HTML
- Explain the page structure used by HTML
- List the drawbacks in HTML 4 and XHTML
- List the new features of HTML5
- Explain CSS
- Explain JavaScript
- Explain jQuery
- Explain browser support for HTML5

## *1.1 Introduction*

HyperText Markup Language (HTML) was introduced in the year 1990. Since then, there has been continuous evolution in the technology leading to introduction of new versions. Some features were introduced in specifications, whereas others were introduced in software releases.

HTML 4 was recommended as a standard by W3C in 1997. HTML5 is the next version of HTML and will be the new standard. HTML 4.01 was the previous version of HTML which was released in 1999. Since then, there have been constant evolutions and additions to the World Wide Web (WWW). Majority of the browsers support HTML5 elements and Application Programming Interfaces (APIs).

### *1.1.1 Evolution of Computing*

The era of computing started with the use of stand-alone computers to carry out different computing operations. These computers were isolated and not connected to each other.

Eventually as the years passed, the growth of computing expanded in multiple diverse fields such as business, education, and military due to quick data processing. With such a huge expansion, organizations felt the need of sharing the processed data among their people to save time and effort. This marked the beginning of computer networks. Organizations began to connect their computers and share data amongst their people. These networks are as follows:

- Local Area Network (LAN)
- Metropolitan Area Network (MAN)
- Wide Area Network (WAN)

LAN refers to a computer network in a small geographical area such as office, home, or school. MAN refers to a network that covers a city. WAN refers to a network that connects LANs and MANs across the globe.

Figure 1.1 shows the evolution of computing.

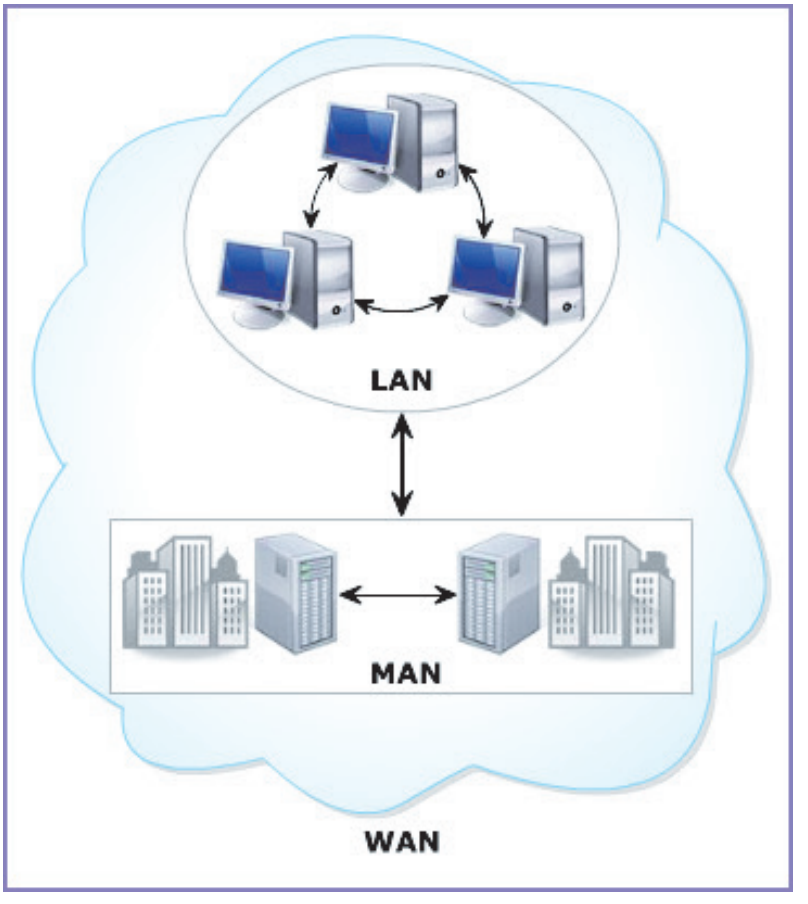



Figure 1.1: Evolution of Computing

## *1.1.2 Web and Internet*

The advent of WANs raised a strong need to share data across the globe rather than just sharing the data within the organization. This is because organizations can share their problems, solutions, experiences, and updates along with other organizations and customers. This would facilitate faster analysis and decision-making process. This resulted in the evolution of the Web, also referred as World Wide Web or WWW. Therefore, Internet is known as the largest WAN.

The Web is a way to access information using the Internet that is referred to as a network of networks. Here, multiple computers are connected to each other irrespective of their geographical locations. Information is made available across the globe in the form of Web pages.

A Web page is a file that contains information and instructions to display the information to the users. Figure 1.2 shows the relation between Web and Internet.

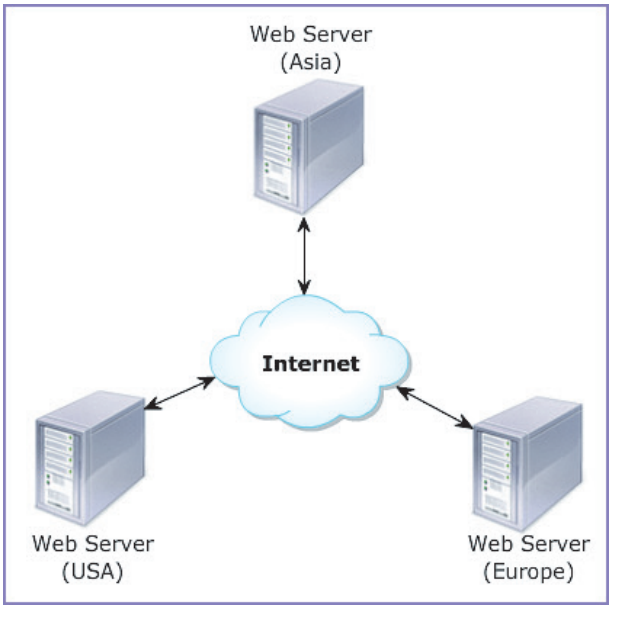



**Figure 1.2: Web and Internet**

## *1.1.3 Web Communication*

Web pages are stored on a Web server to make them available on the Internet so that users can view them. A Web server is a computer with high processing speed connected to the Internet and is used to host Web pages. Web browsers such as Microsoft Internet Explorer or Netscape Navigator are used to interpret and display the Web pages using a protocol (set of rules). The most popular protocol used to view Web pages is Hypertext Transfer Protocol (HTTP). It is a protocol that specifies how a Web page will be retrieved from the Web Server.

The steps to view a Web page in a browser are as follows:

1. The user specifies the Uniform Resource Locator (URL) of the Web page in a browser.
2. The client browser sends the URL request to the appropriate Web server.

3. The Web server processes the request and sends the Web page as a response to the browser as shown in figure 1.3.

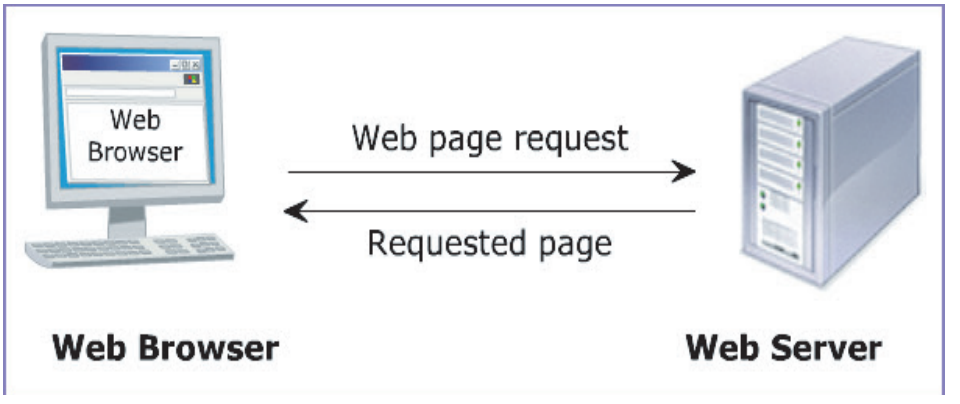



Figure 1.3: Web Communication

**Note** - A URL is a unique path of a Web page and it is in the form of `[protocol]://[Web server name]/[path of the file to be searched]`.

## *1.1.4 Static Web Pages*

Evolution of Web started with the creation of static Web pages. A static Web page consists of contents specified by the Web page designer. These contents include text, images, audios, and videos. The same content is displayed to each user on the Internet each time the page is requested.

Therefore, such a Web page is called a static Web page, as the contents of the Web page remain unchanged. The only way to update a static Web page is to change the content manually.

Static Web pages are developed using HTML. HTML is a language used to create Web pages that include hypertext along with other contents. Hypertext refers to the content in a Web page, which is linked to another Web page. The advantages of static Web pages are as follows:

- Simple to design as it does not provide interactivity. This leads to reduction in programming effort and complexity of Web pages.

- Ideal for specifying the content appearance as static Web pages focus on content presentation, which is one-way flow of information.

### *1.1.5 Dynamic Web Pages*

Static Web pages are easy to design and develop but they have few limitations. Firstly, static Web pages are difficult to maintain, as they need to be updated manually. This raises the risk of inconsistency and incomplete content in static Web pages, which might make them unreliable. Secondly, they do not allow any user interaction. This means that users can only view and read the contents. These limitations led to the need of dynamic Web pages.

A dynamic Web page generates content 'on-demand' when user provides certain inputs. It accepts the inputs from the user based on which it displays the content in the browser. Consider an example of an online store where the users can buy different products by selecting them online. Based on the selected products (input), a page with the total cost is displayed to the user.

### *1.1.6 Technologies*

Dynamic Web sites can include Web pages containing static as well as dynamic content. The main advantage of using dynamic Web page is that, it allows customizing the content and its appearance in the browser. A dynamic Web site interacts with the database to generate dynamic content. A database is a collection of organized data. This data can be stored and retrieved using the technologies used in creating dynamic Web sites. Some of the technologies used for creating dynamic Web sites are as follows:

- **JavaScript**

  JavaScript is a scripting language developed by Netscape for creating dynamic Web pages. It is used to develop interactive Web pages by adding programming to HTML.

- **Cascading Style Sheets (CSS)**

  CSS are style sheets that specify the formatting of a Web page for both static and dynamic Web pages. The formatting options include font, color, background, spacing, positioning, and borders. It is used in combination with JavaScript to format Web pages dynamically.

- **Extensible HTML (XHTML)**

  XHTML is a language that combines HTML with Extensible Markup Language (XML). XML allows defining your own data in a structured format, which can be displayed in any browser. When you use XHTML with JavaScript, the required user-defined data is displayed each time the Web page is loaded in the browser.

- **Dynamic HTML (DHTML)**

  Dynamic HTML (DHTML) uses JavaScript and CSS to make dynamic Web pages. It allows you to transform the look and feel of Web pages. It allows Web pages to respond to the user's actions and enables focus on the content changes in the browser.

## *1.2 History*

HTML is a markup language used primarily to create hypertext Web pages, which are published on the Web and displayed in any Web browser. A markup language is a set of notations that specifies how the content should look in the browser. HTML is derived from Standard Generalized Markup Language

(SGML), which is the mother language of HTML. SGML is a markup language that defines the structure of other markup languages.

HTML has evolved over the years with the introduction of improved set of standards and specifications. HTML 1.0 was the first version of HTML introduced in 1993. At that time, there were very less people involved in designing Web sites. HTML 2.0 was introduced in 1995 and included the complete HTML 1.0 specifications with additional features. The other versions are as follows:

➔ **HTML 3.0**

HTML 3.0 specifications included new features for the Netscape Navigator browser as it became very popular. The new improvements did not work on any other browsers such as Internet Explorer. Therefore, this specification was abandoned.

➔ **HTML 3.2**

Additional browser-specific features revolutionized the need for standardization of HTML. Therefore, the World Wide Consortium (W3C) organization was formed to specify and maintain the HTML standards. HTML 3.2 was the first specification introduced by W3C in January 1997 and was fully supported by all the Web browsers.

➔ **HTML 4.0**

W3C introduced HTML 4.0 in December 1997 with the motive for facilitating support for CSS, DHTML, and JavaScript. However, HTML 4.0 prevailed for a short period and was revised, which led to HTML 4.01 specification in 1999.

➔ **HTML5**

HTML5 is cooperative project between the W3C and the Web Hypertext Application Technology Working Group (WHATWG).

W3C was busy working with XHTML 2.0 and WHATWG was working with Web forms, new HTML features, and applications. In 2006, the two groups decided to work together and develop a new version of HTML.

Some basic rules for HTML5 that were established are as follows:

- Introduction of new features should be based on HTML, CSS, DOM, and JavaScript
- More markup should be used to replace scripting
- HTML5 must be device independent
- Need for external plug-ins, such as Flash, to be reduced
- Better error handling capabilities
- Development process should be completely visible to the public

## *1.3 Layout of a Page in HTML5*

The basic structure of a HTML5 document remains the same. Each HTML5 page consists of a head section containing unseen elements and links and a body section where the visible elements of the document are present.

HTML offers different tags to build and organize the content in the body of the document. The body structure generates the visible part of the document. One of the elements provided for body is `<table>` tag.

Tables help in improving user's experience by helping the user to visualize the Web site in an organized manner. Eventually, other elements replaced the function of tables. These elements have lesser code and are faster, thus facilitating creation, portability, and maintenance of a HTML5 Web site.

The `<div>` element was another element that was introduced in this field. With the integration of HTML, CSS, and JavaScript, and the usage of more interactive applications, the `<div>` tag was frequently used. Both the `<div>` and `<table>` elements, did not provide information about the sections of the body that the element may be representing. Content such as scripts, images, links, text, menus, forms, and so on could be used between the opening and closing `<div>` tags.

HTML5 includes new elements that identify and organize each part of the document body. In HTML5, the most significant sections of a document are separated and marked. Hence, the main structure does not depend on the `<div>` or `<table>` tags.

A typical HTML page would have a page header, footer, and middle page content. Within the middle page content, at the top level, it may have navigation, content, and aside columns. Also, within the content, more sections can be embedded depending on the page's specific content.

Figure 1.4 shows the regular layout of a Web page in HTML5.

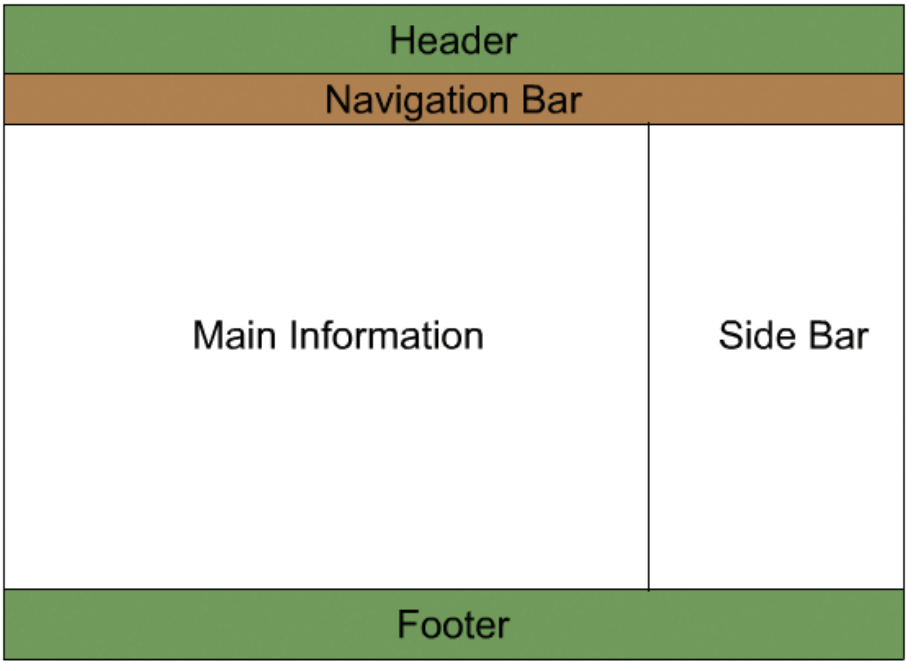



**Figure 1.4: Layout of a Web Page in HTML5**

The **Header** on the top usually has the logo, name, subtitles, and short descriptions of the Web site or Web page.

After that is the **Navigation Bar** that includes a menu that contains links for navigation. Web users can navigate to different pages or documents using the navigation bar.

The most relevant content is generally shown in the middle of the page. The content presented in the **Main Information** part of the layout usually has a top priority. It can have a list of products, description of products, blogs, or any other important information.

The **Side Bar** shows a list of links that lead to certain items that may be present internally on the Web site or on an external link. For example, in a blog, the last column offers a list of links that can lead to the blog entries, information about the author, and so on. These two sections are extremely flexible. Web designers can perform variety of actions, such as inserting more rows or splitting the columns, to edit the Web page as required.

The **footer** at the bottom is used to represent general information about the Web site. This can include the author or the company name, links regarding rules, terms and conditions, location maps, and any other additional data.

## *1.4 Drawbacks in HTML 4 and XHTML*

HTML 4 was a standard that was universally accepted for developing Web sites. It is a very stable coding language which ignores small coding errors. Also, HTML 4 is majorly compatible with all important browsers.

HTML5 adds new capabilities to the previous version of HTML. It is a revised code build on the basis of HTML 4. HTML5 provides major improvement through better interactivity, multimedia services, and application handling. For example, you can directly play a video or audio on any browser without the need to install external plug-ins such as Flash or Silverlight.

XHTML was a version of HTML 4 that was created along with XML. It was a rigid, standards-based language that allowed no room for errors. XHTML was supposed to be the next version of HTML 4 but due to interoperability problems, it took a backseat and HTML5 would be the next standard for Web site development.

## *1.5 New and More Flexible Approach of HTML5*

There are various rumors with regards to HTML5 currently. It can help a simple Web site developer to even a game developer. HTML5 has different aspects for everyone. Some of the aspects of HTML are as follows:

- For a multimedia person, HTML5 gets rid of plug-ins and uses new native support for audio and video.
- For a Web designer, HTML5 provides descriptive semantics.
- For a programmer, HTML5 helps to create rich Internet clients. These clients can be built without using plug-ins such as Flash. For this, you can use canvas and JavaScript to create better interfaces and animations. Canvas is a rectangular area on the Web page that uses JavaScript. A developer can control every single pixel in the area. The canvas element has several ways to draw paths, rectangles, filled rectangles, circles, images, and so on.
- For a client-side programmer, the Web workers is one of the features provided that can make JavaScript more efficient. Web workers is a JavaScript based API that is used to run background scripts in a Web application. This helps to mitigate the effect of the background script affecting the main process that is being executed.
- For database administrator, HTML5 has client-side storage and caching functionality.
- For a design expert, CSS in HTML5 has been improved by added features such as advanced selectors, animations, drop-shadows, and so on.
- For a mobile programmer, a lot of features are included for mobile applications.

HTML5 is a family of technologies that gives whole new options for building Web pages and applications.

## *1.5.1 Working of HTML5*

HTML5 is made up of a family of technologies. HTML consists of markups, improved CSS with CSS3 that provides added options to style your pages. There is also JavaScript and a new set of JavaScript APIs that are available in HTML5.

The process generally followed for HTML5 is as follows:

1. The browser loads the document, which includes HTML markup and CSS style.
2. After the browser loads the page, it also creates an internal model of the document that contains all the elements of HTML markup.
3. The browser also loads the JavaScript code, which executes after the page loads.
4. The APIs give access to audio, video, 2D drawing with the canvas, local storage, and other technologies that are required to build apps.

## *1.6 New Features of HTML5*

Some of the new features introduced in HTML5 are as follows:

- The `<canvas>` element is used for 2D drawing.
- New content-specific elements, such as `<article>`, `<nav>`, `<header>`, `<footer>`, `<section>`, and so on helps to structure the document.
- HTML5 has local storage support.
- The `<audio>` and `<video>` elements is available for media playback.
- New form controls, such as calendar, date, time, e-mail, URL, search, and so on have been provided by HTML5.
- The Web workers API is added to support background processes without disturbing the main process. The common problems faced by Web applications are slow performance when a large set of data is processed. This is due to the fact that all the processes are executed in a single thread. Web workers help to solve this problem.
- The Web Sockets API provides a continuous connection between a server and a client by using a specific port. Thus, the Web applications become efficient as the data can be easily exchanged between client and server without reloading the page constantly.
- Easier access to location specific data which is made available by devices having Global Positioning System (GPS) capabilities. This improved functionality is achieved with the help of API.

➔ HTML5 allows Web applications to be executed offline by storing the files and other resources required in the application cache. Web application data is saved locally using Web SQL databases.

## *1.7 Cascading Style Sheets*

HTML5 along with CSS and JavaScript forms an integrated instrument. CSS is basically a language that works along with HTML to provide visual styles to the elements of the document, such as size, color, backgrounds, borders, and so on.

A style sheet is a collection of rules that specifies the appearance of data in an HTML document. HTML is a markup language that focuses only on the layout of the content on a Web page. However, applying layouts to more than one occurrence of an HTML element in an HTML page is a tedious job.

For example, if you want to change the text in the H2 element to bold, this has to be done manually for all the H2 elements. Such a manual task might result into human errors such as missing an occurrence of the H2 element for applying the bold format. This results in format inconsistency among the H2 elements within an HTML page. Further, the specified formatting might not have same appearance across various devices such as computers and mobiles.

Style sheets overcome these problems by specifying the formatting instructions in a separate file as shown in figure 1.5.

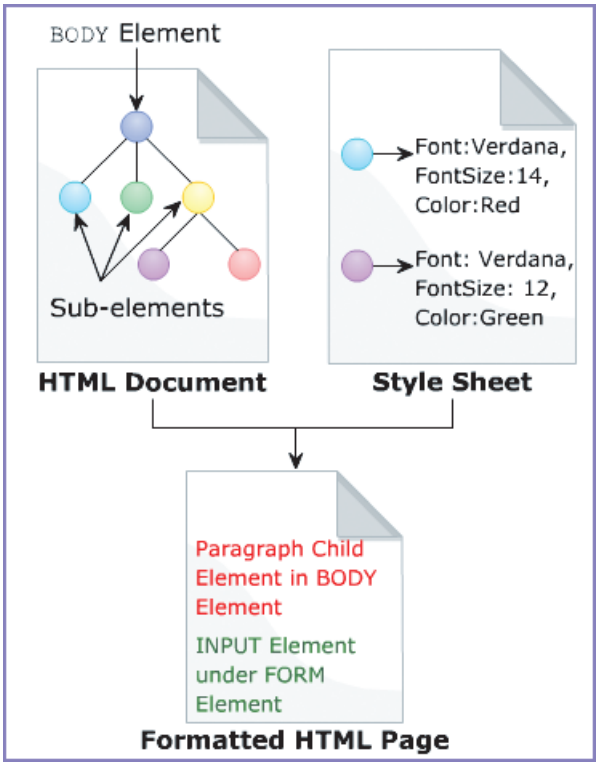



Figure 1.5: Need for Style Sheets

A CSS is a rule-based language, which specifies the formatting instructions for the content specified in an HTML page. Its purpose is to separate HTML content from its formatting so that Web page designers would not worry about the formatting and layout. This is because they can define the layout and formatting of the content in a separate file saved with an extension of .css. In the .css file, the formatting instructions for an element are referred to as a rule set. Each rule defines how the content specified within an element should be displayed in a Web browser.

While displaying the HTML page, the browser identifies the .css file for the page and applies the rules for the specified elements. You can merge the rules from different .css files or can edit them. This task of combining and matching rules from different files is referred to as cascading. Figure 1.6 shows an example of CSS.

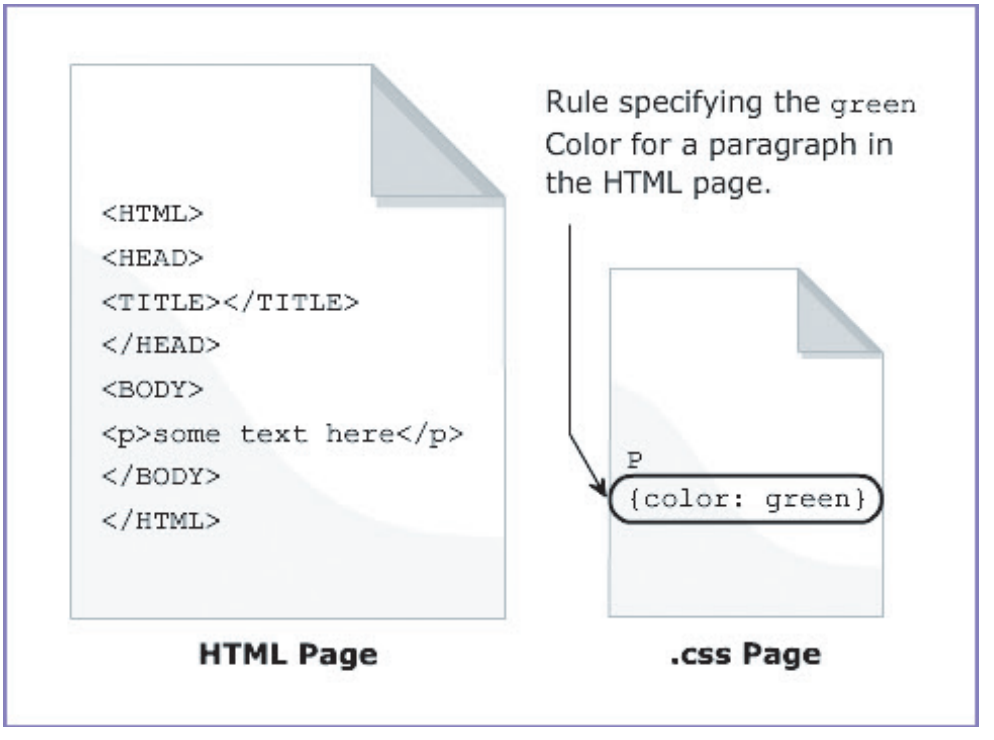



Figure 1.6: Example of CSS

## *1.7.1 Benefits of CSS*

Multiple HTML pages can use a CSS document. CSS provides some useful benefits that make it an ideal choice to specify the appearance of the content in an HTML page. These benefits are as follows:

- **Code Reusability**: CSS saves time by specifying the formatting options of an element only once and applying them to multiple HTML pages.

- **Less HTML Code**: CSS helps in reducing the file size of HTML documents by specifying the formatting instructions in another file.

- **Device Independence**: CSS is designed for different devices to provide the same look and feel of the HTML page across them.

## *1.7.2 Working of CSS*

You can embed the CSS code within the HTML code or link the HTML file externally to the CSS file. The browser will locate the style sheet irrespective of its location and will apply the style to the HTML page. There are certain steps involved in applying a style sheet to an HTML page. These steps are as follows:

1. The user requests for a Web page from the browser using the URL.
2. The server responds with the HTML file and related files such as image files, audio files, and external .css files, if any.
3. The browser executes the CSS code using the rendering engine and applies the styles to the HTML file.
4. The Web page is then displayed in the browser.

The rendering engine is a software that applies the formatting instructions to the Web page and displays the formatted content on the screen. Figure 1.7 shows the working of CSS.

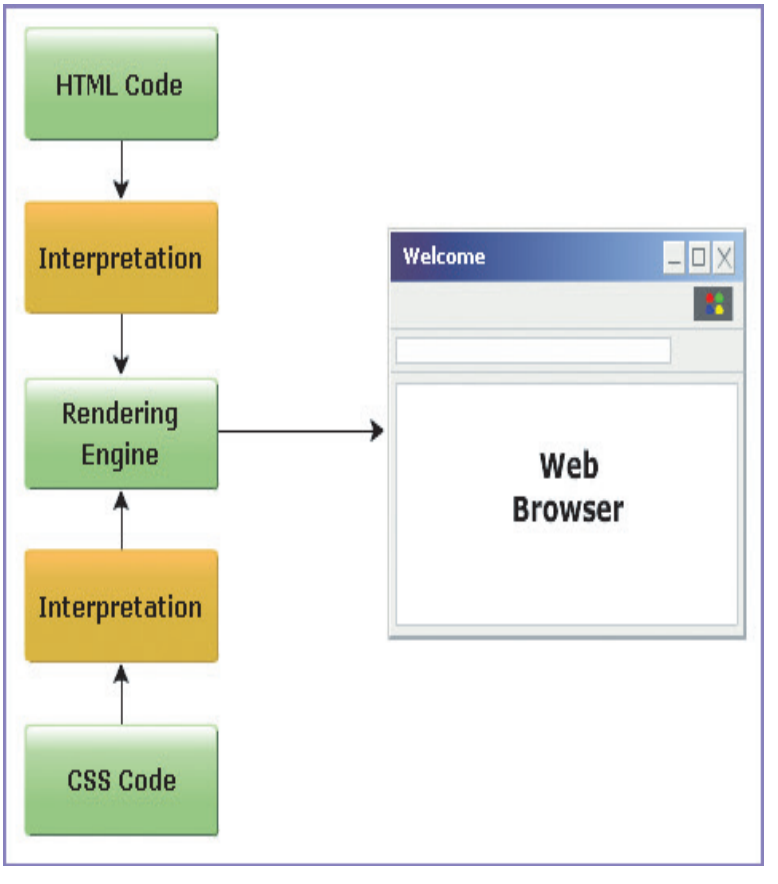



Figure 1.7: Working of CSS

## *1.8 JavaScript*

JavaScript helps to build dynamic Web pages by ensuring maximum user interactivity. JavaScript is a scripting language that supports object-oriented programming style. This means that it provides objects for specifying functionalities. An object has a unique identity, state, and behavior.

JavaScript being a light-weight programming language is embedded directly into HTML pages. JavaScript is also free for use by all. It is the most popular scripting language and is supported by the major browsers.

### *1.8.1 Functionality of JavaScript*

JavaScript helps to include additional expression and functionality to your Web pages. Some of the tasks that can be performed using JavaScript and HTML5 are as follows:

- With HTML5 and JavaScript, you can create a 2D drawable surface in your page without using any plug-ins.
- Use Web Workers to turbo charge the JavaScript code to perform advanced computation or make an application more responsive.
- Access any Web service and bring that data back to your application in real time.
- No need for special plug-ins to play video.
- Create your own video playback controls using HTML and JavaScript.
- There is no need to use browser cookies as the browser local storage can be used.
- Use JavaScript to perform full video processing in the browser. You can also create special effects and even directly manipulate video pixels.

Besides the points mentioned JavaScript can also perform the following functionalities:

- JavaScript helps Web designer to insert code snippets into the HTML pages without the need to have in-depth programming knowledge.
- JavaScript can be used to execute events on certain user actions such as on click of a HTML element, page load, and so on.
- HTML elements can be manipulated by using JavaScript.
- The browser information of a Web site visitor can be collected by using JavaScript.

## *1.9 jQuery*

jQuery is a JavaScript library which is supported on multiple browsers. It simplifies the designing of client-side scripting on HTML pages. The jQuery library is based on modular approach that allows the creation of powerful and dynamic Web applications. The use of jQuery on HTML pages enable developers to abstract the low-level interaction code with pre-defined library developed on top of the JavaScript. This also helps to keep the client-side script short and concise.

Some of the features of jQuery library are as follows:

1. Easier to understand syntax that helps to navigate the document
2. Event handling
3. Advanced effects and animation
4. Developing AJAX-based Web applications

**Note** - AJAX is a development technique that is used to create asynchronous Web applications.

jQuery is a preferred library used by developers, as it is easy to understand than JavaScript. Also, the features of jQuery enable the development of rich Web applications in a shorter period.

## *1.10 Browser Support*

Currently, no browsers have full HTML5 support as HTML5 has not yet been introduced as the official standard.

However, the major browsers, such as Chrome, Firefox, Opera, Safari, Internet Explorer, and so on, are trying to add new HTML5 features to the latest versions of the browsers.

## *1.11 Check Your Progress*

1. HTML is a _____ language.

| | | | |
|---|---|---|---|
| (A) | Markup | (C) | HyperMark |
| (B) | Markdown | (D) | MarkHyper |

2. Which of the following statements about evolution of Web and dynamic Web sites are true?

| | |
|---|---|
| (A) | A Web server is a computer that allows you to make Web pages available on the Internet. |
| (B) | MAN is a computer network that covers a small area such as office or school. |
| (C) | HTTP is a protocol used for requesting Web pages over the Internet. |
| (D) | A Web browser is a software that processes user requests and responds with the requested page to the Web server. |
| (E) | A dynamic Web page is a Web page that can be created using CSS and JavaScript. |

3. Which of the following is a new tag in HTML5?

| | | | |
|---|---|---|---|
| (A) | Head | (C) | Body |
| (B) | Header | (D) | Title |

4. Which of the following is a rule-based language?

| | | | |
|---|---|---|---|
| (A) | CSS | (C) | JavaScript |
| (B) | jQuery | (D) | Geolocation |

5. jQuery is a JavaScript library which is supported on _____ browsers.

| | | | |
|---|---|---|---|
| (A) | Single | (C) | Dual |
| (B) | Multiple | (D) | Non-individual |

## *1.11.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | A, C, E |
| 3. | B |
| 4. | A |
| 5. | B |

# Summary

- HTML5 is cooperative project between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).
- The new features of HTML5 would include tags such as <canvas>, <article>, <nav>, <header>, <footer>, <section>, <audio>, <video>, and so on.
- Some of the technologies used for creating dynamic Web sites are JavaScript, CSS, XHTML, and DHTML.
- A Cascading Style Sheet (CSS) is a rule-based language, which specifies the formatting instructions for the content specified in an HTML page.
- JavaScript is a scripting language that allows you to build dynamic Web pages by ensuring maximum user interactivity.
- jQuery is a JavaScript library which is supported on multiple browsers. It simplifies the designing of client-side scripting on HTML pages.
- The major browsers, such as Chrome, Firefox, Opera, Safari, Internet Explorer, and so on, are trying to add the new HTML5 features to the latest versions of the browsers.

# *Try it Yourself*

1. View the Web site of Aptech Global Leaning Solutions (www.aptech-worldwide.com).
2. Identify the header and footer on the Web site.
3. Create a layout of the basic structure of the body of the Web site.

# Session - 2

# Introduction to HTML5

Welcome to the Session, **Introduction to HTML5**.

This session explains the basic tags that are present in HTML5. This session also lists the different data types, attributes, and entities of HTML5. Finally, the session explains the advantages of using HTML5 in mobile application development.

In this Session, you will learn to:

- Explain the elements constituting an HTML tag
- Describe DOCTYPE declarations
- Explain the basic tags in HTML
- List the different data types, attributes, and entities of HTML5
- Describe container and standalone tags
- Explain the role of HTML5 in mobile devices

## 2.1 Introduction

All elements in HTML5 are organized using tags. The basic tags in HTML5 include `html`, `head`, `title`, `meta`, `link`, `script`, and `body`. The `DOCTYPE` must be provided before inserting the basic tags in HTML5. There are different data types, attributes, and entities that can be applied to the tags present in HTML5. All the tags are either classified into container tags or standalone tags. This classification is based on the use of the end tag for a certain HTML element. HTML5 is also the preferred language for mobile application development because of its various benefits.

This session explains the basic tags that are present in HTML5. This session also lists the different data types, attributes, and entities of HTML5. Finally, the session explains the advantages of using HTML5 in mobile application development.

## 2.2 Elements

An element organizes the content in a Web page hierarchically, which forms the basic HTML structure. It consists of tags, attributes, and content. Tags denote the start and end of an HTML element.

A start tag includes an opening angular bracket (<) followed by the element name, zero or more space separated attributes, and a closing angular bracket (>). Attributes are name/value pairs that describe the element and content format. An end tag is written exactly as the start tag, but the forward slash (/) precedes the element name. Figure 2.1 shows an element in HTML tag.

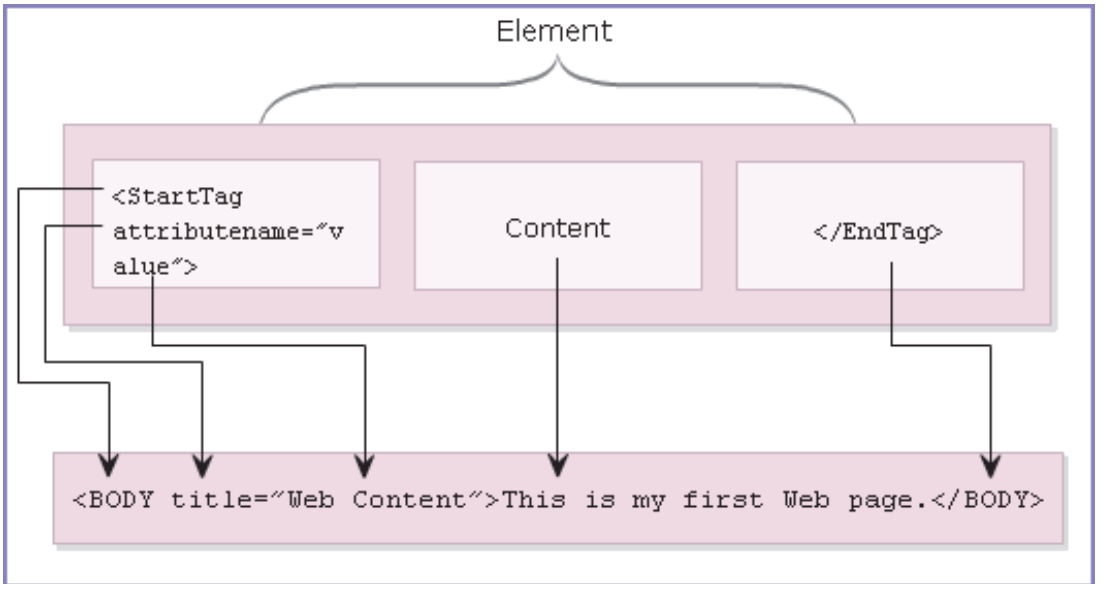



**Figure 2.1: Element**

**Note** - Tags are referred to as the markup in an HTML document.

## 2.3 Doctype

The `DOCTYPE` element informs the browser the HTML version number of your document. It is the first declaration in the HTML5 document before any other HTML code is written. By using a `DOCTYPE`, the browser is able to be more precise in the way it interprets and renders your pages. It is highly recommended to use a `DOCTYPE` at the beginning of all HTML documents.

The new HTML5 DOCTYPE declaration is as follows:

```
<!DOCTYPE html>
```

Not only is this syntax valid for the `DOCTYPE` for HTML5, but it is also the DOCTYPE for all future versions of HTML. This DOCTYPE is compatible even with the older browsers.

The following example shows the use of the new document type specification.

**Example**:

```
<!DOCTYPE html>
```

## 2.4 Basic Tags

An HTML document is made up of different elements, tags, and attributes, which specify the content and its format. Therefore, HTML is both a structural and presentational markup language. Structural markup specifies the structure of the content, while the presentational markup specifies the format.

An HTML page is saved with the `.html` extension. The basic structure of an HTML document mainly consists of seven basic elements. These are as follows:

➔ **HTML**

The `HTML` element is the root element that marks the beginning of an HTML document. It contains the start and end tag in the form of `<HTML>` and `</HTML>` respectively. It is the largest container element as it contains various other elements.

➔ **HEAD**

The `HEAD` element provides information about the Web page such as keywords and language used, which is not displayed on the Web page. Keywords are important terms existing in a Web page used by the search engines to identify the Web page with respect to the search criterion.

### ➔ TITLE

The TITLE element allows you to specify the title of the Web page under the `<TITLE>` and `</TITLE>` tags. The title is displayed on the Title bar of the Web browser. The `TITLE` element is included within the `HEAD` element.

### ➔ META

The meta tag is used for displaying information about the data. In HTML5, the content meta tag which was used for specifying the charset or character encoding has been simplified. The new `<meta>` tag is as follows:

```
<meta charset="utf-8" />
```

UTF-8 is the most commonly used character coding that supports many alphabets. UTF-8 is also being promoted as the new standard.

There are several other attributes associated with the meta tag that can be used to declare general information about the page. This information is not displayed in the browser. Meta tags provide search engines, browsers, and Web services the information that is required to preview or acquire a summary of the relevant data of your document.

In HTML5, it is not very important to self-close tags with a slash at the end. Though self-enclosing is recommended for compatibility reasons.

### ➔ LINK

The `<link>` tag is used to define the association between a document and an external resource. It is used to link stylesheets. Its type attribute is used to specify the type of link such as '`text/css`' which points out to a stylesheet.

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

The `type` attribute is not included in HTML5. The reason is that CSS has been declared as the default and standard style for HTML5. So, the new link is as follows:

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

### ➔ SCRIPT

With HTML5, JavaScript is now the standard and default scripting language. Hence, you can remove the type attribute from the script tags too. The new script tag is as follows:

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

The following example shows the use of the script tag.

**Example**:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  <title>HTML Webinar</title>
    <link rel="stylesheet" href="first.css">
    <script src="first.js"></script>
  </head>
</html>
```

➔ **BODY**

The BODY element enables you to add content on the Web page specified under the `<BODY>` and `</BODY>` tags. Content can include text, hyperlinks, and images. You can display the content using various formatting options such as alignment, color, and background. Figure 2.2 shows the basic HTML elements.

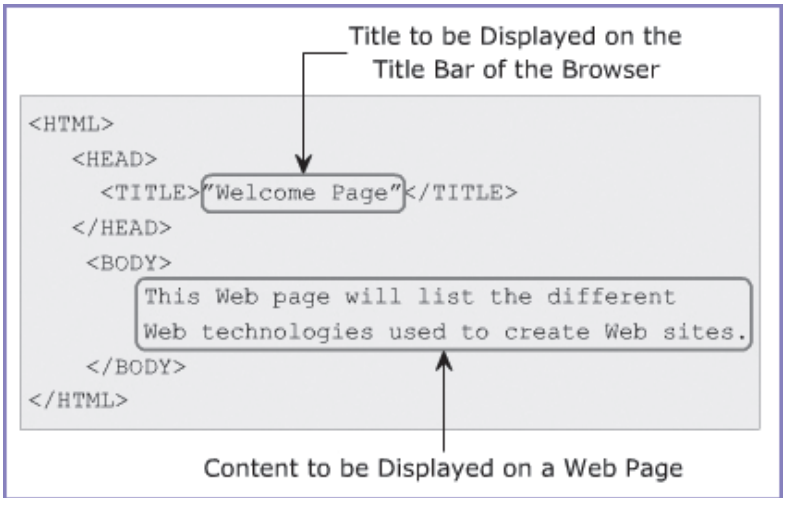



**Figure 2.2: Basic HTML Elements**

## *2.5 Data Types*

A data type specifies the type of value assigned to the attributes and the type of content that is to be displayed on the Web page. The different types of content include text, images, hyperlinks, video, and audio. Data types help in identifying the type of formatting such as color and length of data.

The important basic HTML data types are as follows:

➔ **Text Strings**

Specifies textual content, which is readable by the user.

➔ **Uniform Resource Identifiers (URIs)**

Specifies the location of Web pages or network files.

➔ **Colors**

Specifies the color to be applied to the content on the Web page.

➔ **Lengths**

Specifies the spacing among HTML elements. Length values can be in `Pixels`, `Length`, or `MultiLength`. `Pixels` refer to the smallest dot on the screen. `Length` is specified as a percentage value of `Pixels` or available space on the screen. `MultiLength` can be specified as `Pixels` or percentage.

➔ **Content Types**

Specifies the type of content to be displayed on a Web page. Examples of content types include "`text/html`" for displaying text using HTML format, "`image/gif`" for displaying image of a .gif format, and "`video/mpg`" for displaying a video file of .mpg format. Figure 2.3 shows the different data types.

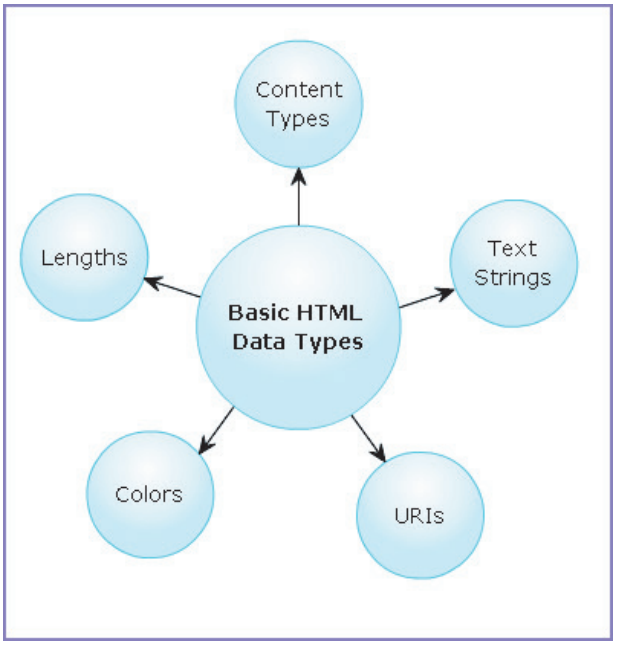



Figure 2.3: Data Types

## 2.6 Attributes

HTML attributes helps to provide some meaning and context to the elements. Some of the global attributes used in HTML5 elements are as follows:

- **class** – Specifies classnames for an element.
- **contextmenu** – Specifies the context menu for an element.
- **dir** – Specifies the direction of the text present for the content.
- **draggable** – Specifies the draggable function of an element.
- **dropzone** – Specifies whether the data when dragged is copied, moved, or linked, when dropped.
- **style** – Specifies the inline CSS style for an element.
- **title** – Specifies additional information about the element.

## 2.7 HTML Entities

Entities are special characters that are reserved in HTML. These entities can be displayed on a HTML5 Web site using the following syntax:

**Syntax**:

```
&entity_name;
```

OR

```
&#entity_number;
```

Table 2.1 shows some of the commonly used HTML entities.

| Output | Description | Entity Name | Entity Number |
|---|---|---|---|
| | non-breaking space |   |   |
| < | less than | < | < |
| > | greater than | > | > |
| & | ampersand | & | & |
| € | euro | € | € |
| © | copyright | © | © |

**Table 2.1: HTML Entities**

## 2.8 Container and Standalone Tags

There are two types of HTML elements namely, container and standalone elements. A container element includes the start tag, contents, sub-elements, and end tag. All the basic HTML elements are container elements. A standalone element consists of the start tag and attributes followed by the end tag as /> without any content.

## 2.9 HTML5 and Mobile Devices

HTML5 has helped to create better and richer mobile applications. For this, APIs are used in HTML5. These APIs support advanced Web application features for mobile browsers.

HTML5 is not supported by older mobile devices. New age smartphones with Apple iOS and Google Android as operating systems support HTML5 compliant browsers. Even Microsoft Windows 7 for Mobile will have a newly developed browser to support HTML5 developed Web sites and applications.

Due to the various mobile platforms available on mobile devices, development of mobile applications is difficult. HTML5 has tried to integrate all the features to deploy mobile applications that would be compatible in all the platforms. HTML5 provides features such as drag-and-drop functionality, video embedding in an application, and even offline capabilities.

As HTML5 is compatible with most mobile operating systems, upto 30% of the cost for development for different operating systems is saved. Also, there is a reduced dependency in third-party components, thus reducing the licensing costs. All the required components will be readily available through the browser in HTML5.

### 2.9.1 Benefits of HTML5 for Mobile Development

The benefits of HTML5 for mobile developments are as follows:

- HMTL5 has included APIs, hence additional plug-ins are not required for mobile browsers.
- Mobile development is easier as knowledge of only HTML5, CSS, and JavaScript is majorly required. These are easier as compared to the other languages used for Mobile development. The development is also faster in HTML5.
- There is a rising growth for mobile applications and due to its enhanced compatibility, HTML5 forms the foundation for developing these mobile applications.

- HTML5 is compatible with most operating system platforms. The mobile applications developed on HTML5 can run on browsers of Android, iOS, Blackberry, Windows Phone, and other mobile operating systems.

- The development cost for creating applications in HTML5 is low.

- Applications based on location and maps will have greater support in HTML5. The plan is to support such applications on browsers, hence making them platform independent.

- Third-party programs are not required in HTML5. Hence, media functions such as audio and video have better functionality and improved support in HTML5.

## *2.10 Check Your Progress*

1. A _____ element includes the start tag, contents, sub-elements, and end tag.

| | | | |
|---|---|---|---|
| (A) | Container | (C) | Emphasis |
| (B) | Standalone | (D) | Formatted |

2. Which of the following DOCTYPE is used in HTML5?

| | | | |
|---|---|---|---|
| (A) | HTML | (C) | Emphasis |
| (B) | Standalone | (D) | Formatted |

3. What does URI stand for?

| | | | |
|---|---|---|---|
| (A) | Uniform Resource Informer | (C) | Uniform Reply Identifier |
| (B) | Uniform Resource Identifier | (D) | Uniform Recourse Information |

4. Which of the following attribute in HTML5 specifies the text direction?

| | | | |
|---|---|---|---|
| (A) | class | (C) | dir |
| (B) | contextmenu | (D) | style |

5. Which of the following is the mobile-based operating system developed by Apple?

| | | | |
|---|---|---|---|
| (A) | Symbian | (C) | Windows 7 |
| (B) | iOS | (D) | Android |

## *2.10.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | A |
| 3. | B |
| 4. | C |
| 5. | B |

# Summary

- An element organizes the content in a Web page hierarchically, which forms the basic HTML structure.
- The DOCTYPE element tells the browser the type of your document.
- A data type specifies the type of value assigned to the attributes and the type of content that is to be displayed on the Web page.
- Entities are special characters that are reserved in HTML.
- A container element includes the start tag, contents, sub-elements, and the end tag.
- A standalone element consists of the start tag and attributes followed by the end tag as /> without any content.
- HTML5 provides features such as drag-and-drop functionality, video embedding in an application, and even offline capabilities for mobile devices.

# *Try it Yourself*

1. Samantha wants to create a Web site for her new Orchestra Group named **We Five**. She wants to design the Web page that will allow the search engine to get the required information. In order for the search engine to get the required information, the page should have the following details:

   a. Doctype – State the type of Doctype in the tag.

   b. HTML

      i. Head

      ii. Title – Provide the name of the orchestra group in the tag.

      iii. Meta – Provide keywords that would be applicable to the orchestra group Web site.

   c. Body – Provide a small introduction about the orchestra group in the body section.

   Add any other additional content and other information as required.

Onlinevarsity

# JOIN GROUPS FOR COLLABORATIVE LEARNING

# Session - 3

# Formatting Text Using Tags

Welcome to the Session, **Formatting Text Using Tags**.

This session explains the different tags related to formatting of text. This session also explains the different types of lists. Finally, the session explains the procedure to change the background of a Web page.

In this Session, you will learn to:

- Explain the Heading tag
- Explain the different tags related to formatting
- Explain monospaced font, preformatted text, and block quotation
- Describe the different types of lists
- Explain the procedure to change the background color and image

## *3.1 Introduction*

The text content of the Web page, form an important part of a Web site. This text content must not only be informative but also attractive. It must be easy to read and must have short and crisp sentences for easy understanding of the Web user. To attract the attention of the user, headings must be appropriately provided. Also, text formatting options such as bold, italics, subscript, superscript, and so on must be applied on the text. Bullets can be also used to list the text in a systematic manner. The background color and background image of a Web page can be specified using HTML.

## *3.2 Headings*

The heading elements define headings for contents such as text and images. They specify a hierarchical structure of a Web page by grouping the contents into different headings.

HTML defines six levels of heading that ranges from H1 to H6. The H1 element specifies the top-level heading, which is displayed with the largest size. The H6 element specifies the lowest-level heading, which is displayed with the smallest size.

Code Snippet 1 demonstrates how to specify the six levels of heading in an HTML page.

**Code Snippet 1**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Headings</title>
    </head>
    <body>
        <h1>H1 Heading</h1>
        <h2>H2 Heading</h2>
        <h3>H3 Heading</h3>
        <h4>H4 Heading</h4>
        <h5>H5 Heading</h5>
        <h6>H6 Heading</h6>
    </body>
</html>
```

The heading under the H1 tags will be displayed with the largest size. Each subsequent heading will be displayed in a size lower than its previous heading.

The heading under the H6 tags will be displayed with the lowest size. Figure 3.1 shows the different types of headings.

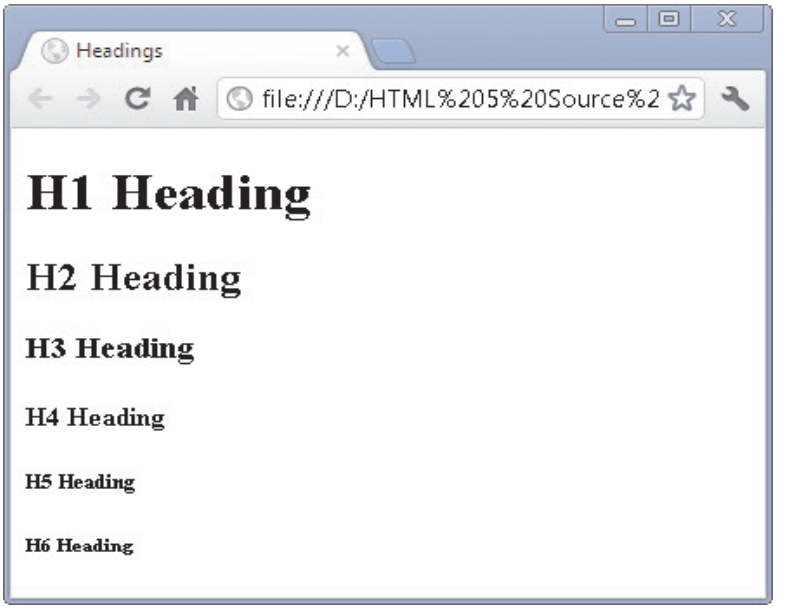



Figure 3.1: Different Types of Headings

## 3.2.1 HGROUP

The `<hgroup>` element is a new element defined in HTML5. It is used to group titles and their subtitles. The element is used to group a set of h1–h6 elements. These are used for headings that have multiple levels that can include subheadings, alternative titles, taglines, and so on. The main advantage of using the `<hgroup>` tag is to create a document outline. Code Snippet 2 shows the use of the `<hgroup>` tag.

**Code Snippet 2**:

```
<hgroup>
  <h1>Title of Post One </h1>
  <h2>Subtitle of Post One </h2>
</hgroup>
```

## 3.3 Formatting

The content format determines how the content will appear in the browser. For example, when you visit a Web site, some text appears in a specific format such as bold or underlined. This means that the formatted content makes an HTML page look readable and presentable. In HTML, formatting is applied to the text by using formatting elements, which are container elements.

The commonly used HTML formatting elements are as follows:

- B

  The B element displays the text in bold. The text that needs to be displayed in bold is enclosed between  and  tags.

- I

  The I element displays the text in italic. The text that needs to be displayed in italic is enclosed between  and  tags.

- SMALL

  The SMALL element makes the text appear smaller in a browser. The text that needs to be displayed smaller is enclosed between <small> and </small> tags.

- U

  The U element applies an underline to the text. The text that needs to be underlined is enclosed between <u> and </u> tags.

Code Snippet 3 demonstrates the use of the basic formatting elements.

**Code Snippet 3**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Formats</title>
    </head>
    <body>
        <h2>Using HTML Formatting Elements</h2><br>
        <b>This text is displayed in bold.</b><br>
        <i>This text is displayed in italic.</i><br>
        <u>This text is underlined.</u><br>
        <small>This text is displayed smaller.</small>
    </body>
</html>
```

Figure 3.2 shows a Web site displaying basic text formats.

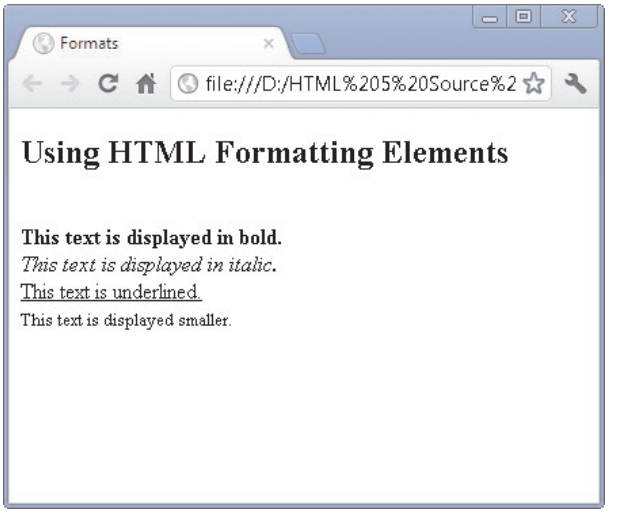



**Figure 3.2: Basic Text Formats**

You might want to display some text or characters above or below the baseline of its adjacent text. Further, you might want to strike through some text to indicate that it is deleted. Therefore, HTML provides you with some more formatting elements. These formatting elements are as follows:

➔ `DEL`

The `DEL` element encloses text, which has been deleted from the document. The text to be deleted is placed in the `<del>` and `</del>` tags.

➔ `INS`

The `INS` element encloses text, which has been inserted in the document. The text to be inserted is placed in the `<ins>` and `</ins>` tags. The INS element can be used with DEL element to inform the user about the deleted text, which is replaced by the new text.

➔ `STRONG`

The `STRONG` element emphasizes the text as compared to its surrounding text. The text to be emphasized is placed in the `<strong>` and `</strong>` tags.

➔ `SUB`

The `SUB` element displays the text as subscript. The text to be displayed as subscript is enclosed in `<sub>` and `</sub>` tags.

➔ `SUP`

The `SUP` element displays the text as superscript. The text to be displayed as superscript is enclosed in `<sup>` and `</sup>` tags.

Code Snippet 4 demonstrates the use of `DEL`, `INS`, `STRONG`, `SUB`, and `SUP` elements.

**Code Snippet 4**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Updating and Shifting Text</title>
    </head>
    <body>
        <h3>Updating, Emphasizing, and Shifting Text</h3>
        This is an example of <del>deleted</del> <ins>inserted
        </ins> text.<br/>
        This is an example of <strong>Strong</strong> text.<br/>
        This is an example of <sub>subscript</sub>text.<br/>
        This is an example of <sup>superscript</sup> text.<br/>
    </body>
</html>
```

Figure 3.3 shows the other text formatting elements.

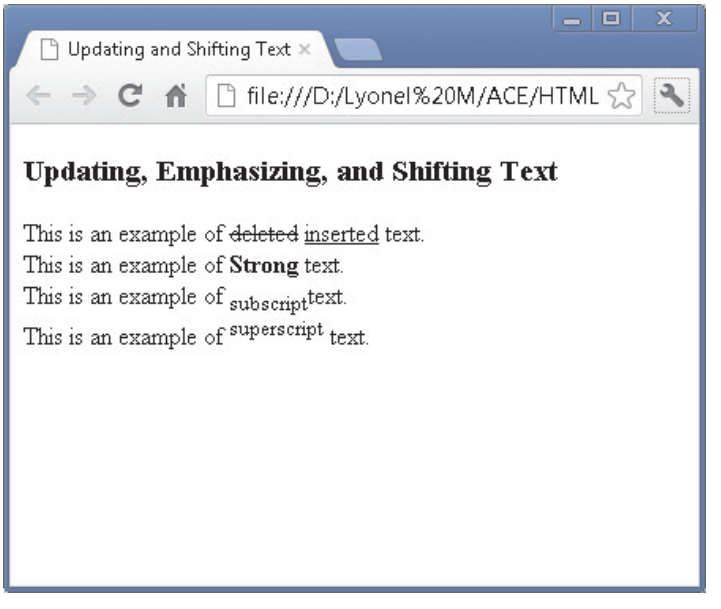



**Figure 3.3: Other Text Formatting Elements**

**Note** - The <font> tag present in HTML 4 has been removed in HTML5. To change the font size, style, color, and so on, the style attribute must be used in CSS3. This will be covered in later sessions.

## *3.4 Monospaced and Preformatted Text*

By using monospaced font in HTML5 a Web developer can provide the same amount of horizontal space between the fonts, even if the font size, shape, and type are not the same. Monospaced fonts are used for programming code scripts, instruction texts, and ASCII characters.

The `<pre>` tag is used to apply preformatted text content to a Web page. The `<pre>` tag applies a fixed-font width to the text content. It also maintains a standard formatting for spaces and line breaks. The `<pre>` tag is usually used to when you want to copy paste content from a source but do not want to change the formatting of the same. The content would be copied while maintaining all the line breaks and spaces.

Table 3.1 lists some of the other predefined tags available for formatting of text in HTML.

| Tag | Description |
|---|---|
|  | Used for emphasized text |
| <dfn> | Used for a definition term |
| <code> | Used for a computer code |
| <samp> | Used for sample output from a computer program |
| <cite> | Used for a citation |

**Table 3.1: Predefined Tags**

### *3.4.1 Format a Block Quotation*

To define a long quotation or block quotation, the `<blockquote>` tag can be used. When the `<blockquote>` tag is used, the quotation is indented in browsers.

Code Snippet 5 shows the use of the `<blockquote>` tag.

**Code Snippet 5**:

```
<blockquote>

    "When one door closes, another opens; but we often look so long
    and so regretfully upon the closed door that we do not see the
    one which has opened for us." -Alexander Graham Bell

</blockquote>
```

## *3.5 Lists*

A list is a collection of items, which might be organized in a sequential or unsequential manner. You can use a list to display related items that belong to a particular category. For example, to display the types of computers, such as mainframe, microcomputer, and laptops, you would organize these items one below the other under the Types of Computers category. Similarly, HTML allows you to display related items in a list on a Web page.

A list in HTML can contain paragraphs, line breaks, images, links, and other lists. The items within a list are displayed on a Web page one below the other using bullets. HTML supports three types of lists. These are as follows:

- Ordered
- Unordered
- Definition

Figure 3.4 shows the different types of lists.

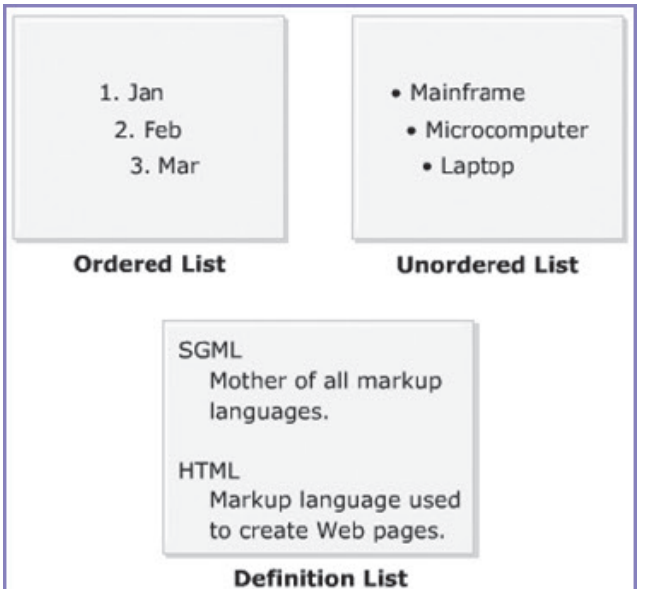



**Figure 3.4: Different Types of Lists**

## *3.5.1 Ordered List*

An ordered list is a list of items arranged in a particular order. Here, the order of the items is important as it indicates a sequential flow. For example, to display the days in a week or months in a year, you would use numbered bullets. Similarly, HTML allows you to implement ordered lists where each item in the list is displayed using a numbered or alphabetic bullet.

HTML provides two elements for creating an ordered list. These are as follows:

- OL: Creates an ordered list.
- LI: Specifies an item and it is a sub-element of the OL element.

Code Snippet 6 demonstrates the use of `OL` and `LI` tags to display weekdays as an ordered list.

**Code Snippet 6**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Days in a Week</title>
    </head>
    <body>
        <h2>Days in a Week:</h2>
        <ol>
            <li>Sunday</li>
            <li>Monday</li>
            <li>Tuesday</li>
            <li>Wednesday</li>
            <li>Thursday</li>
            <li>Friday</li>
            <li>Saturday</li>
        </ol>
    </body>
</html>
```

Figure 3.5 shows an example of an ordered list.

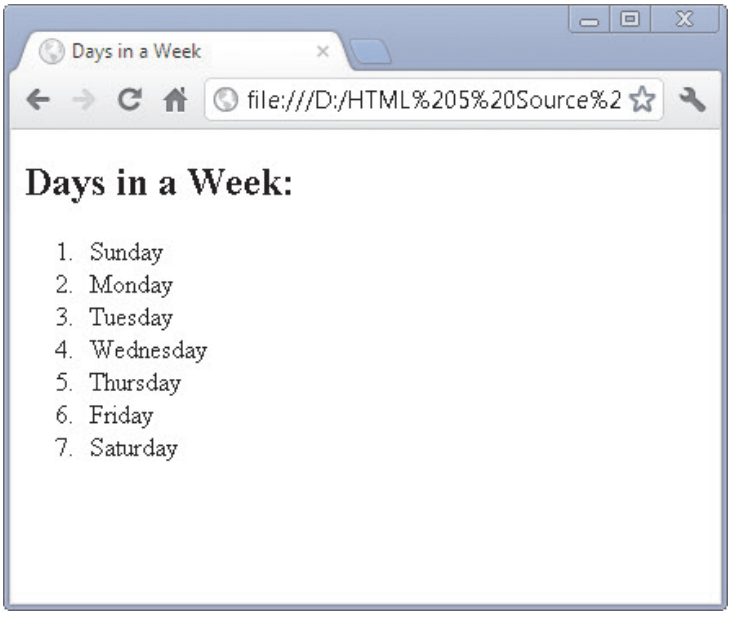



**Figure 3.5: Ordered List**

Different numbering styles, such as roman numerals or alphabetic bullets, can be applied to an ordered list. Table 3.2 shows the different numbering styles that can be specified in an ordered list.

| Property's Value | Example |
|---|---|
| `decimal` | 1, 2, 3, ... |
| `lower-alpha` | a, b, c, ... |
| `upper-alpha` | A, B, C, ... |
| `lower-roman` | i, ii, iii, ... |
| `upper-roman` | I, II, III, ... |

**Table 3.2: Different Numbering Styles**

The `list-style-type` property is used to specify a numbering style for the ordered list. It is the property of the `style` attribute, which is specified within the `<ol>` tag.

Code Snippet 7 demonstrates the use of `list-style-type` property for adding a numbering style to a numbered list.

**Code Snippet 7**:

```
<ol style=list-style-type:lower-roman>
    <li>Sunday</li>
    <li>Monday</li>
    <li>Tuesday</li>
    <li>Wednesday</li>
    <li>Thursday</li>
    <li>Friday</li>
    <li>Saturday</li>
</ol>
```

The `list-style-type` property of the `style` attribute in the code is set to `lower-roman`. The property and its value are separated by a colon. This means that the days of the week will be displayed sequentially by applying the lower-cased Roman numbers as bullets. Figure 3.6 shows the `lower-roman` style.

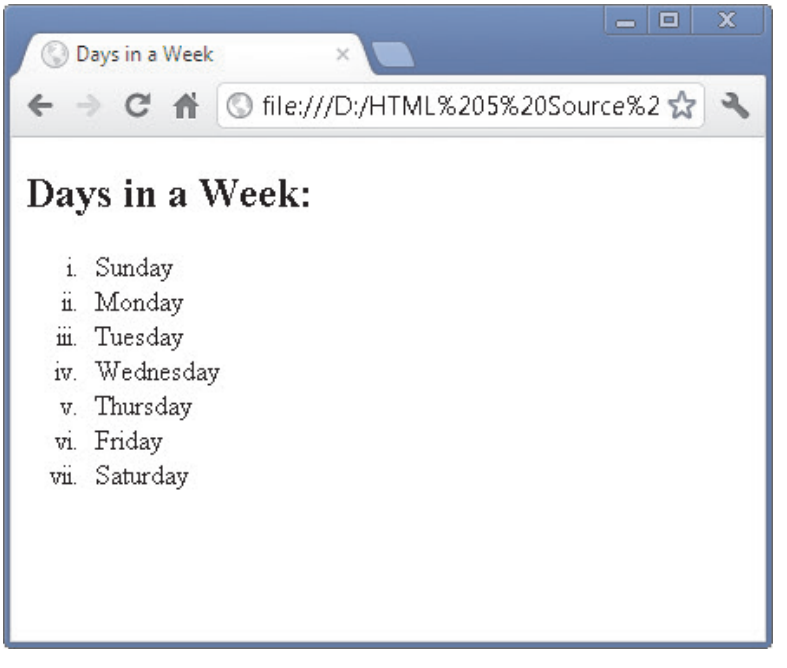



Figure 3.6: lower-roman Style

## *3.5.2 Unordered List*

An unordered list is a list where the items are arranged in a random order. This means that you will create an unordered list when the order of related items is not important.

For example, to list the features of a product, you would create an unordered list. Each item in an unordered list is displayed using symbolic bullets such as circles and squares. These bullets are similar to the bullets provided by Microsoft Office Word.

HTML provides the UL element to create an unordered list.

Code Snippet 8 demonstrates how to display the features of a product as an unordered list.

**Code Snippet 8**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Features of EasyPad</title>
    </head>
    <body>
        <h2>Features of EasyPad</h2>
        <ul>
            <li>Opens many files at a time</li>
            <li>Unlimited undo and redo</li>
            <li>Reads and writes both Windows and Unix files</li>
        </ul>
    </body>
</html>
```

Figure 3.7 shows an example of an unordered list.

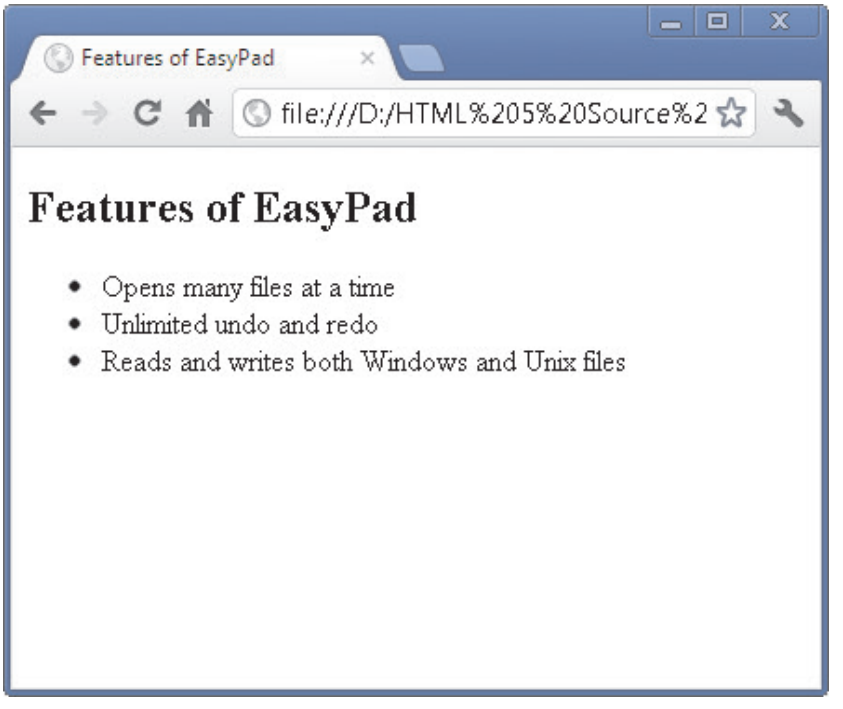



**Figure 3.7: Unordered List**

The `UL` element contains the `<ul>` tag and multiple `li` sub-elements. The `<ul>` tag marks the beginning of the unordered list. Each of the `li` sub-elements starts with the `<li>` tag followed by a feature of the `EasyPad` product. Each feature will be displayed with the default symbolic bullet, which is a small black disc.

The `list-style-type` property specifies the type of bullet to be applied to an unordered list. There are three types of bullets defined for the unordered lists in HTML. These bullet types are namely, `disc`, `square`, and `circle`. The default value is `disc`, which is applied to the unordered list, even if the `list-style-type` property is not specified.

Code Snippet 9 demonstrates how to apply the square bullet to an unordered list.

**Code Snippet 9**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Wild Animals</title>
    </head>
    <body>
        <h2>Wild Animals</h2>
        <ul style="list-style-type:square">
            <li>Lion</li>
            <li>Tiger</li>
            <li>Leopard</li>
            <li>Wolf</li>
        </ul>
    </body>
</html>
```

The `list-style-type` property of the style attribute is set to square. This means that the unordered list of wild animals will be displayed using the square bullet as shown in figure 3.8.

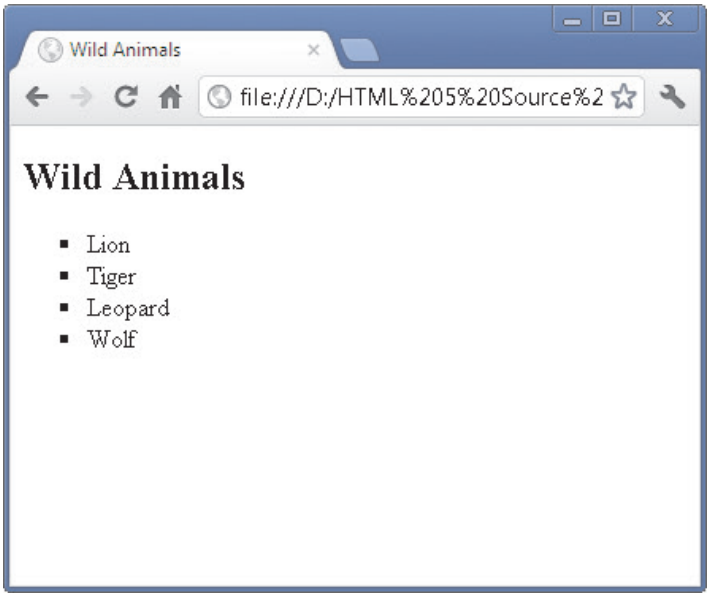



**Figure 3.8: Square Bulleted Unordered List**

## *3.5.3 Definition List*

A definition list refers to a collection of terms with their corresponding descriptions. For example, you can display a glossary on a Web page by creating a definition list, which will contain the terms and their descriptions. A definition list appears with the term indented on the left followed by the description on the right or on the next line. By default, the description text appears on the next line and is aligned with respect to the term.

You can specify a single line or multiple lines of description for each term. HTML provides three elements to create a definition list. These elements are as follows:

- `DL`

  Is a container element that consists of the `DT` and `DD` sub-elements. It specifies that a definition list will be created using these elements.

- `DT`

  Specifies the term to be defined or described.

- `DD`

  Specifies the definition or description of the term.

Consider a scenario, where you want to create a Web page that will display the types of nouns with their descriptions. To do this, you must create a definition list. The steps to create a definition list are as follows:

1. Specify the DL element to indicate that you want to create a definition list.
2. Use the DT element to specify the term such as Common Noun.
3. Use the DD element to specify the description of the term.

Code Snippet 10 demonstrates the way to create a definition list.

**Code Snippet 10**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Types of Nouns</title>
    </head>
    <body>
        <h2>Types of Nouns</h2>
        <dl>
            <dt><b>Common Noun:</b></dt>
            <dd>It is a name of an object in general, such as
            pencil, pen, paper, and so on.</dd>
            <dt><b>Proper Noun:</b></dt>
            <dd>It is the unique name of a person or a place.
            </dd>
        </dl>
    </body>
</html>
```

Figure 3.9 shows an example of a definition list.

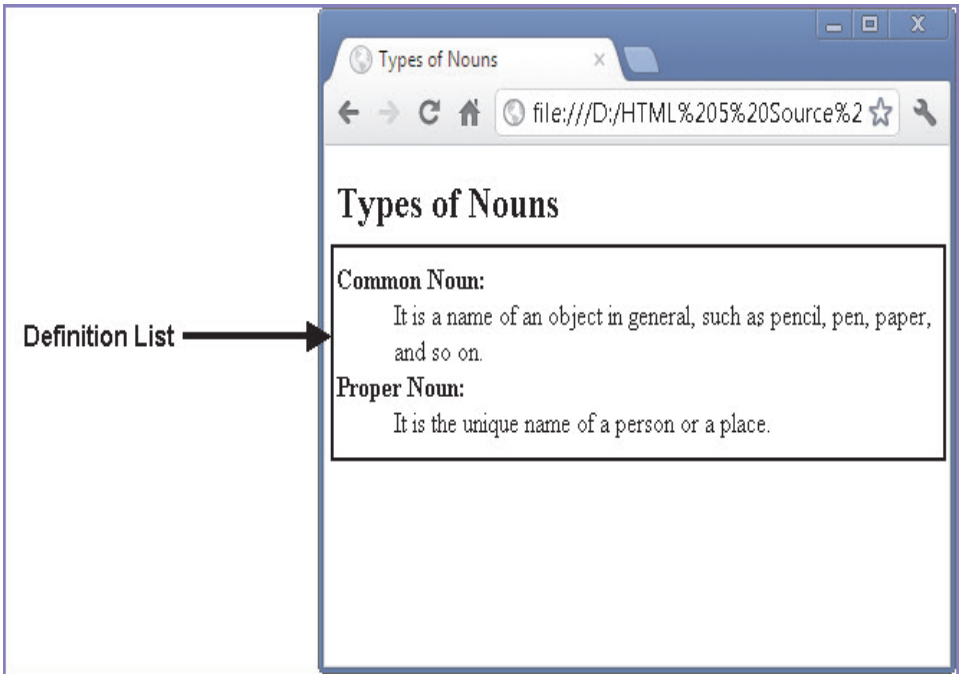



Figure 3.9: Definition List

## *3.6 Background and Foreground Colors*

HTML provides background properties that specify the background color and image for the Web pages. To specify a background for a Web page, use the `background` property. The `background` property is a shorthand property that specifies all the background properties in just one declaration.

The `bgcolor` attribute specifies the background color of a document. The syntax for `bgcolor` is shown.

**Syntax:**

```
<body bgcolor="color_name|hex_number|rgb_number">
```

where,

color_name - Specifies the background color with a color name (such as "red")

hex_number - Specifies the background color with a hex code (such as "#ff0000")

rgb_number - Specifies the background color with an rgb code (such as "rgb(255,0,0)")

Code Snippet 11 demonstrates how to specify a background color for an HTML document.

**Code Snippet 11**:

```
<html>
    <body bgcolor="#E6E6FA">
        <h1>Hello world!</h1>
        <p><a href="http://www.w3schools.com">
        Visit W3Schools.com!</a></p>
    </body>
</html>
```

Another way to specify a background color for a Web page is by using the `style="background-color: color"` attribute. This attribute must be added to the `<body>` tag. An example for applying a background color using the `style` attribute is as follows:

**Example**:

```
<body style="background-color: yellow">
```

The color name 'yellow' can also be replaced by the hex code or the rgb code.

The default text color of the page is specified as the foreground color. The foreground color can be specified by using the `style="color: color"` attribute. An example for applying a background and foreground color using the style attribute is as follows:

**Example**:

```
<body style="background-color: navy; color: yellow">
```

## *3.7 Background Image File*

A Web site developer can also insert an image as the background on a Web page. These background images are not recommended as sometimes the colors in the image may hide the text content. Hence, it is best to choose images with lighter shades. Also, as the image is tiled, it is best to choose an image that blends well and looks like a single image even after it is tiled.

Code Snippet 12 demonstrates the way to specify a background image in an HTML Web page.

**Code Snippet 12**:

```
<html>
    <body background="bgimage.jpg">
    </body>
</html>
```

## *3.8 Check Your Progress*

1. Which of the following formatting element emphasizes the text as compared to its surrounding text?

| | | | |
|---|---|---|---|
| (A) | B | (C) | I |
| (B) | Strong | (D) | U |

2. Which of the following tag is used to specify a definition term?

| | | | |
|---|---|---|---|
| (A) | `<em>` | (C) | `<dfn>` |
| (B) | `<code>` | (D) | `<blockquote>` |

3. Which of the following are the two elements used for creating an ordered list?

| | | | |
|---|---|---|---|
| (A) | OL | (C) | LI |
| (B) | UL | (D) | DF |

4. The _____ tag is used to apply preformatted text content to a Web page.

| | | | |
|---|---|---|---|
| (A) | `<pre>` | (C) | `<dfn>` |
| (B) | `<em>` | (D) | `<a>` |

5. _____ fonts are used for programming code scripts, instruction texts, and ASCII characters.

| | | | |
|---|---|---|---|
| (A) | Spaced | (C) | Hyperspaced |
| (B) | Monospaced | (D) | Preformatted |

## *3.8.1 Answers*

| | |
|---|---|
| 1. | B |
| 2. | B |
| 3. | A, C |
| 4. | A |
| 5. | B |

# Summary

- The heading elements define headings for contents such as text and images.
- The <hgroup> element is used to group titles and their subtitles.
- Monospaced fonts are used for programming code scripts, instruction texts, and ASCII characters.
- The <pre> tag is used to apply preformatted text content to a Web page.
- To define a long quotation or block quotation, the <blockquote> tag can be used.
- A list is a collection of items, which might be organized in a sequential or nonsequential manner. HTML supports three types of lists namely, ordered, unordered, and definition.
- HTML provides background properties that specify the background color and image for the Web pages.

# *Try it Yourself*

1. Create a Web page and insert the content provided containing the same formatting style as shown.

   **Sample Content**

   A **multi-tier enterprise application** consists of two types of objects: **application logic components** and **persistent data objects**. Application logic components represent actions and define methods that perform common tasks such as calculating the order price, customer credit card billing, and so on. Session beans model the application logic components as they contain the logic to perform the application tasks.

   It is easy and simple to manage the business data present in large organizations as objects rather than as relational rows in a database. Business data when treated as objects are easy to handle and manage as they are compact in nature.

   Entities are not associated with only EJBs and can be used in **Java 2 Standard Edition (J2SE)** environment. The differences between an Entity and a Session bean are as follows:

   - Entities have a client-visibility and a persistent identity that is distinct from their object reference. Each entity is distinct from the other because of their identities. Clients can use identities to pass the reference of an entity to other applications. However, this is not possible with Session beans.
   - Entity bean cannot be accessed remotely, whereas Session beans can be accessed locally as well as remotely.
   - Entity's lifetime is not dependent on the application's lifetime. Entity bean have a much longer lifetime than a client's session as it depends on how long the data is present in the database.
   - Entities represent data in a storage that is permanent as well as fault-tolerant and hence, they are long lasting and can survive application server or database crash. This data can be reconstructed in the memory. Hence, it can be said that entity is an in-memory representation of persistent data that:
     - Can be loaded from persistent store and its field populated with the stored data
     - Can be modified in-memory to change the data values
     - Can be saved and correspondingly database is updated
2. Apply bold formatting for the required words as shown in the sample content.
3. Apply an unordered bullet list to the text as shown in the sample content.

# Session - 4

# Creating Hyperlinks and Anchors

Welcome to the Session, **Creating Hyperlinks and Anchors**.

This session explains hyperlinks and their absolute and relative paths. This session also explains the ways to hyperlink a Web page, e-mail address, or any other content. Finally, the session explains the procedure to create anchors on a Web page.

In this Session, you will learn to:

- Describe hyperlinks
- Explain absolute and relative paths
- Explain how to hyperlink to a Web page and e-mail address
- Explain how to hyperlink to anchors and other content

## *4.1 Introduction*

A Web site is a collection of Web pages, which facilitate sharing information on the Internet. There should be some mechanism to allow the user to navigate through the Web pages of a Web site. It should also allow the user to view the Web pages of other Web sites, if required. These tasks can be achieved by creating hyperlinks in HTML.

## *4.2 Hyperlinks*

A hyperlink is referred to as a link. It refers to linking to another Web page or to a section in the same Web page.

The A (anchor) element is used to create a hyperlink. You can specify a text or an image as a hyperlink. When you move the mouse over such content, the cursor changes into a hand with its index finger pointing towards the content. This means that clicking them will take you to the respective link. To specify the linked page section or linked Web page, you must use the attributes of the A element. Table 4.1 lists the attributes of the A element.

| Attribute | Description |
|---|---|
| `href` | Specifies the URL of the Web page to be linked or the value of the name attribute. |
| `hreflang` | Indicates the language of the destination URL. |
| `name` | Specifies the section of the Web page, which is to be linked. |

**Table 4.1: Attributes of the A Element**

**Note** - The A element cannot contain any other A elements.

The basic syntax to provide a hyperlink is always the same. The `<a>` tag is used to provide a hyperlink. This contains the href= attribute that would contain the link to a URL or path of a Web page. An example of a href attribute code is as follows:

```
<a href=" http://www.aptech-worldwide.com/">
```

The description and reference text that will serve as a hyperlink must be provided before closing the `<a>` tag by using `</a>`.

An example of a hyperlink along with its output is as follows:

**Example**:

```
<html>
    <head>
    </head>
    <body>
        <a href="http://www.aptech-woldwide.com/">
        Click to view the Aptech Web site</a>
    </body>
</html>
```

The output of the example is shown in figure 4.1.

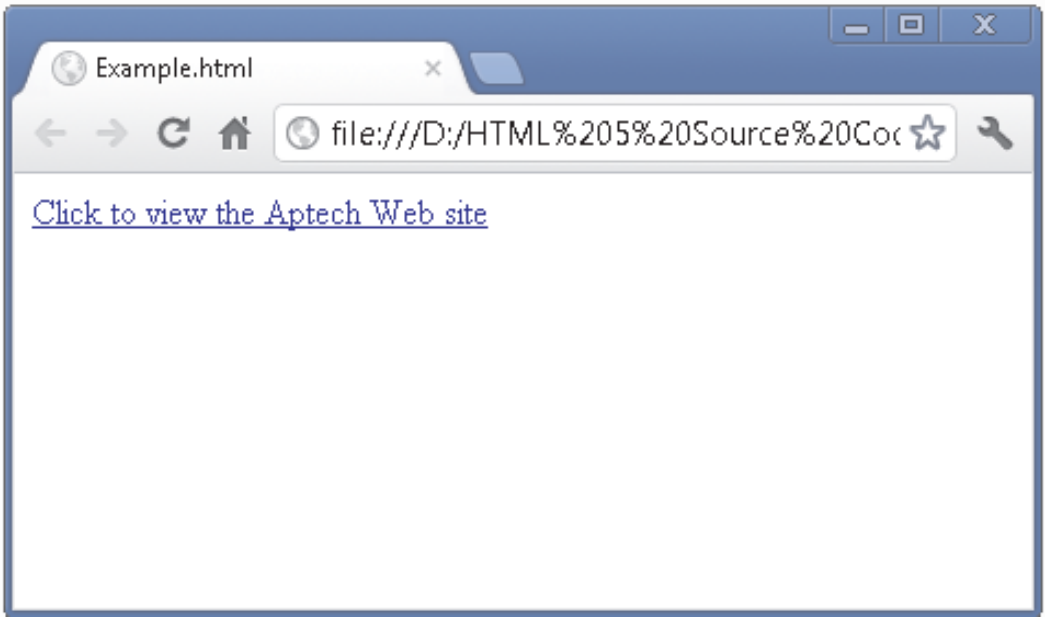



**Figure 4.1: Output of `a href`**

## *4.2.1 Target Attributes*

The `target` attribute of the `A` element specifies the location where the linked Web page will open when a link is clicked.

You can assign values to the `target` attribute. Table 4.2 lists some of the values of the `target` attribute.

| Value | Description |
|---|---|
| `_blank` | Loads the target URL in a new blank window. |
| `_self` | Loads the target URL in the same window as that of the current Web page. |
| `_top` | Loads the target URL in the complete area of window. |

**Table 4.2: Values of the `target` Attribute**

## *4.3 Absolute and Relative Paths*

Absolute paths are links that contain the complete address to get to a Web page. Absolute paths are the best way to link to a Web site. The syntax of an absolute path is as follows:

**Syntax:**

```
<a href=" http://www.aptech-worldwide.com/pages/about-us/
aboutus_aboutaptechworldwide.html ">Aptech Web site</a>
```

Relative paths are links that are provided when the files of a Web page are in the same folder as the page displaying the link. The syntax of a relative path is as follows:

**Syntax:**

```
<a href="aboutus_aboutaptechworldwide.html">Aptech Web site</a>
```

To link to the files present in the subfolder, you need to provide the path to the subfolder. For example, if the file `aboutus_aboutaptechworldwide.html` is in a subfolder named `about-us` then the syntax is as follows:

**Syntax:**

```
<a href="about-us/aboutus_aboutaptechworldwide.html">Aptech Web site</a>
```

Files that are present in folders that are one level up can also be linked using a relative path. The syntax to link to a file one level up is as follows:

**Syntax:**

```
<a href="../aboutus_aboutaptechworldwide.html">Aptech Web site
</a>
```

## *4.4 Hyperlink to an E-mail Address*

Hyperlinks can be even applied to e-mail addresses in the same way as they can be given for Web pages. There are various tasks that can be performed when a hyperlink are given to an e-mail. Some of these tasks include starting the default e-mail client, creating a new message, inserting the recipients address, adding the subject line, and so on.

To add an e-mail to a hyperlink, the `href=` attribute must be used and followed by `mailto:email address`. Code Snippet 1 shows the way to hyperlink an e-mail address.

**Code Snippet 1**:

```
<a href=”mailto:customercare@aptech.ac.in”>Customer Care</a>
```

To automatically add a subject line in the new e-mail message, the `?subject= attribute` must be inserted after the e-mail address. Code Snippet 2 shows the way to add a subject line to a hyperlinked e-mail address.

**Code Snippet 2**:

```
<a href=”mailto:customercare@aptech.ac.in?subject=E-mail to Customer
Care”>Customer Care</a>
```

## *4.5 Hyperlink to Other Types of Content*

Hyperlinks can be used to not only refer to another Web page or e-mail address but also can be used to link to other files and documents. Some of the files that are commonly linked on Web pages using hyperlinks are zipped files (.zip), executable files (.exe), documents (.doc), PDF reader files (.pdf), and so on. Hyperlinks can also be used to link to graphical .jpg and .gif files. To specify a file instead of the Web page, the name of the file must be provided in the `<a>` tag as shown in Code Snippet 3.

**Code Snippet 3**:

```
<a href=”Compressed.zip”>Click to download the compressed zip file</a>
```

## *4.6 Check Your Progress*

1. Which of the following attribute is used to provide a hyperlink?

| | | | |
|---|---|---|---|
| (A) | `<a>` | (C) | `<b>` |
| (B) | `<h>` | (D) | `<em>` |

2. Which of the following value loads the target URL in the same window of the current Web page?

| | | | |
|---|---|---|---|
| (A) | `_blank` | (C) | `_top` |
| (B) | `_self` | (D) | `_bottom` |

3. Which of the following paths has links that contain the complete address to get to a Web page?

| | | | |
|---|---|---|---|
| (A) | Absolute | (C) | Relative |
| (B) | Non-Absolute | (D) | Non-Relative |

4. To automatically add a subject line in the new e-mail message, the _____ attribute must be inserted after the e-mail address.

| | | | |
|---|---|---|---|
| (A) | `?subject=` | (C) | `?subject line=` |
| (B) | `?subjectline=` | (D) | `?topic=` |

5. The _____ attribute of the A element specifies the location where the linked Web page will open when a link is clicked.

| | | | |
|---|---|---|---|
| (A) | `subject` | (C) | `target` |
| (B) | `object` | (D) | `pageopen` |

## *4.6.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | B |
| 3. | A |
| 4. | A |
| 5. | C |

# Summary

- A hyperlink is referred to as a link. It refers to linking to another Web page or to a section in the same Web page.
- The A (anchor) element is used to create a hyperlink.
- The target attribute of the A element specifies the location where the linked Web page will open when a link is clicked.
- Absolute paths are links that contain the complete address to get to a Web page.
- Relative paths are links that are provided when the files of a Web page are in the same folder as the page displaying the link.
- To add an e-mail to a hyperlink, the href= attribute must be followed by mail to: email address.
- Hyperlinks can also be used to link to files and documents such as zipped files (.zip), executable files (.exe), documents (.doc), PDF reader files (.pdf), and so on.

# *Try it Yourself*

1. **Solution Anywhere** is an open source software development organization headquarters at Chicago, USA. The company wants to make its presence felt worldwide by creating a Web site which will highlight the activities performed by the organization. The organization also provides all the software that are available as a freeware. You as a Web site developer for the organization have been asked to create the following four separate Web pages for the organization:

   a. Home

   b. About Us

   c. Downloads

   d. Contact Us

2. Link all four Web pages to each other using hyperlinks on all four pages.

3. In the Downloads page, provide a .EXE file along with a downloadable PDF or Document help file.

4. In the Contact Us page, provide an e-mail address that is hyperlinked. This e-mail link must open a new e-mail with the Subject line as 'New E-mail'.

Onlinevarsity

# INDUSTRY BEST PRACTICES

## SYNC WITH THE INDUSTRY

# Session - 5

# Introduction to CSS3

Welcome to the Session, **Introduction to CSS3**.

This session explains the concept of CSS along with the CSS structure and rules. It also explains the different approaches that can be employed by the user for defining the styles and linking the stylesheets.

In this Session, you will learn to:

- Identify the new functions of CSS3
- Explain the different types of selectors
- Explain nested tags
- Define Classes and IDs for applying styles
- Explain the process to apply styles to hyperlink

## *5.1 Introduction*

CSS is a style sheet language used for informing the browser how to present a document. It uses markup language for describing the presentation semantics of a document. In other words, an HTML document defines the content of the file whereas the CSS file defines how HTML elements are to be displayed.

## *5.2 Cascading Style Sheet 3*

CSS is a mechanism used for adding style such as fonts, colors, and spacing to Web documents. CSS has multiple levels and profiles. Each level of CSS is updated from the earlier version, by adding new features. CSS version are denoted as CSS1, CSS2, CSS3, and CSS4, where the numbers are different for each version or level.

**Note** - The drafting of CSS4 was started by W3C on Sep 29, 2009. However, it is currently not supported by any Web browser.

CSS3 is divided into multiple documents called “modules”. Each of these modules have new capabilities or extends the features present in CSS2. Drafting of CSS3 started when publication of the original CSS2 recommendation was released. The first CSS3 drafts were released on June 1999. CSS3 extends variety of new ways to create an impact with any designs, with quite a few important changes. The CSS3 logo is displayed in figure 5.1.

**Figure 5.1: CSS3 Logo**

## *5.2.1 Modules*

Since CSS3 is available as modules and is still evolving, there are many modules having different stability and status. Out of the fifty modules published by the CSS working group, only three modules are released as recommendations and they are as follows:

- CSS Color Level 3
- CSS Namespaces
- Selectors Level 3

The following modules are stable and in recommendation stage:

- Media Queries
- CSS style Attributes

The following modules are in testing phase and in recommendation stage:

- CSS Backgrounds and Borders Level 3
- CSS Image Values and Replaced Content Level 3
- CSS Marquee
- CSS Multi-column Layout
- CSS Speech
- CSS Mobile Profile 2.0
- CSS TV Profile 1.0

The following modules are in refining phase and in working draft stage:

- CSS Transforms
- CSS Transitions
- CSS Values and Units Level 3
- CSS Print Profile

The following modules are in revising phase and in working draft and recommendation stage:

- CSS Animations
- CSS Flexible Box Layout
- CSS Fonts Level 3
- CSS Paged Media Level 3
- CSS Text Level 3
- CSS Basic User Interface Level 3
- CSS Writing Modes Level 3
- CSSOM View

The following modules are in exploring phase and in working draft stage:

- CSS Cascading and Inheritance Level 3
- CSS Conditional Rules Level 3
- CSS Grid Layout
- CSS Grid Template Layout
- CSS Line Grid
- CSS Lists Level 3
- CSS Tables Level 3
- Selectors Level 4
- CSS Object Model

The following modules are in rewriting phase and in working draft stage:

- CSS Line Layout Level 3

- CSS Ruby
- CSS Syntax Level 3

The following modules are in abandoned phase and in working draft stage:

- Behavioral Extensions to CSS
- CSS Hyperlink Presentation

## *5.2.2 CSS Syntax*

The general syntax of CSS consists of three parts namely, selector, property, and value. A selector is an HTML element for which you want to specify the style or the formatting instruction. A property of a selected element is a CSS property that specifies the type of the style to be applied to the selector. CSS allows controlling the appearance of the content by providing various properties. These properties include text properties, positioning properties, font properties, color properties, and so on. A value refers to the value of the CSS property. A CSS property can have multiple values. For example, the values of the color property include red, green, yellow, and so on.

The property and the value for a selector are separated with a colon (`:`). They are enclosed within the curly brackets (`{ }`) that is known as the declaration block. Figure 5.2 shows a CSS syntax.

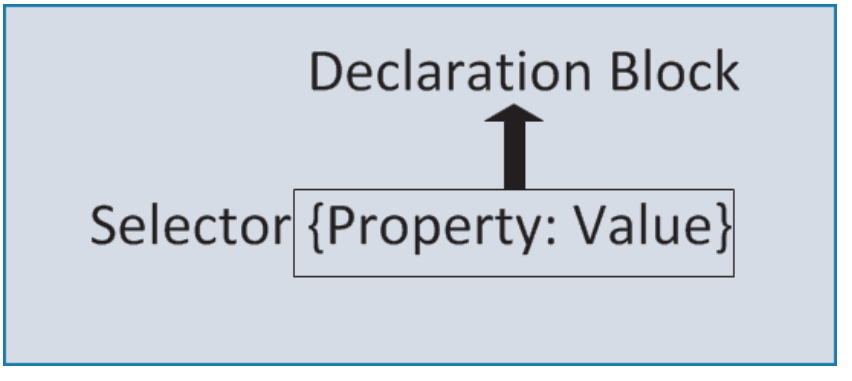



**Figure 5.2: CSS Syntax**

You can have various combinations to specify rules for HTML elements. First, you can specify multiple property-value pairs for a selector, which are separated by a semicolon (`;`) within the declaration block. Second, you can specify multiple selectors for a single property by grouping the selectors. To group the selectors, the selectors are separated by commas followed by a declaration block of properties and values. Third, you can specify properties for multiple selectors. Here, the comma-separated selectors are followed with multiple property-value pairs.

Figure 5.3 shows multiple properties and selectors.

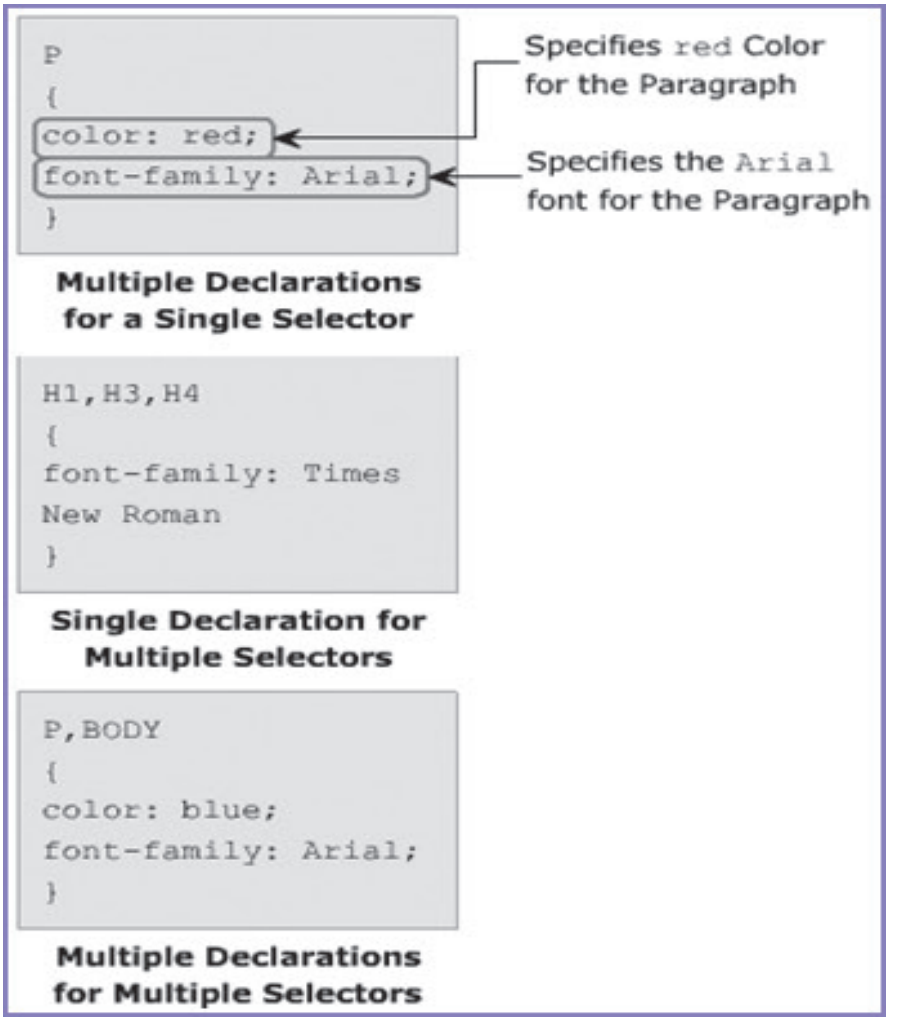



Figure 5.3: Multiple Properties and Selectors

## *5.2.3 Length Measurement Units*

CSS uses various units of measurements for specifying size of the font, width and height of margins, and so on. These units measure the horizontal and vertical length of the content. CSS supports two types of length measurement units namely, relative and absolute.

➔ **Relative**: Relative length specifies the length units related to other length property that are calculated in comparison to a current value.

Table 5.1 lists the relative length units.

| Relative Length | Description |
|---|---|
| `em` | Specifies the font size (height) of a particular font. The `em` unit is relative to the value of the `font-size` property of the selector. |
| `ex` | Specifies the 'x-height' of a particular font. The 'x-height' value is approximately half the font size or the height of the lowercase letter 'x'. |
| `px` | Specifies the size in pixels, which is relative to the screen of the device. |

**Table 5.1: Relative Length Units**

Figure 5.4 shows relative lengths.

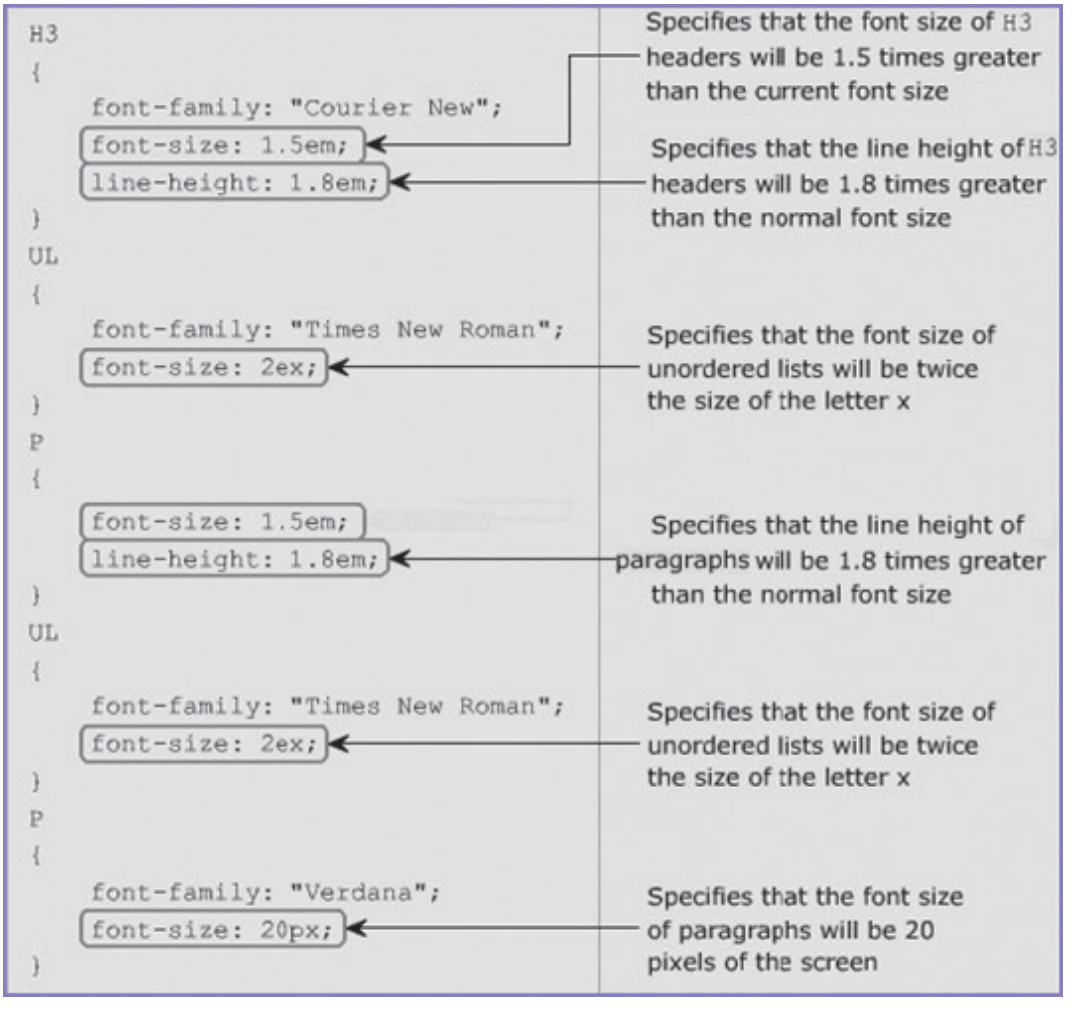



**Figure 5.4: Relative Lengths**

- **Absolute**: Absolute lengths are specified when the Web page designer is aware of the physical properties of the output device. These are specific and fixed values. Table 5.2 lists the absolute lengths.

| Absolute Length | Description |
|---|---|
| in | Specifies the size in inches, where 1 inch = 2.54 centimeters. |
| cm | Specifies the size in centimeters. |
| mm | Specifies the size in millimeters. |
| pt | Specifies the size in points, where 1 point = 1/72th of an inch. |
| pc | Specifies the size in picas, where 1 pica = 12 points. |

Table 5.2: Absolute Lengths

Figure 5.5 shows absolute lengths.

```
OL
{
    font-family: "Times New Roman";
    font-size: 0.5cm;
}
TD
{
    font-size: 0.2in;
}
CAPTION
{
    font-size: 3mm;
}
```

Figure 5.5: Absolute Lengths

- **Percentages**: Percentage allows specifying the length of the content, which is relative to another value.

Figure 5.6 shows the use of percentage in defining the style.

```
H1
 {
     font-size: 120%;
     line-height: 200%;
 }
```

**Figure 5.6: Use of Percentage in Defining the Style**

In the figure, the CSS code specifies the styles for the `H1` element. The `font-size` property is set to a value of `120%`. This means that the size of the header will appear `20%` greater than its current size. The `line-height` property is set to the value `200%`. This means that the height of the line will be double the value of the `font-size` property.

## *5.3 Types of Style Sheets*

There are three types of `style` sheets namely, inline, internal or embedded, and external style sheets. An inline style sheet uses the style attribute within an HTML element to specify the style for HTML elements.

An internal style sheet is also included within the HTML document. However, it is defined using the `style` element within the `style` element. The style rules appear in a declaration block for each HTML element under the `style` element. The type attribute of the `style` element specifies the content `type`, which is `text/css`. This means that the content under the `style` element is CSS code. You can specify any combinations of specifying style rules. The style rules specified for an element will be applied to all the sub-elements. Internal style sheets are useful when styles are to be applied to a specific Web page.

### *5.3.1 Internal/Embedded Styles*

Internal styles are placed inside the `<head>` section of a particular Web page source code. These styles can be re-used in the same Web page in which they are placed.

Figure 5.7 shows an internal style.

```
<head>
  <meta charset="utf-8">
  <title>Sample HTML5 Structure</title>
  <style>
  h1,h2{
  margin:0px;
  font-size:1.5em;
  }
  footer{
  background-color:#999;
  text-align:center;
  }
  </style>
</head>
```

**Figure 5.7: Internal Style**

In figure 5.7, inside the `<style>` tag, CSS styles for `<h1>`, `<h2>`, and `<footer>` tags are defined. This can be re-used in the same Web page multiple times.

## *5.3.2 Inline Styles*

Inline styles are placed directly inside an HTML element. A Web designer cannot use the style builder to create an inline style. Inline style cannot be reused at any point of time in a Web page.

Code Snippet 1 demonstrates the use of an inline style for `<p>` tag.

**Code Snippet 1**:

```
<p style="font-size: 14px; color: purple;"></p>
```

## *5.3.3 External Style Sheet*

An external CSS is defined in a separate file and is saved with the `.css` extension. It provides the benefit of reusability by implementing common style rules for multiple HTML pages. Hence, external CSS are widely used to provide a consistent look across the Web pages of a Web site.

Figure 5.8 shows an example of external CSS code.

```
BODY
{
        background-color: gray;
        font-family: arial;
        font-style: italic;
}
```

**Figure 5.8: External CSS Code**

Explanation for the code shown in figure 5.8 is as follows:

`background-color: gray;`

Specifies the background color of the Web page as `gray`.

`Font-family: arial;`

Specifies the font of the textual content as `arial`.

`Font-style: italic;`

Specifies the font style of the textual content as `italic`.

Figure 5.9 shows an example of HTML code using an external CSS style sheet.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<LINK rel="stylesheet" type="text/css" href="body.css"/>
<TITLE>Webex e-Server</TITLE>
</HEAD>
<BODY>
This is the fastest web server..!!
</BODY>
</HTML>
```

**Figure 5.9: HTML Code using an External CSS Style Sheet**

Explanation for the code shown in figure 5.9 is as follows:

`<LINK`

Specifies that the HTML page is linked to another object.

`rel=”stylesheet”`

Specifies that the linked object is a style sheet.

`type=”text/css” href=”body.css”/>`

Refers to an external style sheet of the content type, `text/css`.

**Note** - An inline style sheet holds the highest priority, which means that the style specified for the same element in any other style sheet will be ignored. The browser-specific styvles are applied to a Web page when there are no styles specified for a Web page.

## *5.4 Selectors*

Selectors refer to the HTML elements with the styles the users want to apply to them. The three different types of CSS selectors are as follows:

- Type selector
- Class selector
- ID selector
- Universal selector

### *5.4.1 Type Selector*

A type selector simply specifies the element name along with the styles to be applied to that element. This results in applying the specified styles to all the occurrence of that element in a Web page. Here, the styles are specified only once for an HTML element and are applied to all the occurrences of that element. Figure 5.10 shows an example of type selector.

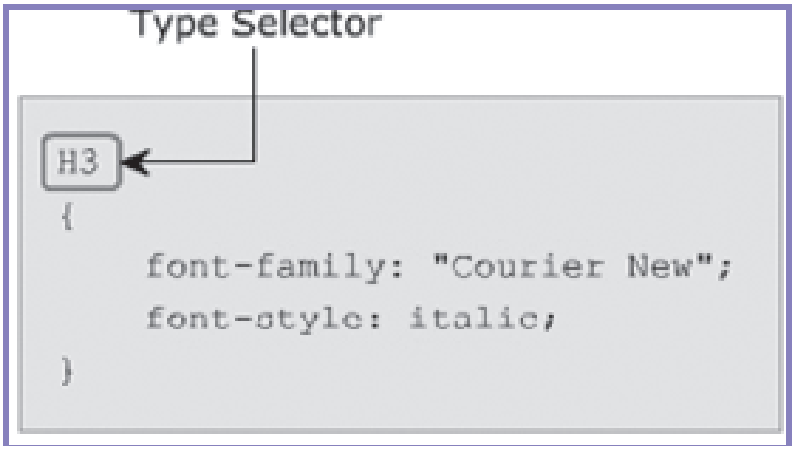



**Figure 5.10: Type Selector**

## *5.4.2 Class Selector*

A class selector matches elements, whose `class` attribute is set in an HTML page and applies styles to the content of all those elements. For example, if there are `span` and `div` elements in a Web page with their `class` attributes set, the style specified for the class selector will be applied to both the elements. A class selector starts with a period followed by the value of the class attribute. Figure 5.11 shows an example of class selector.

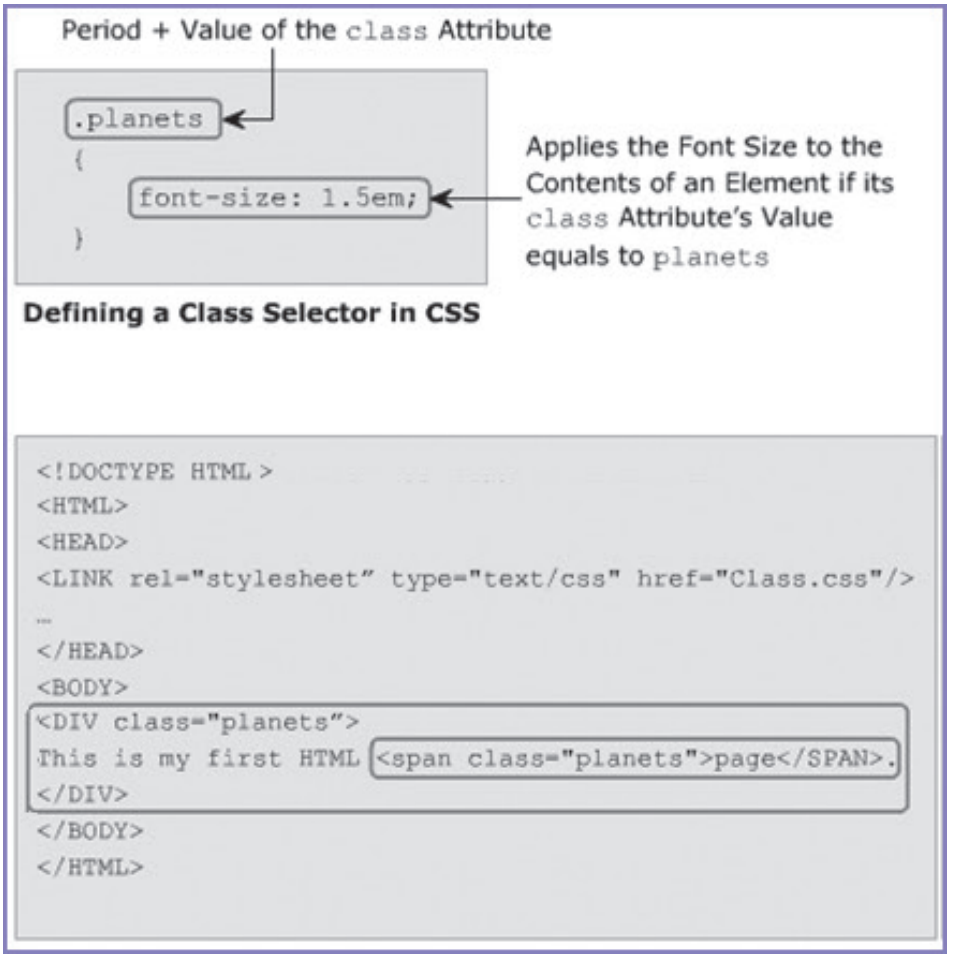



**Figure 5.11: Class Selector**

In figure 5.11, the style will be applied to both the places where the values of the class attribute have been set to **planets**. If the style is required to be applied to only the `<div>` element then class element would be used as shown in Code Snippet 2.

**Code Snippet 2**:

```
div.planets
     {
       font-size: 1.5em;
     }
```

## 5.4.3 ID Selector

An ID selector matches an element whose id attribute is set in an HTML page and applies styles to the content of that element. The ID selector specifies styles for an element whose `id` attribute is set to a unique value.

An ID selector starts with the hash symbol (`#`) followed by the id attribute's value and the declaration block. Figure 5.12 shows an example of ID selector.

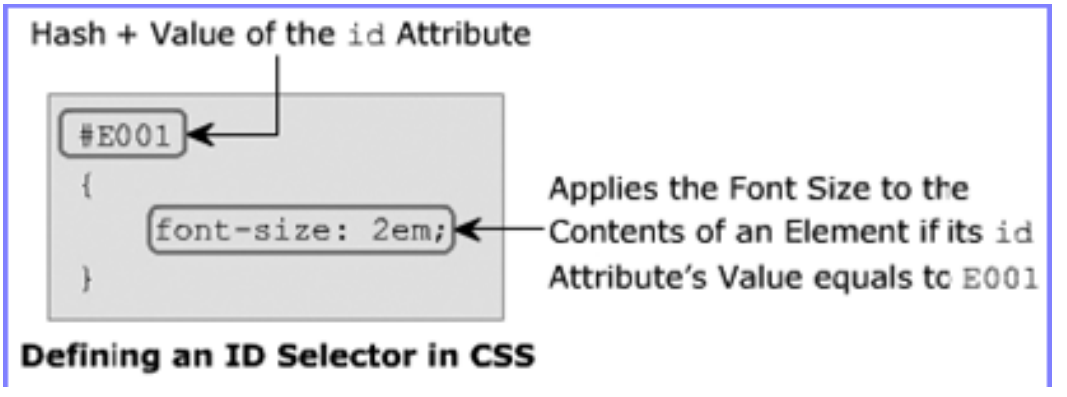



Figure 5.12: Example of ID Selector

## 5.4.4 Universal Selector

The universal selector can be applied to all elements in the document. This means that it applies the specified styles to the content of all the elements. It is represented by an asterisk (`*`) sign. For example, universal selector is used define the font family for all the elements as shown in Code Snippet 3.

**Code Snippet 3**:

```
* {
   font-family: Verdana, Calibri, sans-serif;
  }
```

## 5.4.5 Generic Cascading Order

Consider a scenario where you have multiple style sheets defined for an HTML page. These style sheets might have various selectors and multiple styles defined for an HTML element. Therefore, W3C has defined some rules for applying styles to an HTML element. These rules are as follows:

- Gather all the styles that are to be applied to an element.
- Sort the declarations by the source and type of style sheet. The source specifies the origin from where the styles are rendered.

The highest priority is given to the external style sheet defined by an author. The next priority is of the reader, which can be a software that reads the content (screen reader software), and the last priority is of the browser.

➔ Sort the declarations by the priority of a selector, where the ID selector has the highest priority.

➔ Sort the declaration according to the specified order.

Figure 5.13 shows the generic cascading order.

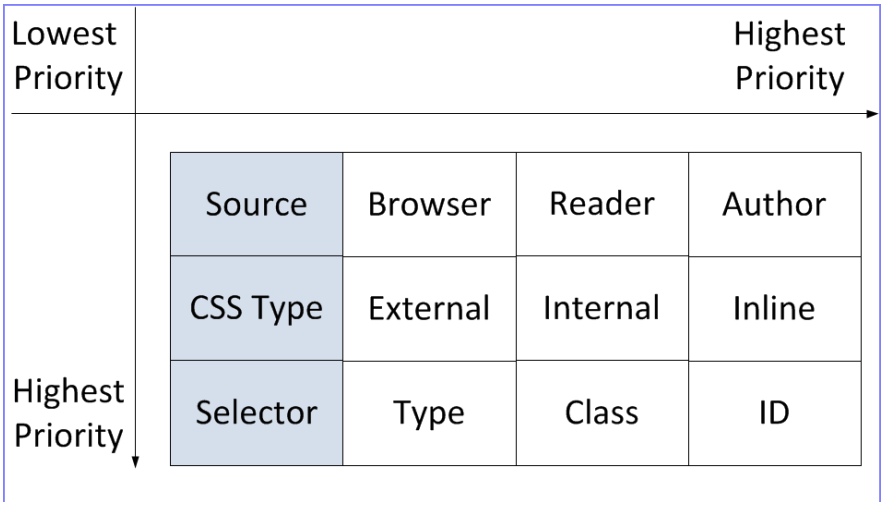



**Figure 5.13: Generic Cascading Order**

## *5.4.6 Comments*

A comment refers to the descriptive text that allows a Web page designer to provide information about the CSS code. Comments make the program more readable and help the designer to explain the styles specified for elements. This is helpful when other Web designers analyze the CSS code.

The browser can identify comments as they are marked with special characters, which are '/*' and '*/'. When the browser encounters these symbols, the text within them are ignored and are not displayed in the browser. You can have single-line and multi-line comments in the CSS file.

Figure 5.14 shows an example using comments in a CSS code.

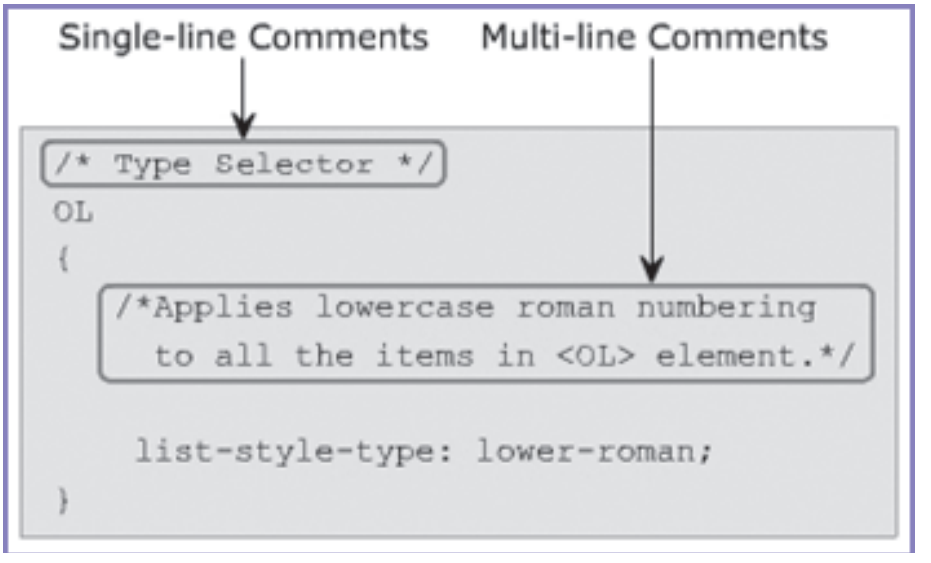



Figure 5.14: Comments in a CSS Code

## *5.5 Grouping and Nesting of Selectors*

In style sheets there are often elements with the same style.

Code Snippet 4 demonstrates elements with same style.

**Code Snippet 4**:

```
h1
{
 color:green;
}
h2
{
 color:green;
}
p
{
 color:green;
}
```

To reduce the code, developers can group selectors. Separate each selector with a comma. Code Snippet 5 displays the grouping of selectors.

**Code Snippet 5:**

```
h1,h2,p
{
 color:green;
}
```

It is also possible to apply a style for a selector within a selector. Code Snippet 6 demonstrates the nesting of selectors.

**Code Snippet 6**:

```
p
{
 color:green;
}
 .changed
{
 background-color:red
}
```

In the code, a style is specified for all the paragraphs and another style is specified for all elements whose class attribute has the value set to changed.

## *5.5.1 Pseudo Classes*

Consider a scenario where a Web site consists of multiple Web pages linked through hyperlinks. Browse through various Web pages by randomly clicking the links within the main page. At times, it might happen that unknowingly the same Web page get open that you have already visited. In such a case, you might feel the need for a mechanism that could differentiate the already visited links from the remaining ones. In CSS, this is possible by using pseudo classes.

Pseudo classes allow the users to apply different styles to the elements such as buttons, hyperlinks, and so on.

Table 5.3 lists the different states of an element.

| State | Description |
|---|---|
| active | Defines a different style to an element that is activated by the user. |
| hover | Defines a different style to an element when the mouse pointer is moved over it. |
| link | Defines a different style to an unvisited hyperlink. |
| visited | Defines a different style to the visited hyperlink. |

**Table 5.3: Different States of an Element**

The syntax demonstrates how to declare a pseudo class.

**Syntax**:

`selector_name:state_name {property: value}` where,

`selector_name`: Is an element name.

`state_name`: Is one of the states of an element.

`property`: Is any CSS property such as color, border, and font.

Table 5.4 lists selector name and its descriptions.

| Selector Name | Description |
|---|---|
| `:link` | Is used for selecting all unvisited links |
| `:visited` | Is used for selecting all visited links |
| `:active` | Is used for selecting the active link |
| `:hover` | Is used for selecting links on mouse over |
| `:focus` | Is used for selecting the input element which has focus |
| `:first-letter` | Is used for selecting the first letter of every `<p>` element |
| `:first-line` | Is used for selecting the first line of every `<p>` element |
| `:first-child` | Is used for selecting every <p> elements that is the first child of its parent |
| `:before` | Is used for inserting content before every `<p>` element |
| `:after` | Is used for inserting content after every `<p>` element |
| `:lang` (language) | Is used for selecting every <p> element with a lang attribute value |

**Table 5.4: Selector Name and its Descriptions**

Pseudo classes specify the styles to be applied on an element depending on its state. In CSS3, a selector can contain multiple pseudo-classes. These pseudo-classes should not be mutually exclusive. For example, the selectors `a:visited:hover` and `a:link:hover` are applicable, but `a:link:visited` is not applicable because `:link` and `:visited` are mutually exclusive selectors. The HTML code creates a form that accepts the customer details and provides a link that allows the user to view the bill as shown in figure 5.15.

```
<!DOCTYPE HTML >
<HTML>
<HEAD>
<TITLE>Bill Payment Form</TITLE>
<LINK rel="Stylesheet" type="text/css" href="Payment.css" />
</HEAD>
<BODY>
<H2>Payment Details</H2>
<FORM method="POST" action="">
<TABLE>
<TR>
<TD>Name:</TD>
<TD><INPUT type="text" /></TD>
</TR>
<TR>
<TD>Payment Mode:</TD>
<TD>
<SELECT>
<OPTION>Select...</OPTION>
<OPTION>Credit Card</OPTION>
<OPTION>Cash</OPTION>
</SELECT>
</TD>
</TR>
<TR>
<TD>Total Amount:</TD>
<TD><INPUT type="text" /></TD>
</TR>
</TABLE>
</FORM>
<A href="printform.html">Click here to view the bill.</A>
</BODY>
</HTML>
```

**Figure 5.15: HTML Code**

CSS code specifies the different styles for the visited links, unvisited links, and for the links when the mouse hovers over it.

Figure 5.16 shows a style sheet code.

```
a:link
{
        color: white;
        background-color: black;
        border: 2px solid white;
}

a:visited
{
        color: white;
        background-color: brown;
        border: 2px solid white;
}

a:hover
{
        color: black;
        background-color: white;
        border: 2px solid black;
}
```

**Figure 5.16: Style Sheet Code**

Explanation for the code shown in figure 5.16 is as follows:

```
a:link
{
  color: white;
  background-color: black;
  border: 2px solid white;
}
```

Specifies the styles for an unvisited link.

```
a:visited
{
  color: white;
  background-color: brown;
  border: 2px solid white;
}
```

Specifies the styles for a visited link.

```
a:hover
{
  color: black;
  background-color: white;
  border: 2px solid black;
}
```

Specifies the styles for a link when a mouse hovers over it.

Figure 5.17 shows an example output of mouse hovered link.

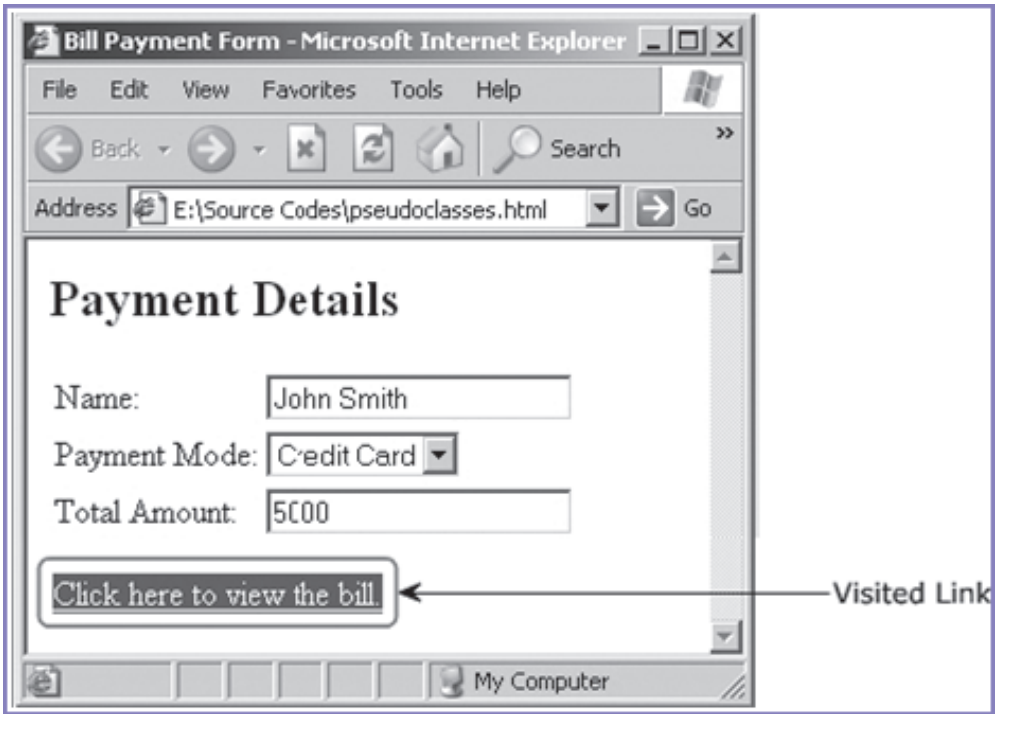



**Figure 5.17: Output of Mouse Hovered Link**

## *5.5.2 Purpose of Pseudo Elements*

Consider a scenario where you are designing a Web site that explains the important technical terms. While defining such terms, you might feel the need to emphasize more on the first letter by applying different styles. It becomes difficult if you try to apply styles only on the first letter of a line or paragraph. This can be achieved by using the pseudo elements.

Pseudo elements provide you with a flexibility to apply styles to a specific part of the content such as a first letter or first line. This allows you to control the appearance of that specific content without affecting the rest of the content.

Figure 5.18 shows the purpose of pseudo elements.

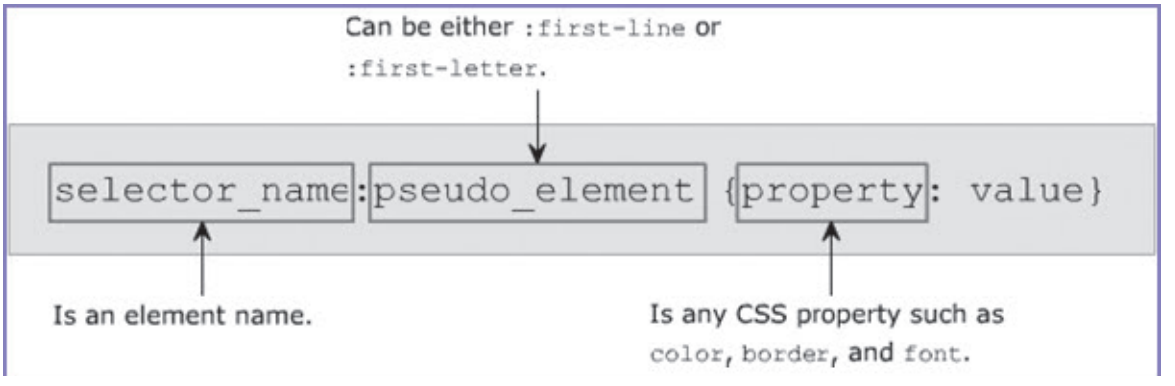



**Figure 5.18: Purpose of Pseudo Elements**

Pseudo element adds some special effects to HTML elements such as `<p>`, `<body>`, and so on.

## Pseudo Elements

The `:first-line` and `:first-letter` pseudo elements allow you to apply styles to the first line and first letter respectively.

### ➔ :first-line

The `:first-line` pseudo element allows you to apply styles to the first line.

Figure 5.19 shows an HTML code where the `:first-line` pseudo element will be used.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>E-Commerce</TITLE>
<LINK rel="Stylesheet" type="text/css" href="E-commerce.css" />
</HEAD>
<BODY>
<H2>E-Commerce</H2>
<P>E-commerce (Electronic commerce) is defined as the sale and
purchase of products over the Internet. E-mail, accounting,
shipment information, and enterprise information reporting are the
some common applications of e-commerce.</P>
</BODY>
</HTML>
```

**Figure 5.19:** `::first-line` **Pseudo Element**

The Code Snippet 9 in the style sheet declares the style that will be applied to the first line in the paragraph.

**Code Snippet 9**:

```
p:first-line
{
font-family: "Tahoma";
font-weight: bolder;
background-color: #FFFFCC;
}
```

Specifies the styles to be applied to the first line of the paragraph content.

➔ **:first-letter**

The `:first-letter` pseudo element allows you to apply styles to the first letter.

Figure 5.20 shows an example of the HTML code for the `:first-letter` pseudo element.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Physical Chemistry</title>
    <link rel="stylesheet" type="text/css" href="Cefine.css"></link>
  </head>
  <body>
<h2>Physical Chemistry</h2>
<p>Physical chemistry is a branch of chemistry that analyzes the physical
chemicals.</p>
```

**Figure 5.20:** `:first-letter` **Pseudo Element**

Figure 5.21 shows CSS code for the `:first-letter` pseudo element.

```
p:first-letter
{
 font-family:fantasy;
 font-size:xx-large;
 font-weight:bold;
 }
```

**Figure 5.21: CSS Code for the** `:first-letter` **Pseudo Element**

Explanation for the code shown in figure 5.21 is as follows:

`p:first-letter`

Specifies the styles to be applied on the first letter of the paragraph content.

Figure 5.22 shows the output of `:first-letter` pseudo element.

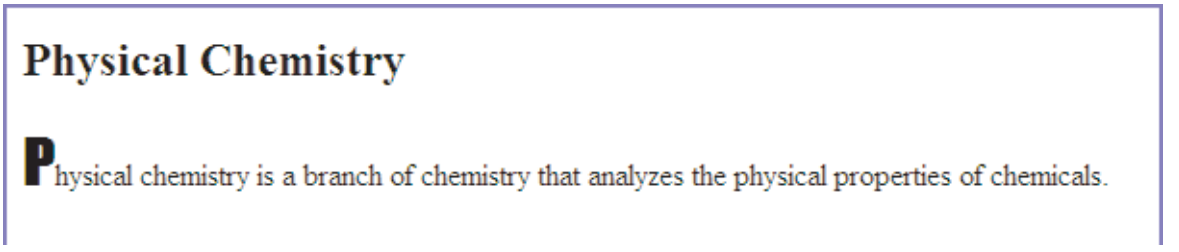



**Figure 5.22: Output of `:first-letter` Pseudo Element**

## *5.6 Styles to Hyperlink*

CSS can be used to change the appearance and behavior of hyperlinks. To do this, use the following selectors/pseudo-classes:

- ➔ `a`
- ➔ `a:link`
- ➔ `a:visited`
- ➔ `a:hover`
- ➔ `a:active`

This selectors/pseudo classes represent the 'anchor' element (specified using the HTML 'a' tag) and its various states.

There are two other ways to assign hyperlink styles. They are as follows:

1. Div specific
2. Link specific

### *5.6.1 ID Specific Hyperlink Styles*

A hyperlink styles can be created and assigned to a specific `div`. This will have all the hyperlinks present within the div to follow the specified rules. It is irrelevant if the div is an (`#`) `id` or (`.`) `class`.

Code Snippet 10 displays the style for a `div id` named `navone`.

**Code Snippet 10**:

```
#navone a:link {
     text-decoration: underline;
     color: #005;
     background: transparent;
}
#navone a:visited {
text-decoration: none;
     color: #FFA500;
     background: transparent;
}
#navone a:hover {
     text-decoration: none;
     color: #FFA500;
     background: transparent;
}
#navone a:focus {
     text-decoration: none;
     color: #FFA500;
     background: transparent;
}
#navone a:active {
     text-decoration: none;
     color: #FFA500;
     background: transparent;
}
```

## *5.6.2 Class Specific Hyperlink Styles*

Specific styling can be assigned to a specific type of hyperlink. This is achieved by creating the style rules in the CSS. For this type of hyperlink styling, a class is used generally than an id. A point to note that an id can only be used once on a page whereas a class can be used multiple times as required.

Code snippet 11 displays the use of CSS and HTML file containing a hyperlink with the value of class set to `navone`.

**Code Snippet 11**:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
<style>
.navone:link {
     text-decoration: underline;
     color: #FFF;
     background: #008;
     font-size:30px;
}
.navone:visited {
     text-decoration: none;
     color: #FFF;
     background: #06a;
}
.navone:hover {
text-decoration: none;
     color: #FFF;
     background: #000;
}
.navone:focus {
     text-decoration: none;
     color: #FFF;
     background: #06b;
}
.navone:active {
     text-decoration: underline;
     color: #FFF;
```

```
background: #06F;
}
</style>
 </head>
 <body>
<a href="6.html" class="navone">LinkText</a>
 </body>
</html>
```

This link will use the style rule class of `navone` even if it is placed inside a `div` that has `div` specific hyperlink style rules.

Figure 5.23 shows output of a class specific hyperlink.

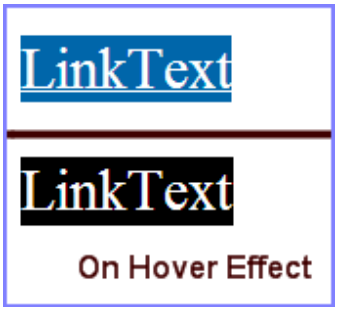



**Figure 5.23: Output of a Class Specific Hyperlink**

## *5.7 Check Your Progress*

1. Which of the following statements are true for class selector?

| | |
|---|---|
| (A) | It starts with a period followed by the value of the class attribute |
| (B) | It matches elements and applies the style to the content of those elements whose class attribute is same |
| (C) | It specifies the element name along with the style |
| (D) | It starts with a hash symbol followed by the value of the class attribute |

2. Absolute lengths are specified when the Web page designer is aware of the _____of the output device.

| | | | |
|---|---|---|---|
| (A) | Text properties | (C) | Image properties |
| (B) | Physical properties | (D) | Positioning properties |

3. Which of these options represent valid style sheets?

| | | | |
|---|---|---|---|
| (A) | Vertical | (C) | Horizontal |
| (B) | Inline | (D) | Embedded |

4. Which of these options are valid selectors?

| | | | |
|---|---|---|---|
| (A) | ID | (C) | External |
| (B) | Inline | (D) | Class |

5. Which of the following statements are valid for CSS?

| | |
|---|---|
| (A) | Can specify multiple property-value pairs for a selector |
| (B) | Can specify multiple property-value pairs for selector separated by a comma |
| (C) | Can specify multiple selectors for a single property by grouping the selector |
| (D) | Can specify only a single selector for a single property |

## *5.7.1 Answers*

| | |
|---|---|
| 1. | A, B |
| 2. | B |
| 3. | B, D |
| 4. | A |
| 5. | A, C |

# Summary

- CSS is a mechanism for adding style such as fonts, colors, and spacing to Web documents. CSS has multiple levels and profiles.
- The general syntax of CSS consists of three parts namely, selector, property, and value.
- Selectors refer to the HTML elements with the styles that are applied to them and they can be Type, Class, ID, or Universal selectors.
- A comment refers to the descriptive text that allows a Web page designer to provide information about the CSS code.
- Pseudo classes allow the users to apply different styles to the elements such as buttons, hyperlinks, and so on.
- Psuedo elements allow the developer to apply styles to a specific part of a content such as first letter or first line.
- A hyperlink style can be assigned either through DIV or through link class.

# *Try it Yourself*

1. Joan O'Brien works for a famous jewelry design company headquartered at Seattle, USA. The management of the company wants to increase the sale of the jewelry produced by reaching out to maximum number of clients spread across the globe. This could be achieved by creating a user friendly and attractive Web site for the company. Joan has been asked to develop the Web site and make the site attractive by applying different styles to the contents displayed at the Web site.

Onlinevarsity

# Session - 6

# Formatting Using Style Sheets

Welcome to the Session, **Formatting Using Style Sheets**.

CSS is accepted as the standard for defining the presentation of content in HTML pages. This session explains about text formatting and styling of text through CSS. This session also explains paragraph indenting and applying border style using CSS. Finally, this session explains horizontal alignment and vertical spacing in a paragraph.

In this Session, you will learn to:

- List and explain text and font styles
- Describe inline spans
- Explain paragraph indentation and application of border
- Explain horizontal paragraph alignment
- Explain vertical spacing within a paragraph

## *6.1 Introduction*

Earlier Web designers had limitations in the presentation of text. `<font>` tags were used to change the color and typeface of the text. Sizing was still a concern and the designers used the pre-defined font sizes. Effects such as boldness and strike-through were possible only through basic forms of HTML tags. Even, applying different colors to borders and alignment of a paragraph was a concern. Since, Web page presentation has become an important aspect of Web designing, the stylesheets allow various styles for formatting texts, borders, or paragraphs.

## *6.2 Text and Font Style*

The text properties specify and control the appearance of the text in a Web page. You can change the color of a text, increase or decrease the space between characters, align a text, and so on using the text properties. Table 6.1 lists different text properties.

| Property | Description |
|---|---|
| `color` | It is used for specifying the color of the text. |
| `text-align` | It is used in specifying the horizontal alignment of text in an element. |
| `text-decoration` | It is used for specifying the decoration of the text in an element. |
| `text-indent` | It is used for specifying the indentation of first line of text in an element in `length` or `%`. |
| `text-transform` | It is used in specifying the casing of text in an element. |
| `word-spacing` | It is used for increasing or decreasing the space between words. |

**Table 6.1: Different `text` Properties**

The font properties allow you to specify the font for the text. They allow you to change the different font attributes of the text such as font, size, and style of the text. The browser must support the font specified by the font properties. Otherwise, it will display the default font, which is dependent on the browser. Table 6.2 lists the different font properties.

| Property | Description |
|---|---|
| `font-family` | It is used for specifying the font and can specify a generic family or a specific family name such as 'Serif' or 'Times New Roman'. |
| `font-size` | It is used for specifying the size of the font and can have an absolute or relative value. |
| `font-style` | It is used for specifying the style of the font. |
| `font-variant` | It is used for specifying whether the text should be displayed in small-caps. |

**Table 6.2: Different `font` Properties**

## *6.2.1 Text Styles*

The different text styles such as `text-align`, `text-indent`, and `text-transform` provide different values that allow specifying the alignment, indentation, and casing of text in an element.

The `text-align` property allows the text to be centered, or left or right aligned, or justified. Table 6.3 lists the values of `text-align` property.

| Property | Description |
|---|---|
| `left` | It is used for aligning the text to the left of the Web page. |
| `right` | It is used for aligning the text to the right of the Web page. |
| `center` | It is used for aligning the text in the middle of the Web page. |
| `justify` | It is used for justifying the text on both sides of the Web page. |

**Table 6.3: Values of `text-align` Property**

As discussed the text-indent property is used for specifying the indentation of the text. Table 6.4 lists the values of `text-indent` property.

| Value | Description |
|---|---|
| `length` | It is used in specifying fixed indentation and the default value is 0. |
| `%` | It is used in specifying an indentation as a percentage of the width of the parent element. The parent element is the element within which the selector element is defined. |

**Table 6.4: Values of `text-indent` Property**

The `text-transform` property is for changing the case of letters in a text. Table 6.5 lists the values of `text-transform` property.

| Value | Description |
|---|---|
| `none` | It is used in specifying that the text will be displayed with the same casing as written within the element. |
| `capitalize` | It is used in specifying that the first letter of each word will be capitalized. |
| `uppercase` | It is used in specifying only uppercase letters. |
| `lowercase` | It is used in specifying only lowercase letters. |

**Table 6.5: Values of `text-transform` Property**

Figure 6.1 shows DIV element HTML code.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>

<LINK rel="stylesheet" type="text/css" href="TextProperties.css"/>

<TITLE>Client</TITLE>
</HEAD>

<BODY>
<H2>Client Contact Information</H2>

<DIV>
<H4>Dynamic Solutions</H4>
<P>Tel Number - 445 558 7744</P>
<P>Fax Number - 703 740 6539</P>
</DIV>
</BODY>
</HTML>
```

Figure 6.1: DIV Element HTML Code

### CSS Code

Figure 6.2 displays a CSS code that specifies the text styles for the DIV element. The text-align property is set to left, which will align the text towards the left. The text-indent property is set to 2em, which will indent the text with respect to the font size. The text-transform property is set to uppercase, which will display all the letters in uppercase.

```
div
{
text-align:left;
text-indent:2em;
text-transform:uppercase;
}
```

Figure 6.2: CSS Code

The text specified in the DIV element is aligned towards the left and all the letters are displayed in uppercase.

Figure 6.3 shows output.

Figure 6.3: Output

The `text-decoration` and `word-spacing` properties provides different values that allow the user to specify the decoration and word spacing of text in an element.

Table 6.6 lists the values assigned to the `text-decoration` property.

| Value | Description |
|---|---|
| `none` | It is used for displaying normal text without any formatting. |
| `underline` | It is used for displaying a line under the text. |
| `overline` | It is used for displaying a line over the text. |
| `line-through` | It is used for displaying a line through the text. |
| `blink` | It is used for flashing the text. |

**Table 6.6: Values Assigned to the `text-decoration` Property**

Table 6.7 lists the values assigned to the `word-spacing` property.

| Value | Description |
|---|---|
| `normal` | It is used in specifying normal spacing between words and it is the default value. |
| `length` | It is used in specifying fixed space between words. |

**Table 6.7: Values Assigned to the `word-spacing` Property**

Figure 6.4 shows the header and paragraph HTML code.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>Solar System</TITLE>

<LINK rel="stylesheet" type="text/css"
href="Txtproperties.css"/>

</HEAD>

<BODY>

<H3>Nine Planets</H3>

<P>Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus,
Neptune, Pluto</P>

</BODY>
</HTML>
```

**Figure 6.4: Header and Paragraph HTML Code**

Figure 6.5 displays a CSS code that specifies the text properties for the `BODY` and `H3` elements. The `word-spacing` property is set to `2px` for the `BODY` element. This will display each word by leaving a distance of two pixels. The `text-decoration` property is set to `underline` for the `H3` element. This will underline the heading in the Web page.

```
body
{
  word-spacing:2px;
}
h3
{
  text-decoration:underline;
}
```

**Figure 6.5: CSS Code of text-decoration Property**

Figure 6.6 shows an output of heading tag with underline.

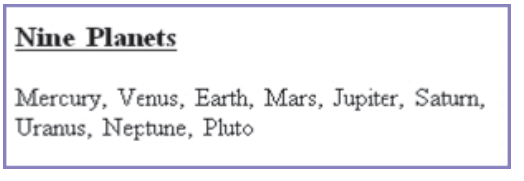



**Figure 6.6: Output of Heading Tag with Underline**

In figure 6.6, the header is underlined and each word in the header and the paragraph is displayed by leaving a distance of two pixels between them.

## *6.3 Inline Span*

The `<span>` tag groups inline-elements in a document. For example, if one word in a sentence need to be bold or colored without using the `<b>` tag then a `<span>` tag is used which can be present within an existing tag.

Code Snippet 1 demonstrates CSS inline style for `<span>` tag.

**Code Snippet 1:**

```
<p>My mother has <span style="color: lightblue">light blue</span> eyes.</p>
Or
<span class="eyesonly">light blue</span>
```

Code Snippet 2 demonstrates CSS external style for `<span>` tag.

**Code Snippet 2:**

```
.eyesonly {font-color: lightblue}
```

The span tag has different attributes; it supports JavaScript event attributes also. Table 6.8 lists different attributes and values used in `<span>` tag.

| Attribute | Value | Description |
|---|---|---|
| class | classname | It is used in specifying the text direction for the content in an element. |
| dir | rtl<br>ltr | It is used in specifying the text direction for the content in an element. |
| id | id | It is used in specifying a unique id for an element. |
| lang | language_code | It is used in specifying a language code for the content in an element. |
| style | style_definition | It is used in specifying an inline style for an element. |
| title | text | It is used in specifying extra information about an element. |
| xml:lang | language_code | It is used in specifying a language code for the content in an element, in XHTML documents. |

**Table 6.8: Different Attributes and Values Used in `<Span>` Tag**

## *6.3.1 Indenting Paragraph*

Indenting is the process of setting off the text from its normal position, either to the left or to the right. In paragraph style, there are three types of indentation:

➔ **First line indent** - The `text-indent` property is used in the CSS for indenting the first line of a paragraph. Code Snippet 3 demonstrates inline style for `<p>` tag and an internal CSS code for first line indent.

**Code Snippet 3**:

```
Inline style
<p style="text-indent: 50px">
Internal CSS
p {text-indent: 50px}
```

Code Snippet 4 demonstrates the use of the `text-indent` property in the HTML file.

**Code Snippet 4:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Font Gallery</title>
<style>
p {text-indent: 150px}
</style>
  </head>
  <body>
<p>The font styles properties allow you to specify the font for the text. They allow you to change the different font attributes of the text such as font, size, and style of the text. The browser must support the font specified by the font properties. Otherwise, it will display the default font, which is dependent on the browser.
</p>
</body>
</html>
```

Figure 6.7 shows the output of text-indent property.

The font styles properties allow you to specify the font for the text. They allow you to change the different font attributes of the text such as font, size, and style of the text. The browser must support the font specified by the font properties. Otherwise, it will display the default font, which is dependent on the browser.

**Figure 6.7: Output of text-indent Property**

➔ **Padding** - The `padding` property is used to add a specified amount of space between the border of an element and its contents.

Code Snippet 5 demonstrates the inline style for `<p>` tag and an internal CSS code for `padding` property.

**Code Snippet 5**:

```
Inline style
<p style="padding: 20px">
Internal CSS
p {padding: 20px}
```

Code Snippet 6 demonstrates the use of the `text-indent` property in the html file.

**Code Snippet 6:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Font Gallery</title>
<style>
p {padding: 20px }
</style>
  </head>
  <body>
<p>The font styles properties allow you to specify the font for the text. They allow you to change the different font attributes of the text such as font, size, and style of the text. The browser must support the font specified by the font properties. Otherwise, it will display the default font, which is dependent on the browser.</p>
</body>
</html>
```

Figure 6.8 shows the padding property.

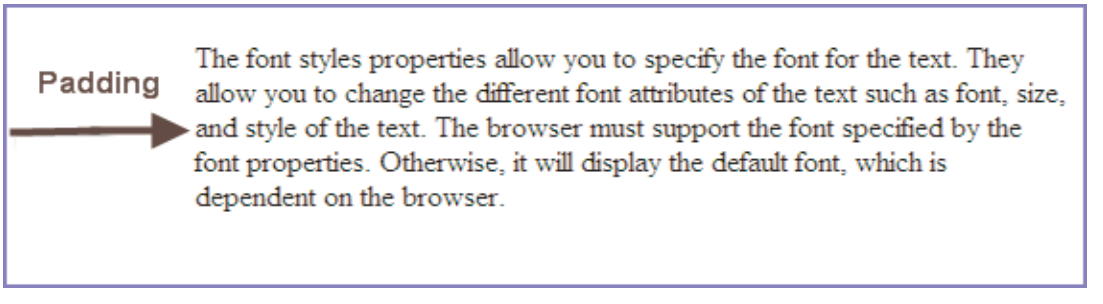



**Figure 6.8: Padding Property**

➔ **Margin** - The `margin` property is used to add a specified amount of white space around an element, on the outside of the element.

Code Snippet 7 demonstrates the inline style for `<p>` tag and an internal CSS code for margin property.

**Code Snippet 7:**

```
Inline style
<p style="margin: 20px">
Internal CSS
p {margin: 20px}
```

Figure 6.9 shows the output of margin property.

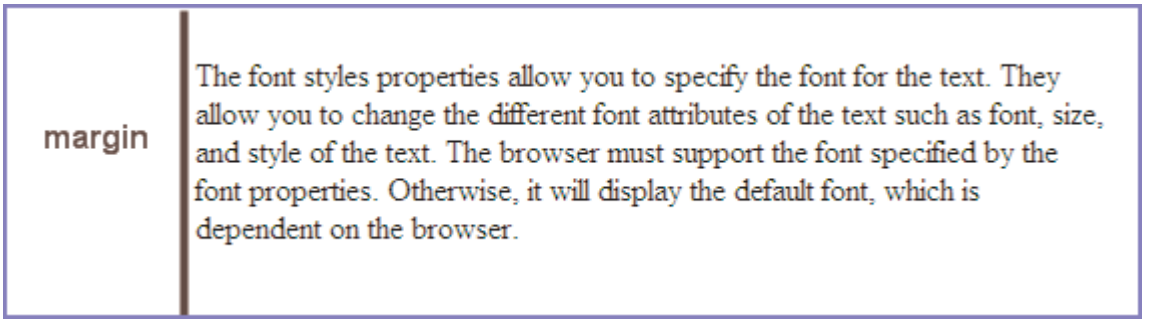



**Figure 6.9: Margin Property**

## *6.4 Border Style*

Borders are rectangular outlines that surround an element. Borders present around text and an image emphasize the content inside the text box. CSS border properties specify the style, color, and width of the border.

Table 6.9 lists the border-style properties.

| border-style properties | Description |
|---|---|
| `border-left-style` | It sets an element's left border. |
| `border-right-style` | It sets an element's right border. |
| `border-top-style` | It sets an element's top border. |
| `border-bottom-style` | It sets an element's bottom border. |

**Table 6.9: `border-style` Properties**

Table 6.10 lists the values of the `border-style` properties.

| Value | Description |
|---|---|
| `dashed` | It is used for specifying a dashed border. |
| `dotted` | It is used for specifying a dotted border. |
| `double` | It is used for specifying two borders. |
| `groove` | It is used for specifying a 3D grooved border. |
| `inset` | It is used for specifying a 3D inset border. |
| `outset` | It is used for specifying a 3D outset border. |
| `ridge` | It is used for specifying a ridged border. |
| `solid` | It is used for specifying a solid border. |

Table 6.10: Values of the `border-style` Properties

Figure 6.10 shows an HTML code.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>MagnaSoftwares</TITLE>
<LINK rel="Stylesheet" type="text/css" href="Styles.css" />
</HEAD>
<BODY>
<DIV id="heading">
<H2>Welcome to MagnaSoftwares</H2>
</DIV>
</BODY>
</HTML>
```

Figure 6.10: HTML Code

Figure 6.11 shows CSS code of border style.

```
#heading
{
background:#FFFFCC;
text-align:center;
border-left-style:ridge;
border-right-style:groove;
border-top-style:dashed;
border-bottom-style:double;
}
```

Figure 6.11: CSS Code of border-style

**Explanation for the code**:

`border-left-style: ridge;`

Applies a ridged border to the left.

`border-right-style: groove;`

Applies a 3D grooved border to the right.

`border-top-style: dashed;`

Applies a dashed border at the top.

`border-bottom-style: double;`

Applies two borders at the bottom.

Figure 6.12 shows the output of `border-style` properties.

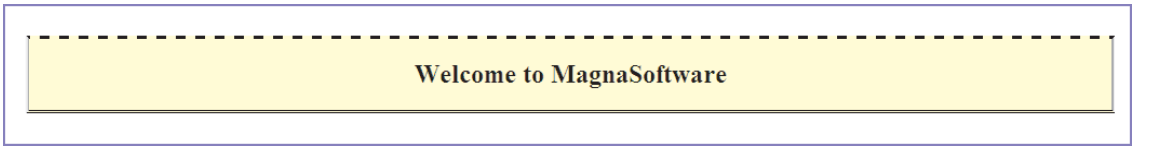



Figure 6.12: Output of `border-style` Properties

### Shorthand Property

To make the code concise CSS allows certain shorthand properties. With the help of these shorthand properties the length of the code is reduced. The shorthand property for setting the border is `border-style`.

Figure 6.13 shows Sample HTML Code.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Corpse - Worlds Largest Flower</title>
  </head>
  <body>
<figure><img class="largest_flower" src="corpse flower.jpg"/></figure>
<h2>World's Largest Flower</h2>
<p>Corpse flower is the world's largest flower.<br/>
Its diameter is about a meter.<br/>
It grows in openings in rainforests on limestone hills of Sumatra, Indonesia.</p>
  </body>
</html>
```

Figure 6.13: Sample HTML Code

Figure 6.14 shows CSS code of shorthand `border-style` properties.

```
.largest_flower
{
 border-style:groove inset outset dashed;
}
```

Figure 6.14: CSS Code of Shorthand `border-style` Properties

**Explanation for the code**:

```
border-style: groove inset outset dashed;
```

Applies a 3D grooved border at the top, 3D inset border at the right, 3D outset border at the bottom, and dashed border at the left.

Figure 6.15 shows output of shorthand `border-style` properties.

**World's Largest Flower**

Corpse flower is the world's largest flower.
Its diameter is about a meter.
It grows in openings in rainforests on limestone hills of Sumatra, Indonesia.

**Figure 6.15: Output of Shorthand `border-style` Properties**

## *6.4.1 Border Color*

The `border-color` property in CSS applies colors to all the four borders. You can also apply four different colors to four borders. There are other border color properties that allow you to individually specify colors of the left, right, top, or bottom border. Table 6.11 lists the different border color properties.

| Property | Description |
|---|---|
| `border-bottom-color` | It is used for specifying the color for the bottom border. |
| `border-left-color` | It is used for specifying the color for the left border. |
| `border-right-color` | It is used for specifying the color for the right border. |
| `border-top-color` | It is used for specifying the color for the top border. |

**Table 6.11: Different `border-color` Properties**

The `border-color` property accepts different color values that determine the different shades of color to be applied to the borders.

The table 6.12 lists the values of the different `border-color` properties.

| Value | Description |
|---|---|
| `color` | It is used in specifying the color to be applied to the border by using either the RGB or hexadecimal value, or the color name itself. |
| `transparent` | It is used for specifying that the border is transparent. |

**Table 6.12: Values of the Different `border-color` Properties**

Figure 6.16 shows an HTML code `DIV` with `border-color` properties.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>HealthCare</TITLE>
<LINK rel="Stylesheet" type="text/css" href="Tips.css" />
</HEAD>
<BODY>
<DIV class="tips">
<H2>Five Essential Health Tips</H2>
<OL>
<LI>Quit Smoking</LI>
<LI>Reduce Stress</LI>
<LI>Protect yourself from Pollution</LI>
<LI>Avoid Excessive Drinking</LI>
<LI>Regular Exercise</LI>
</OL>
</DIV>
</BODY>
</HTML>
```

**Figure 6.16: HTML Code DIV with `border-color` Properties**

Figure 6.17 shows the CSS code of `border-color` properties.

```
.tips
{
background:#FFDDDD;
border-bottom-color:#FF0000;
border-top-color:#FF0000;
border-right-color:#0000FF;
border-left-color:#0000FF;
 }
```

**Figure 6.17: CSS Code of `border-color` Properties**

**Explanation for the code**:

`border-bottom-color: #FF0000;`

Displays the bottom border in red color.

`border-top-color: #FF0000;`

Displays the top border in red color.

`border-right-color: #0000FF;`

Displays the right border in blue color.

`border-left-color: #0000FF;`

Displays the left border in blue color.

**Shorthand Property**

The shorthand property for setting the color of the border is `border-color`.

Figure 6.18 shows an HTML code of a table with `border-color` properties.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>Car Gallery</TITLE>
<LINK rel="Stylesheet" type="text/css" href="Gallery.css" />
</HEAD>
<BODY>
<H2>Car Gallery</H2>
<TABLE border="1">
<TR>
<TD><B>Ferrari</B><BR/><IMG alt="Ferrari" class="carmodel"
src="Ferrari.jpg" /></TD>
<TD><B>Chevrolet</B><BR/><IMG alt="Chevrolet" class="carmodel"
src="Chevrolet.jpg" /></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

**Figure 6.18: HTML Code of a Table with `border-color` Properties**

Figure 6.19 shows CSS code for shorthand `border-color`.

```
body
{
text-align:center;
}
.carmodel
{
border-style:solid;
border-color: Red Blue Green Yellow;
}
```

Figure 6.19: CSS Code of Shorthand `border-color`

**Explanation for the code:**

`border-color: Red Blue Green Yellow;`

Displays the top border in red, right border in blue, bottom border in green, and left border in yellow color.

## *6.4.2 Border Width*

The `border-width` property is a shorthand property that specifies the width for all the four borders. There are other `border-width` properties that allow you to individually specify the left, right, top, or bottom borders. Table 6.13 lists the different `border-width` properties.

| Property | Description |
|---|---|
| `border-bottom-width` | It is used for specifying the width of the bottom border. |
| `border-left-width` | It is used for specifying the width of the left border. |
| `border-right-width` | It is used for specifying the width of the right border. |
| `border-top-width` | It is used for specifying the width of the top border. |

Table 6.13: Different `border-width` Properties

The width of the border can be specified or altered by using the predefined values of the border width properties.

The values of the border width properties specify the way the border will appear. Table 6.14 lists the values of the different `border-width` properties.

| Value | Description |
|---|---|
| `medium` | It is used in specifying a medium border. |
| `length` | It is used in accepting an explicit value that specifies the thickness of border. |
| `thick` | It is used for displaying a thick border. |
| `thin` | It is used in specifying a thin border. |

Table 6.14: Values of the Different `border-width` Properties

Figure 6.20 shows an HTML code for `border-width` properties.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>EasyBank</TITLE>
<LINK rel="Stylesheet" type="text/css" href="banner.css" />
</HEAD>
<BODY>
<DIV class="banner">
<H2>EasyBank - Whole World One Bank</H2>
</DIV>
</BODY>
</HTML>
```

Figure 6.20: HTML Code for `border-width` Properties

Figure 6.21 shows the CSS code of `border-width` properties.

```
.banner
{
text-align:center;
background-color:#C0C0C0;
border-style:solid;
border-right-style:none;
border-left-style:none;
border-top-width: thick;
border-bottom-width: thick;
font-family:fantasy;
}
```

Figure 6.21: CSS Code of `border-width` Properties

**Explanation for the code**:

`border-top-width: thick;`

Displays a thick top border.

`border-bottom-width: thick;`

Displays a thick bottom border.

Figure 6.22 shows the output of `border-width` properties.

Figure 6.22: Output of `border-width` Properties

### Shorthand Property

The shorthand property for setting the border is `border-width`. Figure 6.23 shows an HTML code using the shorthand `border-width` properties.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>UNICEF</TITLE>
<LINK rel="Stylesheet" type="text/css" href="UNICEF.css" />
</HEAD>
<BODY>
<H2>About UNICEF</H2>
<P class="aboutus">
UNICEF is an organization that supports and works for children's
rights, development and protection.
</P>
</BODY>
</HTML>
```

Figure 6.23: HTML Code for Shorthand `border-width` Properties

Figure 6.24 shows the CSS code using the shorthand property, `border-width`.

```
.aboutus
{
text-align:justify;
background-color:#FFFFCC;
border-style:solid;
border-width: thick thin thick thin;
}
```

Figure 6.24: CSS Code Using the Shorthand Property `border-width`

**Explanation for the code**:

```
border-width: thick thin thick thin;
```

Specifies a top and bottom border as thick and right and left border as thin.

Figure 6.25 shows output using shorthand code of `border-width` properties.

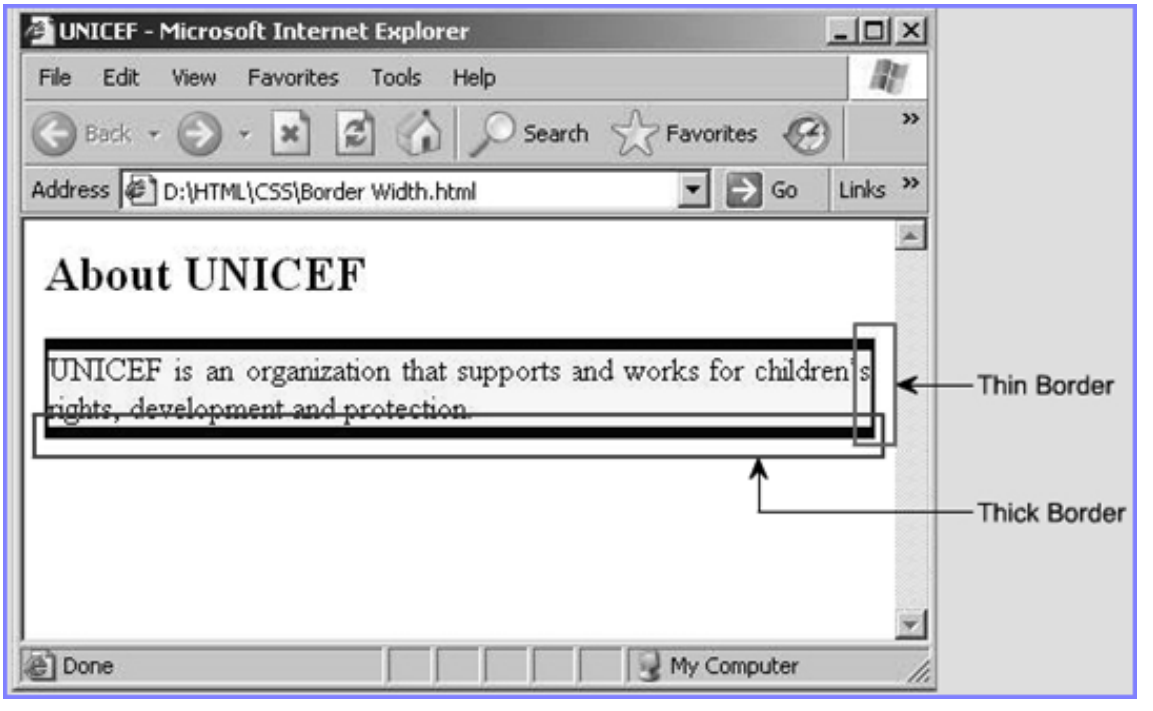



Figure 6.25: Output Using Shorthand Code of `border-width` Properties

## *6.4.3 Shorthand Border*

The `border` shorthand property in CSS specifies all the properties such as style, width, and color for all the four borders. It allows the user to specify the different properties in just one declaration. You can also set these properties individually by using the different shorthand border properties. Table 6.15 lists the different shorthand border properties.

| Property | Description |
|---|---|
| `border-bottom` | It is used in specifying the width, style, and color for the bottom border. |
| `border-left` | It is used in specifying the width, style, and color for the left border. |
| `border-right` | It is used in specifying the width, style, and color for the right border. |
| `border-top` | It is used in specifying the width, style, and color for the top border. |

Table 6.15: Different Shorthand Border Properties

The values of the different border properties determine the type of effect to be applied to the borders.

Figure 6.26 shows an HTML code for shorthand border properties.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Important Note</title>
    <link rel="stylesheet" type="text/css" href="impnote.css"/>
  </head>
  <body>
<h3>Notice:</h3>
<div class="impnote">
<ul style="list-style:square">
<li>Mobiles are not allowed during class hours.</li>
<li>Each student should carry his/her identity card regularly.</li>
</ul>
</div>
  </body>
</html>
```

Figure 6.26: HTML Code for Shorthand Border Properties

Figure 6.27 shows a CSS code using different shorthand border properties.

```
.impnote
{
background-color:#FFFFCC;
border-top:dashed thin #FF0000;
border-bottom:ridge thick #0000FF;
border-right:dotted thin #FF8040;
border-left:inset medium #FF00FF;
}
```

Figure 6.27: CSS Code Using Different Shorthand Border Properties

**Explanation for the code**:

`border-top: dashed thin #FF0000;`

Displays a thin top border with a dashed line in red color.

`border-bottom: ridge thick #0000FF;`

Displays a thick ridged bottom border in blue color.

`border-right: dotted thin #FF8040;`

Displays a thin right border with a dotted line in orange color.

`border-left: inset medium #FF00FF`

Displays a medium 3D inset left border in purple color.

Figure 6.28 shows the output of border properties.

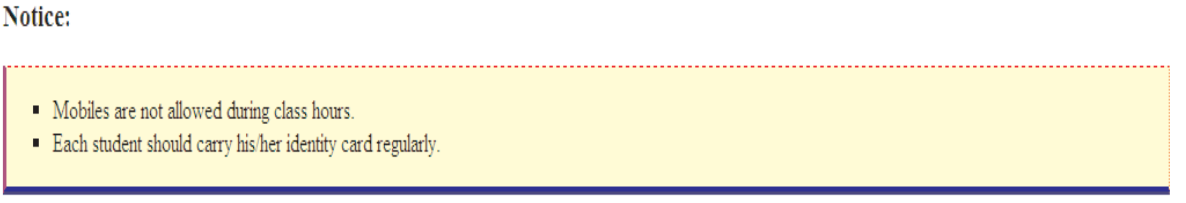



**Figure 6.28: Output of Border Properties**

Figure 6.29 shows an HTML and CSS code of image border property.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Flower Gallery</title>
    <style>
    .flower
    {
        border:solid thin #FF0000;
    }
    </style>
  </head>
  <body>
<h2>Flower</h2>
<table>
<tr>
<td>
<img class="flower" alt="lilac" src="lilac.jpg" height="200px"></img>
</td>
<td valign="top"><h1>Lilac is a species of flowering plants in the olives family. They
are shrubs that ranges from 2 to 10m in height.</h1></td>
</tr>
<tr>
<td>
<img class="flower" alt="sunflower" src="sunflower.jpg" height="200px"
width="150px"></img>
</td>
<td valign="top"><h1>Sunflower is a flowering plants whose stem can grow as high as 3m
tall.</h1></td>
</tr>
</table>
  </body>
</html>
```

**Figure 6.29: HTML and CSS Code of Image Border Property**

**Explanation for the code**:

```
border: solid thin #FF0000;
```

Specifies that all the four borders must be solid in style, thin by width, and red in color.

Figure 6.30 shows the output of Image Border Property.

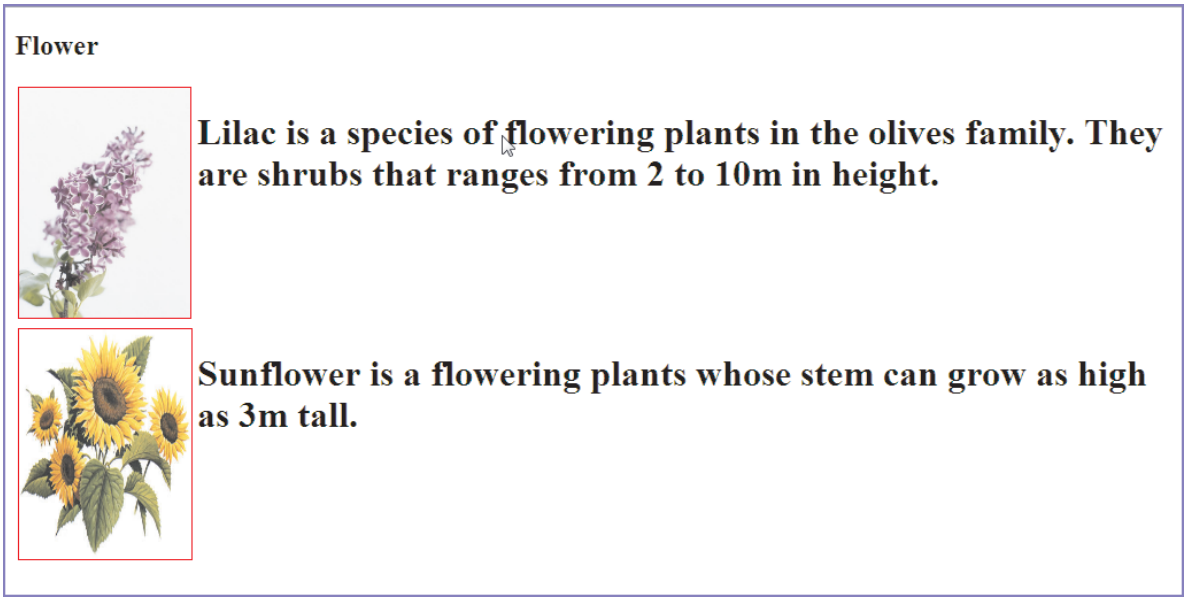



Figure 6.30: Output of Image Border Property

## *6.5 Horizontal Alignment*

In CSS, `text-align` property is used for horizontal alignment of text in an element. This property aligns the inline content of a block.

Table 6.16 lists all values of `text-align` property.

| Value | Description |
|---|---|
| left | Aligns the text to the left. |
| right | Aligns the text to the right. |
| center | Centers the text. |
| justify | Aligns text to both left and right margins by adding space between words (like in newspapers and magazines). |
| inherit | Specifies that the value of the text-align property should be inherited from the parent element. |

Table 6.16: Values of `text-align` Property

The `text-align` property applies only to block-level elements, such as paragraphs. Hence, `text-align` cannot change the alignment of a single word without changing the alignment of the entire line.

For Western languages, which are read from left to right, the default value of `text-align` is `left`. The text aligns on the left margin and has a ragged right margin. Languages such as Hebrew and Arabic has default align to `right` since they are read from right to left.

## *6.6 Vertical Alignment*

In CSS, `line-height` property is used for vertical alignment of text in an element. This property is also a component of the 'font' shorthand property. It can be applied on block-level elements, table cells, table caption, and so on.

Table 6.17 lists the values of `line-height` property.

| Value | Description |
|---|---|
| normal | A normal line height. This is default. |
| number | A number that will be multiplied with the current font size to set the line height. |
| length | A fixed line height in px, pt, cm, and so on. |
| % | A line height in percent of the current font size. |
| inherit | Specifies that the value of the line-height property should be inherited from the parent element. |

**Table 6.17: Values of `line-height` Property**

## *6.7 Check Your Progress*

1. The _____and _____ properties provide different values that allow the user to specify the decoration and word spacing of text in an element.

| | | | |
|---|---|---|---|
| (A) | `word-spacing` | (C) | `line-height` |
| (B) | `text-decoration` | (D) | `text-align` |

2. The `<span>` tag _____ inline-elements in a document.

| | | | |
|---|---|---|---|
| (A) | Right aligns | (C) | Groups |
| (B) | Left aligns | (D) | Deletes |

3. Which border property specifies the different properties in just one declaration?

| | | | |
|---|---|---|---|
| (A) | border-bottom-color | (C) | border-right-width |
| (B) | border-bottom | (D) | border-left |

4. Match the value against the description:

| Value | | Description | |
|---|---|---|---|
| (A) | `medium` | (1) | Specifies a thin border. |
| (B) | `length` | (2) | Specifies a thick border. |
| (C) | `thick` | (3) | Accepts an explicit value that specifies the thickness of border. |
| (D) | `thin` | (4) | Specifies a medium border. This is the default value. |

| | | | |
|---|---|---|---|
| (A) | a-2, b-1, c-3, d-4 | (C) | a-4, b-1, c-2, d-3 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-4, b-3, c-2, d-1 |

5. Match the following:

| Property | | Description | |
|---|---|---|---|
| (A) | `border-bottom-color` | (1) | Specifies the color for the right border. |
| (B) | `border-left-color` | (2) | Specifies the color for the top border. |
| (C) | `border-right-color` | (3) | Specifies the color for the bottom border. |
| (D) | `border-top-color` | (4) | Specifies the color for the left border. |

| | | | |
|---|---|---|---|
| (A) | a-3, b-4, c-1, d-2 | (C) | a-4, b-3, c-2, d-1 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-2, b-1, c-4, d-3 |

6. What is the default text alignment for Hebrew and Arabic languages?

| | | | |
|---|---|---|---|
| (A) | Left | (C) | Right |
| (B) | Center | (D) | Justify |

## *6.7.1 Answers*

| | |
|---|---|
| 1. | A, B |
| 2. | C |
| 3. | B, D |
| 4. | D |
| 5. | A |
| 6. | C |

# Summary

- The text styles specify and control the appearance of the text in a Web page.
- Indenting is the process of offsetting text from its normal position, either to the left or to the right.
- CSS border properties specify the style, color, and width of the border.
- The border-color property accepts different color values that determine the different shades of color to be applied to the borders.
- The values of the different border properties determine the type of effect to be applied to the borders.
- In CSS, the text-align property is used for horizontal alignment of text in an element.
- In CSS, the line-height property is used for vertical alignment of text in an element.

# *Try it Yourself*

1. Mathew wants to develop a Web page on his biblical findings. For writing major quotes of the bible verses he wants blue bordered text box with center alignment of the content and for its explanation he wants text box with red colored text that would be vertically center aligned. Use CSS code to create this Web page.

# Session - 7
# Displaying Graphics and CSS3 Animation

Welcome to the Session, **Displaying Graphics and CSS3 Animation**.

This session explains formatting of graphics, insertion, sizing, and padding of graphics in Web pages. This session also explains CSS3 animation and how CSS3 can be used on Mobiles.

In this Session, you will learn to:

- ➔ Explain graphic formatting in Web pages
- ➔ Explain graphic insertion, sizing, and padding
- ➔ Explain CSS3 Animation
- ➔ Describe the use of CSS3 on mobile devices

## *7.1 Introduction*

After the release of HTML5 and CSS3 in the market, most of the Web designers are developing graphics based Web page. CSS3 has allowed the designers to style their Web pages graphically with ease. Currently, HTML5 applications provide amazing experiences with the use of new CSS3 animations. The introduction of mobile applications has allowed the users to expand their Web usage to mobile devices. CSS3 has introduced new features specifically for mobile devices.

## *7.2 Graphic Format*

There are many graphic formats available; the most commonly used are Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF), and Portable Network Graphics (PNG).

The difference between each graphic format depends on the following characteristics:

➔ **Color Depth** – It is defined by the number of distinct colors that are represented by a hardware or software. Color depth is defined by the number of bits per pixel (bpp) and it is also called as bit depth. Higher the color depth indicates higher range of colors used.

➔ **Compression/file size** – Graphic files are large, so images are compressed using various techniques. Compression stores the original images in a reduced number of bytes using an algorithm. This image can be expanded back to the original size using a decompression algorithm. In some compression formats, images with less complexity results in smaller compressed file sizes.

The two types of image file compression algorithms used are as follows:

- **Lossless compression** – In this algorithm, file size is reduced but preserves a copy of the original uncompressed image. Lossless compression avoids accumulating stages of re-compression when editing images.
- **Lossy compression** – In this algorithm, a representation of the original uncompressed image is preserved. The image appears to be a copy of the original image but in actuality it is not a copy. Lossy compression achieves smaller file sizes when compared with lossless compression. Generally, lossy compression algorithms allow variable compression that comprises on image quality for file size.

➔ **Animation** – Some graphic format consists of a series of frames that are played one after the other giving an impression of animation. Animated graphics are typically used on a Web page to attract visitor's attention.

Figure 7.1 shows an animated graphic.

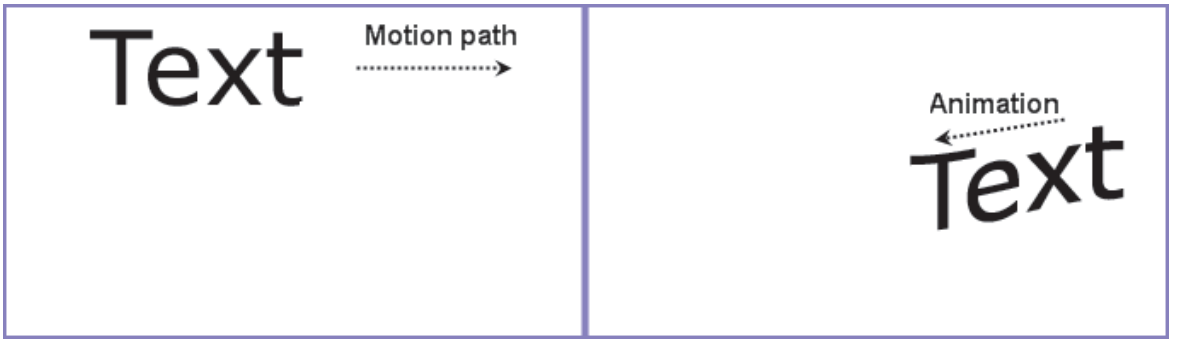



**Figure 7.1: Animated Graphic**

➔ **Transparency** – It is very common on the Web to display an image on a Web page that appears directly against the background color of the page. The background color of the Web page shows through the transparent portion of the image. In a transparent image, one and only one color can be hidden. If the color chosen to make transparent is same as the background of the inserted image, then an irregularly shaped image appears to float on the page. Figure 7.2 shows a transparent image.

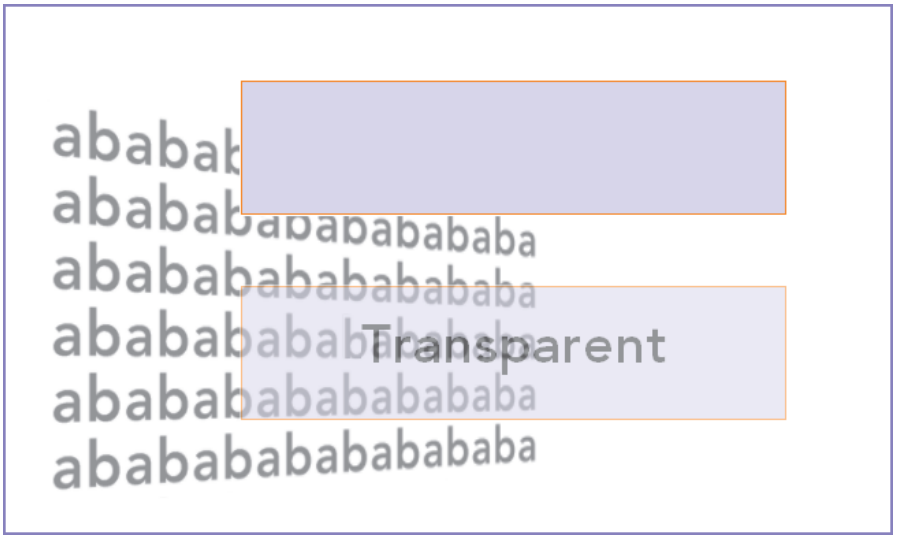



**Figure 7.2: Transparent Image**

## *7.2.1 Graphic Format for the Web*

For Web pages, use of JPEG and PNG graphics are recommended as it provides maximum compatibility with all the devices that might be accessing the Web page. For photos, use of JPEG graphic format and for screen-shots and drawings use of PNG graphic format is recommended. Both these formats compress the picture information to reduce the download time and increase the downloading speed.

- **JPEG** - It uses a lossy compression which means that the image quality is lost in the process of compressing the image. It is recommended that for continuous tone pictures such as photos JPEG should be used. Most JPEG editor allows the user to specify the amount of detail that the user is prepared to lose. If the quality is reduced, then the loss is visible; JPEG is about half the size of PNG.

- **PNG** - It uses lossless compression, which means there is no loss of any image detail. PNG was designed for transferring images on the Internet and not for professional-quality print graphics; therefore it does not support non-RGB color spaces such as CMYK. It supports high color and partial transparency using alpha channels.

**Note** - An alpha channel is a special type of channel used in graphics software for saving selections.

- **GIF** - It uses a lossless compression which means that there is no loss in quality when the image is compressed. The uncompressed image stores its information in a linear fashion. Each line of pixels is read from left to right. An interlaced GIF file stores the lines of the image in a different order. Animated graphics are stored in gif format.

Compatibility and appearance are the keywords on the Web. The inserted images must be visible and undistorted when appearing on any recipient's device. The Web designer can make assumptions that the Web site will open in a computer which will have minimum resolution of `800x600` pixel display capability. If a mobile based Web page needs to be created then the specifications will change.

## *7.2.2 Graphic Insertion*

The `IMG` element is an empty element, which allows the user to insert an image in a Web page. It allows insertion of images and diagrams. The commonly used graphic formats that are supported are namely, GIF, JPEG, BITMAP (BMP), and PNG. The `<img>` tag reserves a space for the image and does not insert the image in the HTML page. It creates a link between the image and the HTML page.

Table 7.1 lists the commonly used attributes of the `IMG` element.

| Attributes | Description |
|---|---|
| `src` | Specifies the path of an image that is to be displayed. |
| `height` | Specifies the height of an image. |
| `width` | Specifies the width of an image. |

**Table 7.1: Commonly Used Attributes of the IMG Element**

Code Snippet 1 demonstrates how to display an image in a Web page using the IMG element.

**Code Snippet 1**:

```
<body>
<img src="UNO.jpg" width="225" height="151"/>
</body>
```

The code uses the `src` attribute of the `IMG` element to insert a `JPEG` image. The attribute specifies the name of the image and also indicates that the image is present in the same folder where the HTML file is saved. The `width` and `height` of the image is set to `225` and `151` pixels respectively by using the width and height attribute. A pixel refers to the smallest dot on the monitor screen.

An image can also be stored in a subfolder of the folder containing the HTML file. In such cases, a reference to the image is made by using the sub folder name as shown in Code Snippet 2.

**Code Snippet 2**:

```
<body>
<img src="image_folder/UNO.jpg" width="225" height="151"/>
</body>
```

To align the image the `float` style attribute can be used to specify the inline style for the element. This will force the image to be aligned to the left or right side of the screen and wrap the surrounding text around the image. Code Snippet 3 demonstrates the use of the float style.

**Code Snippet 3**:

```
<body>
<img src="image_folder/UNO.jpg" style="float:left"/>
</body>
```

Table 7.2 lists the values of float property in the `<img>` tag.

| Value | Description |
|---|---|
| left | The element floats to the left. |
| right | The element floats to the right. |
| none | The element does not float and is the default value. |
| inherit | The element specifies that the value of the float property should be inherited from the parent element. |

**Table 7.2: Values of Float Property in <img> Tag**

HTML5 introduced a new <figure> tag. The <figure> tag acts as a container containing the <img> tag. In other words, it is not a replacement for <img> tag, but acts as a container into which the <img> tag is placed. The <figure> tag specifies self-contained content, such as illustrations, diagrams, photos, code listings, and so on.

While the content of the <figure> element is related to the main flow, its position is independent of the main flow, and if removed it does not affect the flow of the document.

Code Snippet 4 demonstrates the use of <figure> tag.

**Code Snippet 4**:

```
<figure>
  <img src="logo.gif" style="float: left">
  width="304" height="228" />
</figure>
```

The main advantage of using <figure> tag is that it allows the user to use the <figcaption> tag along with it. The <figcaption> tag allows the user to add a caption to the image. The caption always appears along with the image even if the image floats in Web site layout.

Code Snippet 5 demonstrates the use of <figcaption> tag.

**Code Snippet 5**:

```
<figure>
<img src="logo.gif">
<figcaption>This diagram shows the logo of a product.</figcaption>
</figure>
```

The <figure> tag can also assign styles and other attributes to the <figure> element using an external or internal style sheet. A single caption to a group of images can be added using the <figure> tag.

Code Snippet 6 demonstrates how to assign a single caption to a group of images.

**Code Snippet 6**:

```
<figure>
<img src="flower1.jpg">
<img src="flower2.jpg">
<img src="flower3.jpg">
<figcaption>The different types of flowers</figcaption>
</figure>
```

Figure 7.3 shows output of a single caption to a group of images.

Figure 7.3: A Single Caption to a Group of Images

## *7.2.3 CSS Image Sizing and Padding*

Size of an image is specified in pixels. The height and width property sets the height and width of the image. One can specify the width and the height will be resized or vice versa.

**Note** - The height and width property does not include padding, borders, or margins.

Code Snippet 7 demonstrates CSS code for setting the image height and width property.

**Code Snippet 7**:

```
p.ex
{
height:100px;
width:100px;
}
```

Table 7.3 lists different CSS properties and values of images.

| Property | Description | Values |
|---|---|---|
| height | Sets the height of an element | • Auto<br>• Length<br>• %<br>• inherit |

| Property | Description | Values |
|---|---|---|
| max-height | Sets the maximum height of an element | • none<br>• length<br>• %<br>• inherit |
| max-width | Sets the maximum width of an element | • none<br>• length<br>• %<br>• inherit |
| min-height | Sets the minimum height of an element | • length<br>• %<br>• inherit |
| min-width | Sets the minimum width of an element | • length<br>• %<br>• inherit |
| width | Sets the width of an element | • auto<br>• length<br>• %<br>• inherit |

**Table 7.3: Different CSS Properties and Values of Images**

Table 7.4 list the various values used with height and width properties.

| Value | Description |
|---|---|
| auto | The browser calculates the height and is the default value |
| length | Defines the length in pixels (px) |
| % | Defines the height of the containing block in percent format |
| inherit | Specifies that the value of the property should be inherited from the parent element |

**Table 7.4: Various Values Used in Height and Width Properties**

## Padding

The CSS padding property is used to specify the space between the element border and the element content. It is used to separate them from the surrounding element. The background color of the element affects the padding property. Using separate properties such as top, right, bottom, and left, different padding values can be specified and the padding can be changed separately.

Table 7.5 list the various values used in padding property.

| Value | Description |
|---|---|
| length | This property specifies a fixed value for padding in pixels, pt, em, and so on |
| % | This property specifies a value for padding in % of the containing element |

**Table 7.5: Various Values used in Padding Property**

Code Snippet 8 demonstrates the CSS code used for specifying different padding values for different sides.

**Code Snippet 8**:

```
padding-top:10px;
padding-bottom:10px;
padding-right:15px;
padding-left:15px;
```

In the code, the value for padding was set for all the sides.

Instead of using different padding for different sides, users can use a shorthand property. A shorthand property is one where all the padding properties for the different sides are specified in one property. This will result in a shortened code.

The shorthand property for all the padding properties is `padding`. The property can be used to specify one to four values for each of the side. Code Snippet 9 demonstrates the use of the shorthand property for padding.

**Code Snippet 9**:

```
padding:25px 50px 75px 100px;
```

Where top padding is `25px`, right padding is `50px`, bottom padding is `75px`, and left padding is `100px`.

Table 7.6 lists all CSS padding properties.

| Property | Description |
|---|---|
| padding | The browser calculates the height and is the default value |
| padding-bottom | Defines the length in pixels (px) |
| padding-left | Defines the height of the containing block in percent format |
| padding-right | Specifies that the value of the property should be inherited from the parent element |
| padding-top | Sets the top padding of an element |

**Table 7.6: CSS Padding Properties**

## *7.3 Thumbnail Graphics*

The speed of loading a page of a Web site is reduced if high-resolution graphics are used. High-resolution graphics are required to improve the effectiveness of the site and cannot be avoided. Hence, to avoid this issue, thumbnails are used.

A thumbnail is a small image, or a part of a larger image. Clicking the thumbnail image will link to the larger original image, which can be viewed and downloaded. Even a hover effect can be given through CSS and JavaScript.

Code Snippet 10 demonstrates an HTML code for inclusion of a thumbnail image.

**Code Snippet 10:**

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Thumbnail</title>
  <style>
/* general */
     body {
         margin:0;
         padding:40px 80px;
         background:#fff;
         font:70% Arial, Helvetica, sans-serif;
         color:#555;
         line-height:100%;
```

```
}
h1, h2{
      font-size:180%;
      font-weight:normal;
      color:#555;
}
p{
      margin:1em 0;
}
p.text{
      width:500px;
}
a{
      color:#f20;
      text-decoration:none;
}
a:hover{
      color:#999;
        }
img{
      border:none;
}
             /* // general */
             /* thumbnail list */
            ul#thumbs, ul#thumbs li{
            margin:0;
            padding:0;
      list-style:none;
```

```
 }
ul#thumbs li{
      float:left;
      margin-right:0px;
      border:1px solid #999;
      padding:2px;
}
ul#thumbs a{
      display:block;
      float:left;
      width:125px;
      height:135px;
      line-height:50px;
      overflow:hidden;
      position:relative;
      z-index:1;
            }
ul#thumbs a img{
      float:left;
      position:absolute;
      top:0px;
      left:0px;
}
/* mouse over */
      ul#thumbs a:hover{
            overflow:visible;
            z-index:1000;
                  border:none;
            }
```

```
            ul#thumbs a:hover img{
                border:1px solid #999;
                background:#fff;
                padding:2px;
                            }
     /* // mouse over */
     /* clearing floats */
          ul#thumbs:after, li#thumbs:after{
                content:".";
                display:block;
                height:0;
                clear:both;
                visibility:hidden;
                }
          ul#thumbs, li#thumbs{
                display:block;
                }
          ul#thumbs, li#thumbs{
                min-height:1%;
                }
          * html ul#thumbs, * html li#thumbs{
                height:1%;
                }
/* // clearing floats */
/* // thumbnail list */
</style>
  </head>
<body>
  <h2>Thumbnail</h2>
```

```
      <ul id="thumbs">
      <li><a href="HTML5.png" target="_blank"><img src="HTML5_small.jpg"/></
a></li>
      </ul>
  </body>
</html>
```

Figure 7.4 shows output of thumbnail with hover effect.

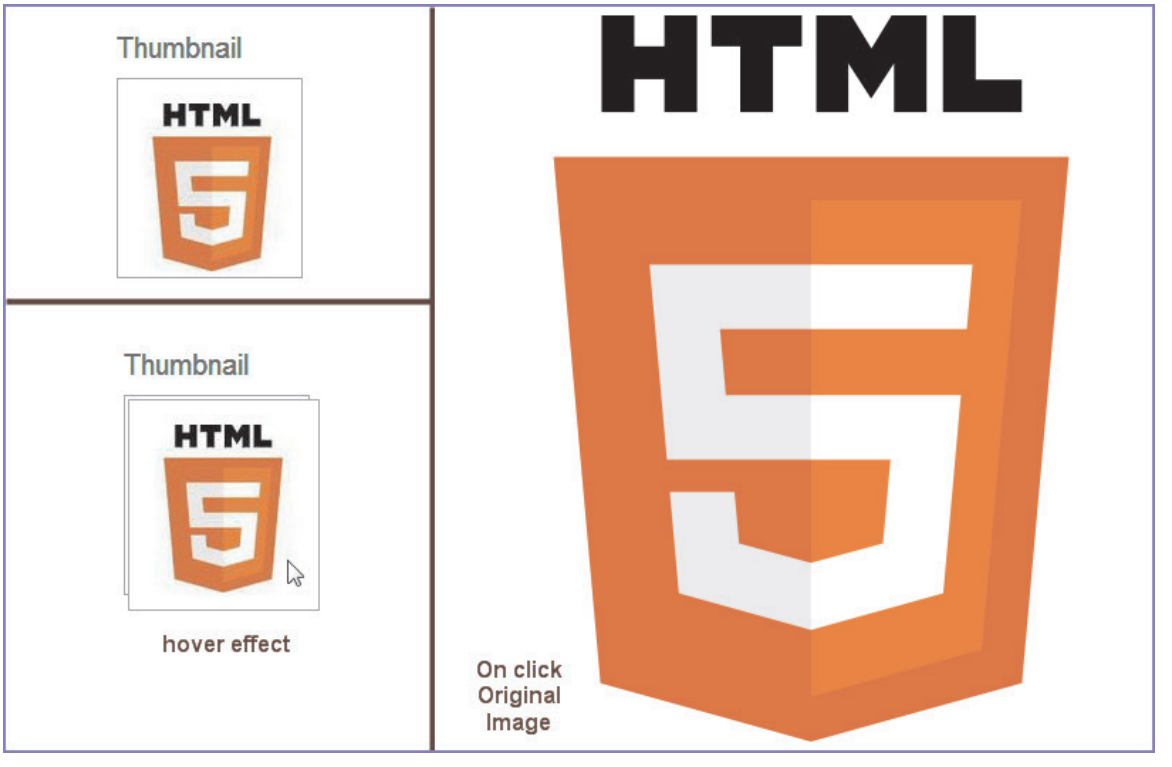



**Figure 7.4: Output of Thumbnail with Hover Effect**

## *7.4 Working with CSS3 Transitions*

Interactivity is one of the important aspects of animation. Earlier, a combination of HTML, CSS, and JavaScript were used to animate objects on the Web. In 2007, Apple introduced the CSS transition, which later became a proprietary feature of Safari called CSS Animation. Representatives from Apple and Mozilla began adding the CSS transitions module to the CSS Level 3 specification, closely modeled on what Apple had already added to Webkit and moz.

All the browsers do not support CSS3 transitions. Browsers that support CSS3 Transitions are as follows:

- Apple Safari 3.1 and later which requires the prefix –webkit-
- Google Chrome which requires the prefix –webkit-
- Mozilla Firefox 3.7 alpha and later which requires the prefix –moz-
- Opera 10.5x and later which requires the prefix –o-

At the moment Internet Explorer 9 does not support CSS3 Transitions.

For performing CSS transitions the two required specifications are as follows:

- The CSS property that needs the effect
- The duration of the effect

Code Snippet 11 demonstrates the use of transition effect on the width property for 3 seconds.

**Code Snippet 11:**

```
div
{
transition: width 3s;
-moz-transition: width 3s; /* Firefox 4 */
-webkit-transition: width 3s; /* Safari and Chrome */
-o-transition: width 3s; /* Opera */
}
```

The effect will start when the specified CSS property changes value. The CSS property changes its value typically when a user moves a mouse over an element. Thus, the user can set the hover for `<div>` elements. Code Snippet 12 demonstrates the same.

**Code Snippet 12**:

```
div:hover
{
width:200px;
}
```

Table 7.7 lists all the transition properties.

| Property | Description |
|---|---|
| transition | Is a shorthand property and is used for setting the four transition properties into a single property |
| transition-property | Is used for specifying the name of the CSS property for which the transition value is set |
| transition-duration | Is used for defining the duration of the transition. Default value is 0 |
| transition-timing-function | Is used for describing how the speed during a transition will be calculated. Default value is “ease” |
| transition-delay | Is used for defining the start of the transition. Default value is 0 |

**Table 7.7: Transition Properties**

Code Snippet 13 demonstrates an HTML and CSS code using all transition properties.

**Code Snippet 13**:

```
<!DOCTYPE html>
<html>
<head>
<style type=”text/css”>
div
{
width:100px;
height:100px;
background:#000000;
transition-property:width;
transition-duration:2s;
transition-timing-function:linear;
transition-delay:1s;
/* Firefox 4 */
-moz-transition-property:width;
```

```
-moz-transition-duration:2s;
-moz-transition-timing-function:linear;
-moz-transition-delay:1s;
/* Safari and Chrome */
-webkit-transition-property:width;
-webkit-transition-duration:2s;
-webkit-transition-timing-function:linear;
-webkit-transition-delay:1s;
}
div:hover
{
width:500px;
}
</style>
</head>
<body>
<p><b>Note:</b> The example</p>

<div></div>

<p>Hover over the div element above, to see the transition effect.</p>
<p><b>Note:</b> The transition effect will wait 1 seconds before starting.</
p>

</body>
</html>
```

Figure 7.5 shows output of all transition properties.

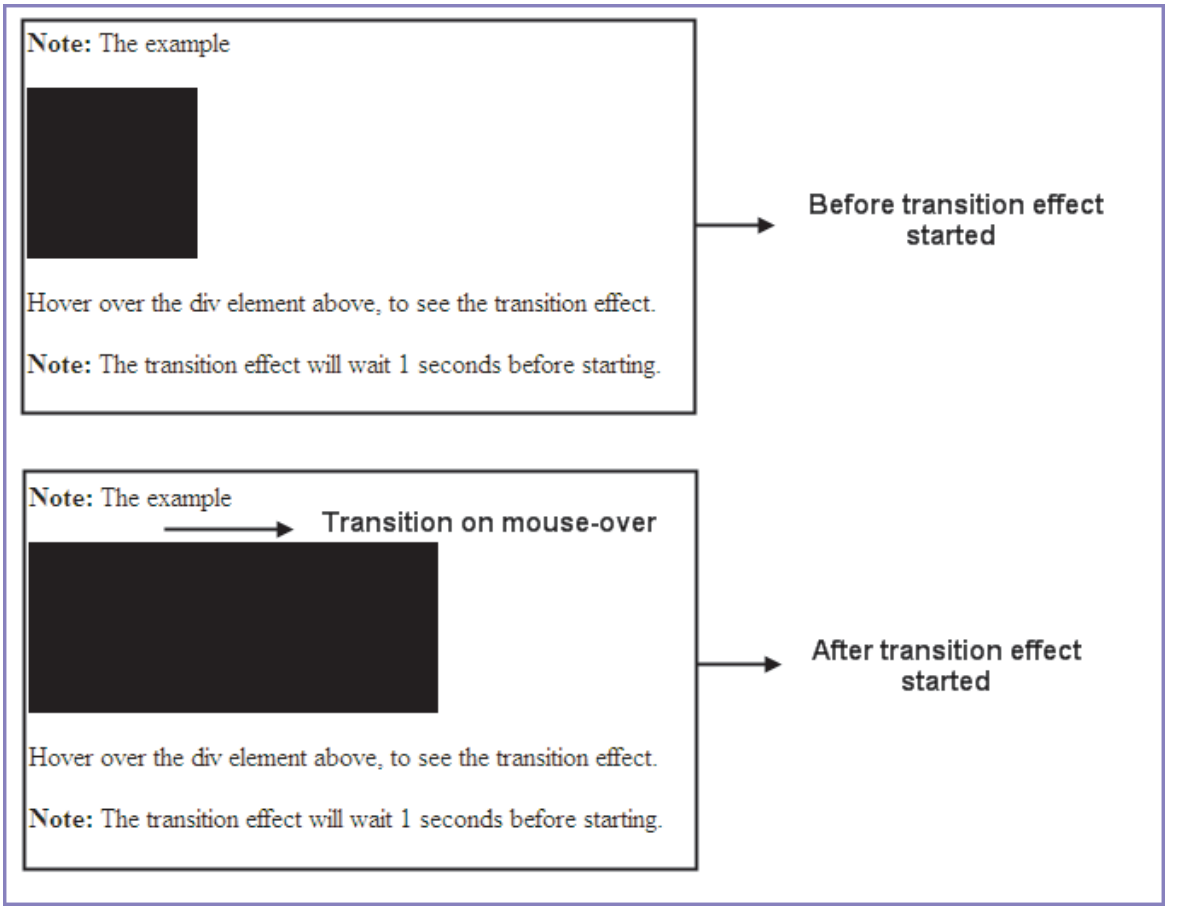



Figure 7.5: Output of all Transition Properties

## *7.5 CSS3 Animation*

CSS3 animations can animate transitions of one CSS style configuration to another. The two components of animation are as follows:

- An animation style describing the animation.
- A keyframes set that specifies the start and end states of the animation's CSS style and possible intermediate waypoints along the way.

The three advantages to CSS3 animations over script-based animation techniques are as follows:

1. Easy to use and anybody can create them without the knowledge of JavaScript.
2. Executes well even under reasonable system load. As simple animations perform poorly in JavaScript, the rendering engine uses the frame-skipping techniques to allow smooth flow of animation.
3. Allows the browser to control the animation sequence, optimize performance and efficiency by reducing the update frequency of animations executing in tabs that aren't currently visible.

## *7.5.1 Configuring the Animation*

A CSS animation sequence can be created by styling the element with the `animation` property. This property can be used to configure the timing, duration, and sequence of the animation. `@keyframes` rule define the appearance of the animation. The keyframe is used to describe the rendering of the element in the animation sequence.

Table 7.8 lists the `@keyframes` rule and all the animation properties.

| Property | Description |
|---|---|
| `@keyframes` | Is used for specifying the animation |
| `animation` | Is a shorthand property representing all the animation properties, except the animation-play-state property |
| `animation-name` | Is used for specifying the name of the @keyframes animation |
| `animation-duration` | Is used for specifying the duration of an animation cycle in seconds or milliseconds. Default value is 0 |
| `animation-timing-function` | Is used for describing the progress of animation over one cycle of its duration. Default value is “ease” |
| `animation-delay` | Is used for specifying the start value of animation. Default value is 0 |
| `animation-iteration-count` | Is used for specifying the number of times an animation is played. Default value is 1 |
| `animation-direction` | Is used for specifying whether or not the animation should play in reverse on alternate cycles. Default value is “normal” |
| `animation-play-state` | Is used for specifying the state of the animation, that is whether it is running or paused. Default value is “running” |

**Table 7.8: `@keyframes` Rule and all the Animation Properties**

The syntax for `@keyframes` is as follows:

**Syntax**:

```
@keyframes myfirst
{
from {background: red;}
to {background: yellow;}
}
@-moz-keyframes myfirst /* Firefox */
{
```

```
from {background: red;}
to {background: yellow;}
}
@-webkit-keyframes myfirst /* Safari and Chrome */
{
from {background: red;}
to {background: yellow;}
}
```

The animation created using `@keyframes` must be bound with the selector for effective execution. For this, specify the name of the animation and the duration of the animation to the selector.

Code Snippet 14 demonstrates HTML and CSS code of `@keyframes` rule and all the animation properties.

**Code Snippet 14**:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
div
{
width:200px;
height:200px;
background:red;
position:relative;
border-radius:100px;
animation-name:myfirst;
animation-duration:4s;
animation-timing-function:linear;
```

```
animation-delay:1s;
animation-iteration-count:infinite;
animation-direction:alternate;
animation-play-state:running;
/* Firefox: */
-moz-border-radius:100px;
-moz-animation-name:myfirst;
-moz-animation-duration:4s;
-moz-animation-timing-function:linear;
-moz-animation-delay:1s;
-moz-animation-iteration-count:infinite;
-moz-animation-direction:alternate;
-moz-animation-play-state:running;
/* Safari and Chrome: */
-webkit-border-radius:100px;
-webkit-animation-name:myfirst;
-webkit-animation-duration:4s;
-webkit-animation-timing-function:linear;
-webkit-animation-delay:1s;
-webkit-animation-iteration-count:infinite;
-webkit-animation-direction:alternate;
-webkit-animation-play-state:running;
}
@keyframes myfirst
{
0% {background:red; left:0px; top:0px;}
25% {background:yellow; left:300px; top:0px;}
50% {background:blue; left:300px; top:300px;}
75% {background:green; left:0px; top:300px;}
```

```
100% {background:red; left:0px; top:0px;}
}
@-moz-keyframes myfirst /* Firefox */
{
0% {background:red; left:0px; top:0px;}
25% {background:yellow; left:300px; top:0px;}
50% {background:blue; left:300px; top:300px;}
75% {background:green; left:0px; top:300px;}
100% {background:red; left:0px; top:0px;}
}
@-webkit-keyframes myfirst /* Safari and Chrome */
{
0% {background:red; left:0px; top:0px;}
25% {background:yellow; left:200px; top:0px;}
50% {background:blue; left:200px; top:200px;}
75% {background:green; left:0px; top:200px;}
100% {background:red; left:0px; top:0px;}
}
</style>
</head>
<body>
<p><b>Note:</b> Animation</p>
<div></div>
</body>
</html>
```

Figure 7.6 shows the output of `@keyframes` rule and all the animation properties.

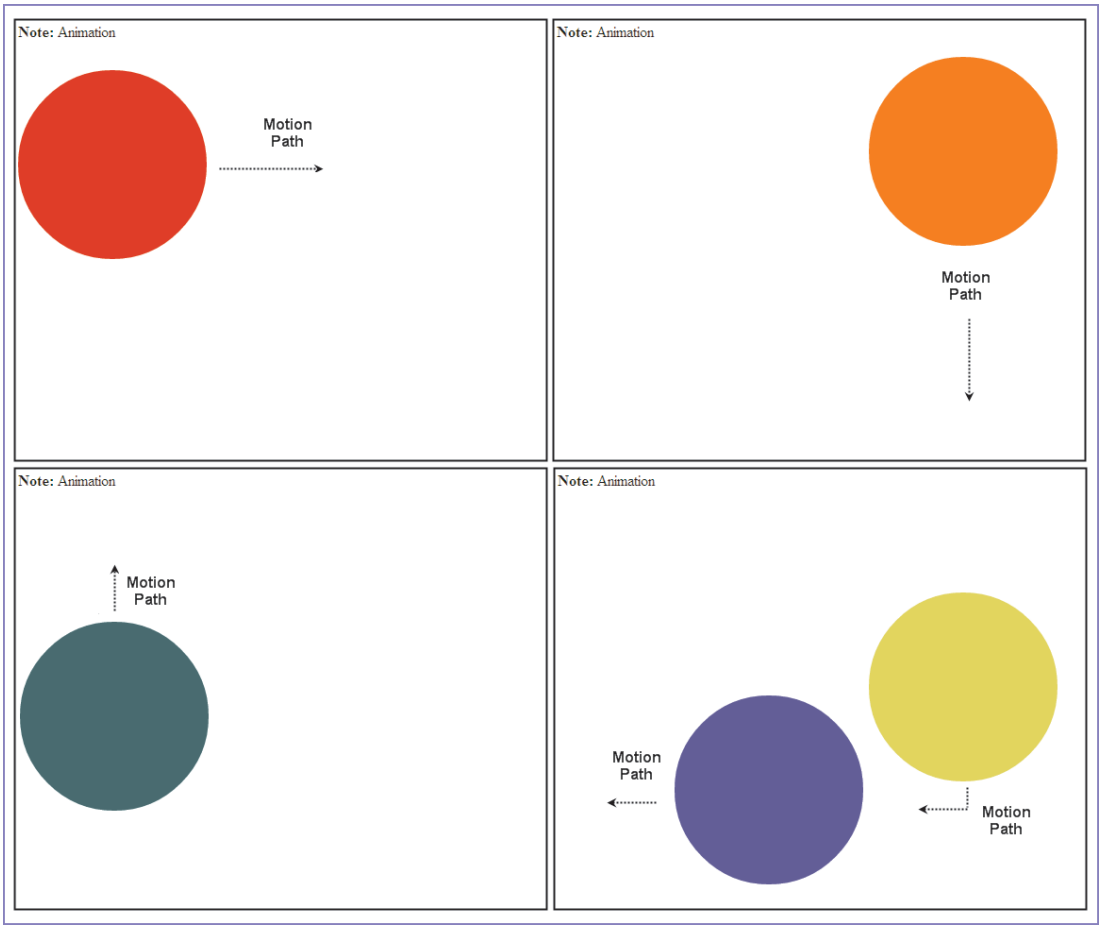



**Figure 7.6: Output of @keyframes Rule and all the Animation Properties**

## *7.6 Using CSS3 on Mobile Devices*

There are different ways to provide Web pages for mobile devices. The user can make use of style sheet for the handheld devices (all mobile browsers do not recognize it). IPhone's Safari and Opera's Mini browsers support a new feature of CSS3 called media queries. These queries allow the user to specify a conditional expression for media type. Code Snippet 15 shows the use of a conditional expression for displaying a link element where the maximum screen width for mobile devices is 480 pixels.

**Code Snippet 15**:

```
<link rel="stylesheet" href="styles/mobile.css" media="only screen and (max-
device-width: 480px)"/>
```

Next, the user can also specify another link element for screen media with a minimum screen width of 481 pixels. In other words, the style sheet for this element can be used for standard computer screens.

Separate Web sites must be developed for mobile devices. The home page of the main site should provide a link that connects to the mobile Web site. This technique identifies the mobile device of the user and renders the mobile Web site automatically in the best view possible. Most mobile Web sites are created to precede the domain name of the main site with m for example **m.aptech-education.com**

To detect a mobile device, a Web site can use JavaScript on the client, a scripting language on the server, or Wireless Universal Resource File (WURFL) on the server.

The five ways to provide Web pages for mobile devices are as follows:

- Define a style sheet for mobile devices
- Include a link to a mobile version of the Web site
- Use JavaScript to detect mobile devices and redirect
- Use a server-side scripting language to detect and redirect
- Use the WURFL to detect mobile devices

**Note** - WURFL is a community effort focused on mobile device detection. WURFL is a set of proprietary APIs and an XML configuration file which contains information about device capabilities and features for a variety of mobile devices.

## *7.7 Coding for Optimum Browser Compatibility*

Web browser compatibility measures are undertaken by Web developers who are committed to producing Web products that provide predictability and consistency across the preferable Web browsers of the targeted end users.

Cross browser compatibility means a Web site that is attuned and reliable in looks, layout, color, functionality, interactivity, and proportion across all existing Web browsers, regardless of the browsers' insignificance or popularity differences from version to version. Multi-browser compatibility is constant and it is functionally rendered across the most commonly used browsers in a client's target market. HTML5 uses different standards and is supported by various browsers. These browsers provide different version of support.

Rendering engines are a set of tools that are used in most browsers that supports different HTML features. Some of the rendering engines of different browsers are as follows:

- **Gecko** - The Gecko engine is the main engine of Mozilla Firefox, and a number of related browsers. It has support for various HTML5 features. Although, Firefox is an eminent and highly appreciated browser in the Web development community, it does not yet have the full support for HTML5.

- **Trident** - The Trident engine is used by different versions of Internet Explorer (IE). Currently, HTML5 is not majorly supported by the Trident engine. IE9 was anticipated to support HTML5 completely, but it has failed to support some features such as the advanced form element support and geolocation.

- **WebKit** - The WebKit engines is supported mainly for the Safari browser used in Apple Macs, iPhones, iPads, and other Apple products. This engine is based on the open source KHTML project. Webkit is also the base for Android based browsers such as Google Chrome.

  WebKit has evolved to become the standard rendering engine for mobile platforms. WebKit has the maximum support for most of the HTML5 elements, although it still does not support everything.

- **Presto** - Presto is the engine used in the Opera browsers. Opera browsers are considered to be a technically superior browser, but market share of Opera browsers is still low.

Each browser interprets the Web site code in a different manner, which means that it can appear differently to users using different browsers.

The best practices for optimum browser compatibility are as follows:

- **Test the Web site in different browsers** - Once the Web site design is created, review the Web site's appearance and functionality on multiple browsers to ensure that all the users are getting the same experience according to the design. Preferably test on different versions of the same browser also as they can show the Web site differently.

- **Write a good clean HTML coding** - Sometimes the Web site may appear correctly in some browsers even if the HTML code is not valid, but there is no guarantee that it will appear correctly in all the browsers. To ensure that the page looks same in all browsers is to write Web pages using valid HTML and CSS codes, and then test it in many browsers. Using External CSS can help pages render and load faster.

## *7.8 Check Your Progress*

1. It is common for two images with the same number of _____ and _____ to have a very different compressed file size.

| | | | |
|---|---|---|---|
| (A) | pixels, color depth | (C) | size, container |
| (B) | height, width | (D) | content, size |

2. Which of the following browsers support CSS3 Transitions?

| | | | |
|---|---|---|---|
| (A) | Apple Safari 2.1 | (C) | Opera 10.5x |
| (B) | Google Chrome | (D) | IE 4 |

3. The CSS padding properties define the _____ between the element border and the element content.

| | | | |
|---|---|---|---|
| (A) | size | (C) | distance |
| (B) | space | (D) | font |

4. Match the features against the usability.

| Value | | Description | |
|---|---|---|---|
| (A) | auto | 1. | Specifies that the value of the height property should be inherited from the parent element |
| (B) | length | 2. | Defines the height in percent of the containing block |
| (C) | % | 3. | Defines the height in px, cm, etc. |
| (D) | inherit | 4. | The browser calculates the height. This is default |

| | | | |
|---|---|---|---|
| (A) | a-2, b-1, c-3, d-4 | (C) | a-4, b-1, c-2, d-3 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-4, b-3, c-2, d-1 |

5. Which of the following is the engine for Safari browser?

| | | | |
|---|---|---|---|
| (A) | Presto | (C) | Trident |
| (B) | WebKit | (D) | Gecko |

6. Which of the following statements are valid for performing CSS transitions?

| | |
|---|---|
| (A) | Specify the name for transition |
| (B) | Specify the CSS property, that needs to add an effect |
| (C) | Specify the duration of the effect |
| (D) | Specify the language to be used |

## *7.8.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | B, C |
| 3. | B |
| 4. | D |
| 5. | B |
| 6. | B, C |

# Summary

- The text styles specify and control the appearance of the text in a Web page.
- Indenting is the process of offsetting text from its normal position, either to the left or to the right.
- CSS border property specifies the style, color, and width of the border.
- The border-color property accepts different color values that determine the different shades of color to be applied to the borders.
- The values of the different border properties determine the type of effect to be applied to the borders.
- In CSS, the text-align property is used for horizontal alignment of text in an element.
- In CSS, the line-height property is used for vertical alignment of text in an element.

# *Try it Yourself*

1. Alex wants to write an article on his Web page related to their products. For writing major quotes of the article he wants to provide suitable width with white background. Also, he wants headings and images in an article should be center aligned. Use CSS code to create this Web page.

# Session - 8
# Creating Navigational Aids and Division-Based Layout

Welcome to the Session, **Creating Navigational Aids and Division-Based Layout**.

This session explains the different types of navigation bar, HTML5 semantic tags and layout. Finally, this session explains the usage of divisions in HTML5.

In this Session, you will learn to:

- ➔ Explain HTML5 semantic tags
- ➔ Explain HTML5 semantic tag layouts
- ➔ Explain the usage of navigation bar
- ➔ Describe a text-based and graphical navigation bar
- ➔ Explain image mapping
- ➔ Explain divisions in HTML5

## *8.1 Introduction*

For designing a Web site, a number of elements and principles are used to get the desired results. Using these principles and elements helps to develop a rich, attractive, efficient, and aesthetically pleasing Web site. In other words, using some HTML tags and graphics does help to develop a useful and efficient Web site. Navigation bar plays an important role in making the Web page user-friendly.

## *8.2 HTML5 Semantic Tags*

HTML5 has evolved by introducing new elements that brought semantics to higher level. New tags were developed to create stable semantic structure. The earlier version of HTML had the universal tag div which was used to accomplish various tasks in the HTML structure. The constraint with div tag is that, it confused the user when multiple div tag was used in large coding. Now, HTML5 has introduced two types of semantic tags. They are namely, text-level and structural.

### *8.2.1 Structural Semantic Tags*

They are the block level elements and are used to structure pages. The new structural semantic elements are as follows:

- **Section** - The section element represents a section of a Web document. It is used for grouping related content and is different from other content groups present on the Web page. It is similar to a div tag though section element has more semantic meaning. In other words, section element is more meaningful as the content inside the section tags should be related.

- **Header** - The header element represents the header of a Web page. It can be used either at the top of the document or at the top of a section. Though most of the Web sites currently uses a single header at the top of the page called masthead, but a Web developer can have multiple headers in a single HTML5 document. This element is used as a container containing a group of introductory content or a set of navigational links.

- **Footer** - The footer is similar to the header and can be present as the footer either for the document or for the section. There can be multiple footer elements in an HTML5 document. A footer element has information about the Web document. The typical contents which are placed in footer are as follows:

  - Authors information
  - Copyright information
  - Text-based navigation bar

  Any metadata for the section can also be included in a footer tag.

- **Aside** - The aside element is used for representing the content that is related to the main text of the document. It aligns itself as a sidebar. As compared with other structural tags its importance is not related with its position within a document, but rather its relationship with the content. It is not mandatory to have an aside element aligned to the right or left of a Web page. It can be at the top, the bottom, or even in the middle of a Web page.

- **Nav** - The nav element represents a section of a Web page that contains navigation links/menus to other Web pages or to other parts within the Web page. In other words, it allows the user to navigate through the Web page and site. This section is created for major navigational information such as a navigation bar for the entire site or for a subsection menu.

- **Article** - The article element represents a section of content that is independent of a Web page or site content. It is self-contained and stands on its own. The possible sources for the article tag are as follows:
  - Blog post
  - News story
  - Comment
  - Review
  - Forum post

**Note** - The div tag must not be entirely replaced by the semantic tags. The semantic tags must only be used for semantically appropriate scenarios.

## *8.2.2 Text-level Semantic Tags*

The text level semantic tags are currently inline elements and they are as follows:

- **Mark** - The `<mark>` tag is used for defining marked or highlighted text because of its relevance to the context. For example, a mark tag can be used for highlighting words in a Web page that a visitor searched for.

- **Time** - The `<time>` tag is used for defining either the time, or a date in the Gregorian calendar. It is used optionally with a time and a time-zone offset. This element can be used to encode dates and times in a machine-readable format. For example, a Web user can add birthday reminders or scheduled events to the user's calendar and enable the search engines to produce better search results.

Table 8.1 lists attribute and value of `<time>` tag.

| Attribute | Value | Description |
|---|---|---|
| datetime | `datetime` | Provides the date/time given by the element's content |
| pubdate | `pubdate` | It is used for specifying publication date and time of the document |

**Table 8.1: Attribute and Value of `<time>` Tag**

Code Snippet 1 demonstrates the code to display date and time.

**Code Snippet 1**:

```
<!DOCTYPE html>
<html>
<body>
<time datetime="13:00">1pm</time>
<time datetime="2011-07-15">July 15th, 2011</time>
<time datetime="2011-07-15T13:00">1pm on July 14th</time>
</body>
</html>
```

The datetime attribute is not mandatory.

Code Snippet 2 demonstrates the code to display date.

**Code Snippet 2**:

```
<time>2011-07-14</time>
```

➔ **Meter** - The `<meter>` tag displays markup or scalar measurement within a defined range. Absolute scalar values, such as height or weight, are not represented automatically by the meter tag. For this, the user must specify the height and weight within the known range of values. It is also used for displaying fractional value.

Code Snippet 3 demonstrates the code to display `<meter>` tag.

**Code Snippet 3**:

```
<meter value="2" min="0" max="10">2 out of 10</meter>
```

Table 8.2 lists attributes and value of `<meter>` tag.

| Attribute | Value | Description |
|---|---|---|
| form | `form_id` | Is used for specifying one or more forms that <meter> element belongs to |
| high | `number` | Is used for specifying the high range value |
| low | `number` | Is used for specifying a range of value that is to be considered as low and should be greater than min attribute value |
| max | `number` | Is used for specifying the maximum value of the range |
| min | `number` | Is used for specifying the minimum value of the range |
| optimum | `number` | Is used for specifying the optimal value for the <meter> tag |
| value | `number` | Is used for specifying the current value of the <meter> tag |

**Table 8.2: Attributes and Value of `<meter>` Tag**

➔ **Progress** - The `<progress>` tag can be used with JavaScript to display the progress of a task.

Table 8.3 lists attributes and value of `<progress>` tag.

| Attribute | Value | Description |
|---|---|---|
| max | `number` | Is used for specifying the work as a floating point number that the task requires in total |
| value | `number` | Is used for specifying how much task has been completed |

**Table 8.3: Attributes and Value of `<progress>` Tag**

Code Snippet 4 demonstrates the code to display `<progress>` tag.

**Code Snippet 4**:

```
<progress value="24" max="120"></progress>
```

## *8.3 HTML5 Semantic Layout*

The sample representation of a Web page layout is displayed in figure 8.1. The section will explain every section defined and the purpose of each section.

In other words, figure 8.1 shows the HTML5 semantic layout and used elements for every section.

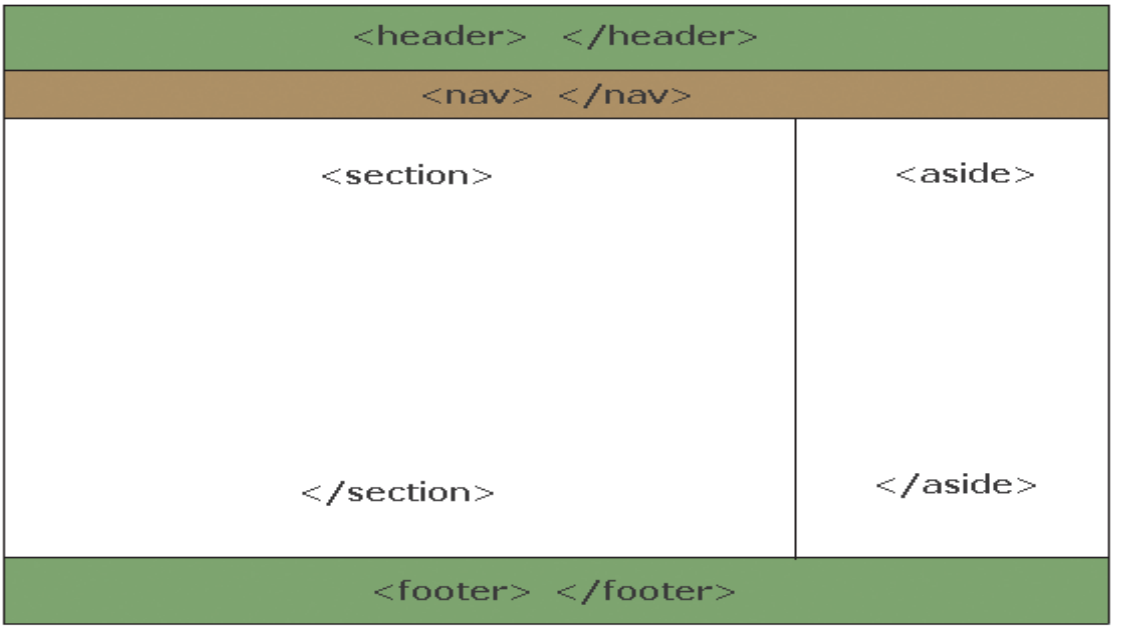



Figure 8.1: HTML5 Semantic Layout

## ➔ <header>

The `<header>` element provides introductory information. This information can include titles, subtitles, logos, and so on. It can also include the navigational aids. The `<head>` tag provides information about the entire document, whereas the `<header>` tag is used only for the body of the Web page or for the sections inside the body.

Code Snippet 5 demonstrates the use of `<header>` tag.

**Code Snippet 5**:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First Page</title>
  </head>
  <body>
  <header>
  <h1>Sample Blog </h1>
  </header>
  </body>
</html>
```

In the code, the `<header>` element shows the commencement of the body. This is the visible part of the document. Inside the `<header>`, the `<h1>` element is used to indicate the importance of the heading.

### ➔ <nav>

The nav element is a section which contains the links to other pages or links to different sections within the page. In other words, it is a section containing the navigation links. Navigational elements are helpful in identifying large blocks of navigational data and are generally not preferred for small navigational displays.

Code Snippet 6 demonstrates the use of `<nav>` tag.

**Code Snippet 6**:

```
<body>
<header>
<h1>Sample Blog</h1>
</header>
<nav>
<ul>
<li> home </li>
<li> help </li>
<li> contact </li>
</ul>
</nav>
</body>
```

In the code, the `<nav>` element is present between the `<body>` tags but after the closure of `<header>` tag.

### ➔ <section>

It is the main information bar that contains the most important information of the document and it can be created in different formats. For example, it can be divided into several blocks or columns.

For example, a Web site's home page could be divided into sections for an introduction, news updates, and contact information.

Code Snippet 7 demonstrates the use of `<section>` tag.

**Code Snippet 7**:

```
<body>
 <header>
 <h1>Sample Blog </h1>
 </header>
 <nav>
<ul>
<li> home </li>
<li> help </li>
<li> contact </li>
</ul>
 </nav>
 <section>
  <h1>Links</h1>
 <ul>
   <li><a href="#">Link 1</a></li>
   <li><a href="#">Link 2</a></li>
   <li><a href="#">Link 3</a></li>
 </ul>
  </section>
  </body>
```

Similar to the navigation bar, the main information bar is a separate section. Therefore, the main information bar appears after the `</nav>` closing tag.

## ➔ <aside>

The `<aside>` element is a column or a section that generally contains data linked to the main information but not as relevant or important as the main information. This element is used for typographical effects, such as for sidebars, for groups of nav elements, for advertising purposes, and for other content that cannot form a part of the main content of the page.

Code Snippet 8 demonstrates the use of `<aside>` tag.

**Code Snippet 8**:

```
<!DOCTYPE html>
<html lang="en">
<body>
<header>
<h1>Sample Blog</h1>
</header>
<nav>
<ul>
  <li>home</li>
  <li>help</li>
  <li>contact</li>
</ul>
</nav>
<section>
<h1>Links</h1>
<ul>
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
</ul>
</section>
<aside>
<blockquote>Archive Number One</blockquote><br>
<blockquote>Archive Number Two</blockquote>
</aside>
</body>
</html>
```

The `<aside>` element can be placed in any part of the site layout. It can also be used in any way as long as the content is not considered as the main content of the document.

Figure 8.2 shows output of Code Snippet 8.

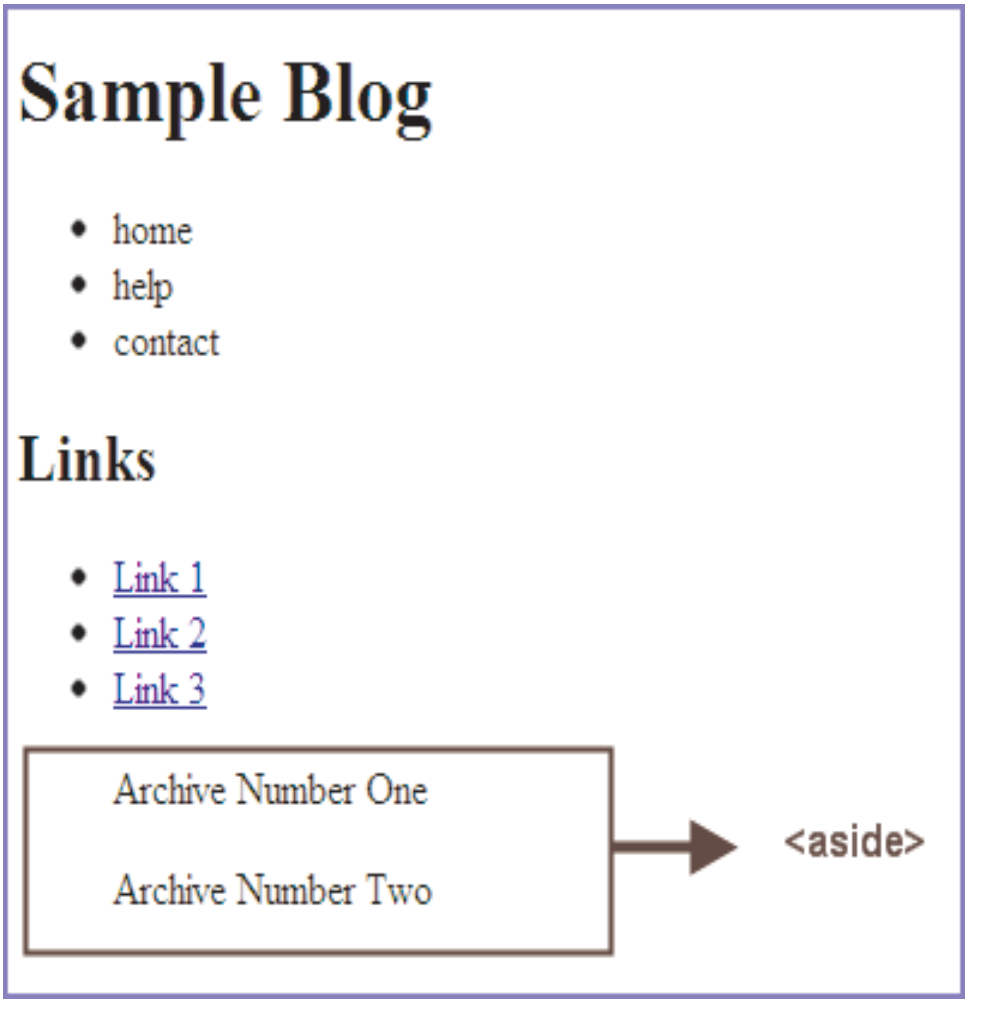



**Figure 8.2: Output of Code Snippet 8**

## ➔ <footer>

HTML5 provides the `<footer>` element to give an end to the document's body. A footer typically contains information about the sections. This can include the author or company details, links to related documents, copyright data, and so on.

Code Snippet 9 demonstrates the use of `<footer>` tag.

**Code Snippet 9**:

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

```
<body>
 <header>
 <h1>Sample Blog </h1>
 </header>
 <nav>
<ul>
<li> home </li>
<li> help </li>
<li> contact </li>
</ul>
 </nav>
 <section>
 <h1>Links</h1>
 <ul>
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
 </ul>
 </section>
 <aside>
 <blockquote>Archive Number One</blockquote> <br>
 <blockquote>Archive Number Two</blockquote>
 </aside>
 <footer>
  Copyright &copy; 2012-2013
 </footer>
 </body>
 </html>
```

Usually, the `<footer>` element represents the end of the body section. However, the `<footer>` tag can be used many times inside the body to represent the end of different sections.

Figure 8.3 shows output of Code Snippet 9.

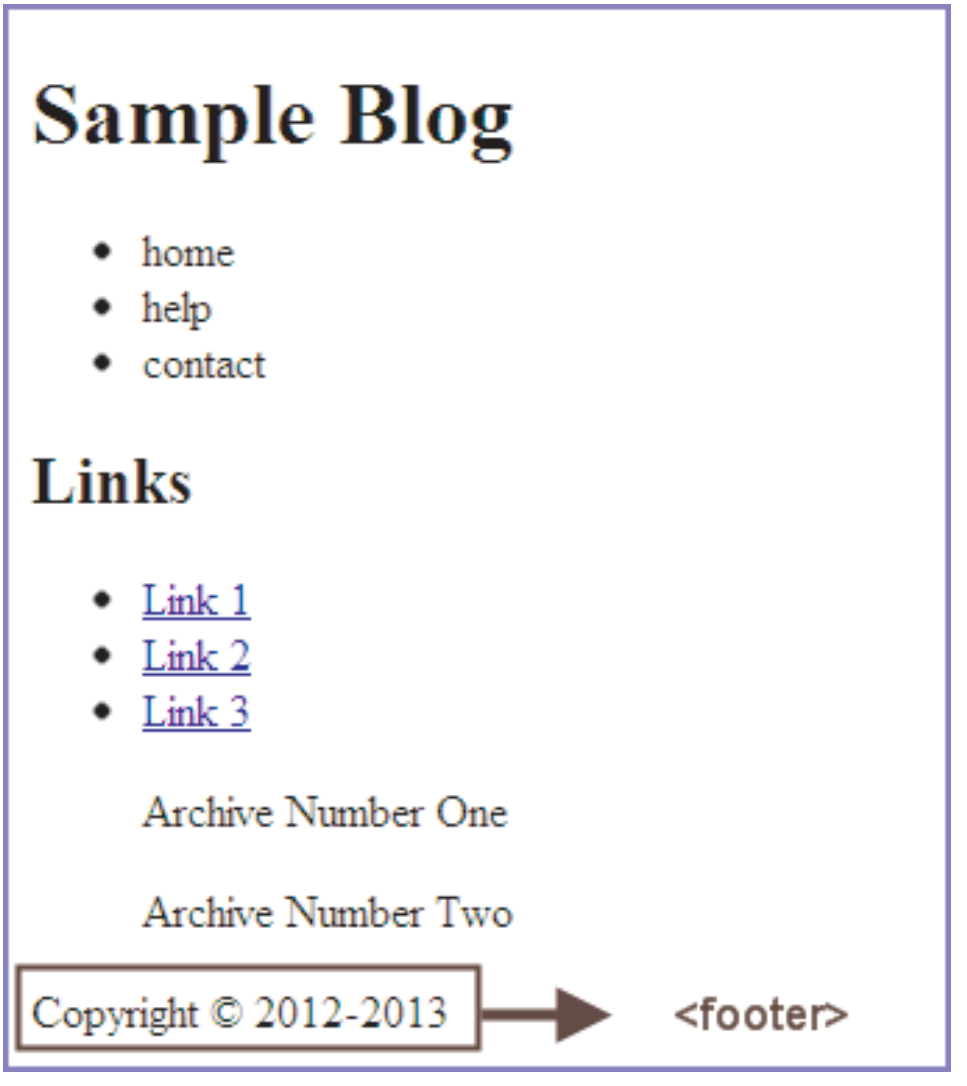



**Figure 8.3 Output of Code Snippet 9**

## ➔ <article>

The `<article>` element helps to insert a self-contained composition in an application, page, document, or site. For example, an `<article>` element could be an interactive widget, an entry in a blog, an article in a newspaper or magazine, a post in a forum, a comment submitted by a user, or any other independent content.

Code Snippet 10 demonstrates the code for `<article>` tag.

**Code Snippet 10**:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
 </head>
<body>
 <header>
 <h1>Sample Blog </h1>
 </header>
 <nav>
<ul>
<li> home </li>
<li> help </li>
<li> contact </li>
</ul>
 </nav>
 <section>
     <article>
      First Blog entry
     </article>
     <article>
      Second Blog entry
     </article>
 </section>
 <aside>
 <blockquote>Archive Number One</blockquote>
 <blockquote>Archive Number Two</blockquote>
 </aside>
<footer>
```

```
Copyright &copy; 2012-2013
 </footer>
 </body>
 </html>
```

In the code, the `<article>` tags are placed within the `<section>` tags. This indicates that the `<article>` tag belongs to this section. The `<article>` tags are placed individually one after another, because each one is an independent part of the `<section>`.

Figure 8.4 shows output of Code Snippet 10.

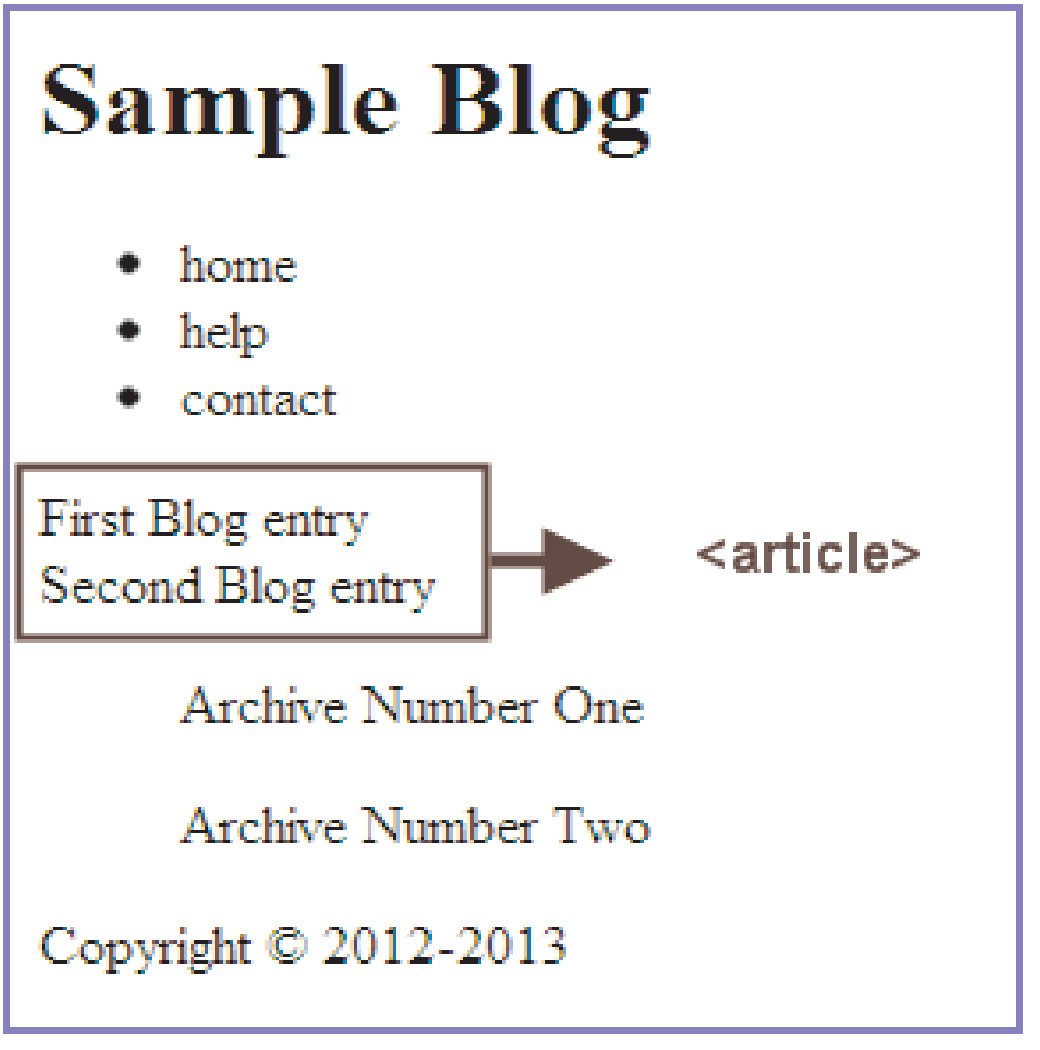



Figure 8.4: Output of Code Snippet 10

Figure 8.5 shows the new layout, after adding the `<article>` tag.

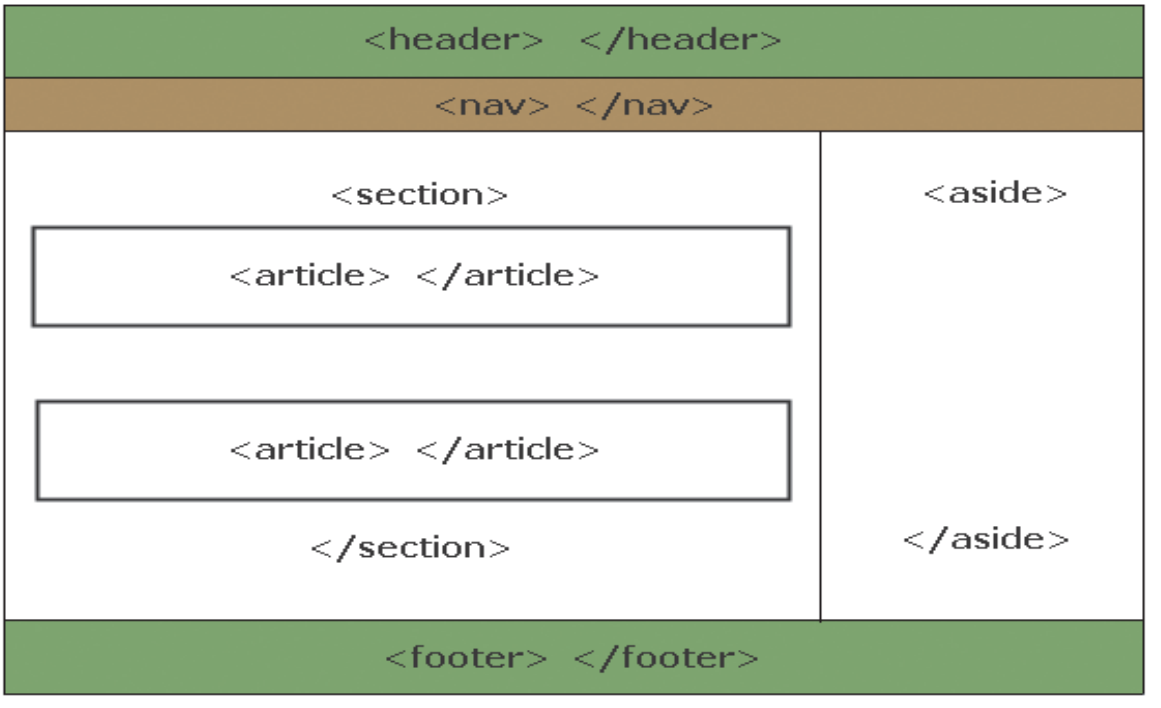



**Figure 8.5: New Layout after Adding the `<article>` tag**

The content of every `<article>` element has its own independent structure.

## *8.4 Navigation Bar*

Navigation is one of the most important elements in Web design. Web-layouts do not have any specific physical representation that a user can depend on except for a consistent navigation menu. It is one of the most important design elements which provide the users with a sense of orientation and guide them through the Web site. Thus, it can be said that navigation is one segment of a Web site's information architecture, but it plays an important role as it is the most visible segment to the end user.

In Web designing, a navigation menu is always on navigation bars, which can be horizontal or vertical. A navigation bar is a section of a Web site or online page intended to support visitors in browsing through the online document. Typically, Web pages will have a primary and a secondary navigation bar on all pages which will include links to the most important sections of the site.

### *8.4.1 Text-based Navigation Bar*

Navigation menu is the most used element than any other element on any Web page. Therefore, it is important to make sure that the Web site visitors should be able to easily navigate through the site structure.

Some users browse Web site with graphics turned off, or use browsers with minimum graphics capability. For such situations, it is essential to provide text-based navigation bars which are created as a stand-alone navigation bars. The developer in addition can also provide a graphical bar. Text-based navigation bars are not associated with icons but are easy to create, and can be displayed in any Web browsers. The advantage of using a text-based navigation bar is that it reduces the loading time of a page. Although a text-based navigation bar is easy to create, it is not interesting because there is very less interaction or visual appeal to the visitor. Text links are hard to distinguish from the regular text that appears on a Web page.

It can be displayed either horizontally or vertically. The font (best to use Web safe fonts), color, and link colors can be determined by the user via the Font pane.

Code Snippet 11 demonstrates the HTML code for a text-based navigation bar.

**Code Snippet 11**:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<nav>
<a href="/home/"><font size="6">Home</font></a> |
<a href="/news/"><font size="6">News</font></a> |
<a href="/contact/"><font size="6">Contact</font></a> |
<a href="/about/"><font size="6">About</font></a>
</nav>
<h1>This is a Text-based Navigation Bar</h1>
</body>
</html>
```

Figure 8.6 shows the output of text-based navigation bar.

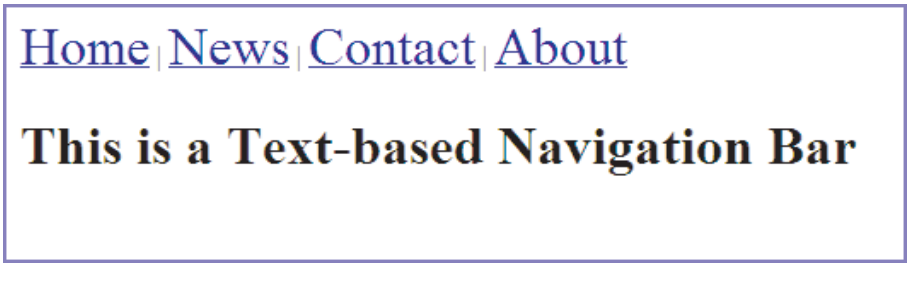



Figure 8.6: Output of Text-Based Navigation Bar

## *8.4.2 Graphical Navigation Bar*

Graphical navigation bar is more captivating than text-based navigation bar as it uses icons. The usability of the page increases with a good choice of icon for the navigation bar. It can also make the Web site more noticeable for the user visiting the Web site. In other words, graphical navigation bar is better than text-based navigation as it gives a visual appeal to the visiting users. The only disadvantage is that, since it uses images, it takes longer time for a page to load. Also, the Web page will be useless for users using a non-graphic browser.

### Graphical Navigation Bar with Rollover Effects

They are similar to the graphical navigation bar except for the additional feature. Moving the mouse over the linked image leads to a change in the state of image. In other words, the state change of image leads to an image swapping process. When the mouse is moved off the image, the image swaps back to the previous view. This rollover effect creates an interactive activity between the Web site and the visitor. This rollover effect has two different activities that include the image in the original view and the changed image after mouse rollover.

**Note** - The pre-load time of a Web page will increase if the rollover effect is used as all the images is required to be fully loaded on the Web page.

Code Snippet 12 demonstrates CSS and HTML code for a graphical navigation bar.

**Code Snippet 12**:

```
<!DOCTYPE html>
<html>
<head>
<style>
/* Main Navigation */
#nav {
 position:relative;
 width:620px;
 margin:0 auto;
 margin-top:50px;
 padding:10px;
}
ul#navigation {
 margin:0px auto;
 position:relative;
```

```
float:left;
 border-left:1px solid #c4dbe7;
 border-right:1px solid #c4dbe7;
}
ul#navigation li {
 display:inline;
 font-size:12px;
 font-weight:bold;
 margin:0;
 padding:0;
 float:left;
 position:relative;
 border-top:1px solid #c4dbe7;
 border-bottom:2px solid #c4dbe7;
}
ul#navigation li a {
 padding:10px 25px;
 color:#616161;
 text-shadow:1px 1px 0px #fff;
 text-decoration:none;
 display:inline-block;
 border-right:1px solid #fff;
 border-left:1px solid #C2C2C2;
 border-top:1px solid #fff;
 background: #f5f5f5;
 -webkit-transition:color 0.2s linear, background 0.2s linear;
 -moz-transition:color 0.2s linear, background 0.2s linear;
```

```
 -o-transition:color 0.2s linear, background 0.2s linear;
 transition:color 0.2s linear, background 0.2s linear;
}
ul#navigation li a:hover {
 background:#f8f8f8;
 color:#282828;
}
ul#navigation li a.first {
 border-left: 0 none;
}
ul#navigation li a.last {
 border-right: 0 none;
}
ul#navigation li:hover > a {
 background:#00FF00;
}
/* Drop-Down Navigation */
ul#navigation li:hover > ul
{
/*these 2 styles are very important,
being the ones which make the drop-down to appear on hover */
 visibility:visible;
opacity:1;
}
ul#navigation ul, ul#navigation ul li ul {
 list-style: none;
 margin: 0;
 padding: 0;
```

```
/*the next 2 styles are very important,
being the ones which make the drop-down to stay hidden */
 visibility:hidden;
 opacity:0;
 position: absolute;
z-index: 99999;
 width:180px;
 background:#f8f8f8;
 box-shadow:1px 1px 3px #ccc;
/* css3 transitions for smooth hover effect */
 -webkit-transition:opacity 0.2s linear, visibility 0.2s linear;
 -moz-transition:opacity 0.2s linear, visibility 0.2s linear;
 -o-transition:opacity 0.2s linear, visibility 0.2s linear;
 transition:opacity 0.2s linear, visibility 0.2s linear;
}
ul#navigation ul {
 top: 43px;
 left: 1px;
}
ul#navigation ul li ul {
top: 0;
 left: 181px; /* strong related to width:180px; from above */
}
ul#navigation ul li {
 clear:both;
 width:100%;
border:0 none;
 border-bottom:1px solid #c9c9c9;
```

```
 }
ul#navigation ul li a {
 background:none;
 padding:7px 15px;
 color:#616161;
 text-shadow:1px 1px 0px #fff;
 text-decoration:none;
 display:inline-block;
 border:0 none;
 float:left;
 clear:both;
 width:150px;
}
</style>
</head>
<body>
<nav id="nav">
<ul id="navigation">
<li><a href=""><font size="4"><img src="Home-icon.png" height="20"
width="20">Home</img></font></a></li>
<li><a href=""><font size="4"><img src="icon-news.png" height="20"
width="20">News</font></a></li>
<li><a href=""><font size="4"><img src="Address-Book-icon.png" height="20"
width="20">Contact</font></a></li>
<li><a href=""><font size="4"><img src="User group.png" height="20"
width="20">About</font></a></li>
</nav>
<br/>
<br/>
<br/>
```

```
<br/>
<h1>This is a Graphical Navigation Bar</h1>
</body>
</html>
```

Figure 8.7 shows output of graphical navigation bar.

Figure 8.7: Output of Graphical Navigation Bar

## *8.5 Image Map*

Image maps are images with clickable areas. These areas in image-maps when clicked will link to another page. The image maps have to be used intelligently to make it effective. If they are not used appropriately they can confuse the users. The `<map>` tag is used to define an image-map. The `<map>` element contains a number of `<area>` elements for defining the clickable areas in the image map. In HTML5, if the id attribute of the `<map>` tag is specified, then it must have the same value as the name attribute.

Table 8.4 shows `<map>` tag attribute and its value.

| Attribute | Value | Description |
|---|---|---|
| name | `mapname` | It is used for specifying the name of an image-map |

Table 8.4: `<map>` Tag Attribute and its Value

Follow these guidelines to create an image map:

1. Use the `<img>` tag to insert and link an image. In the `<img>` tag, use the usemap attribute to define the image map name.
2. Use the `<map>` tag to create a map with the same name. Inside this `<map>` tag, define the clickable areas with the `<area>` tag.

Code Snippet 13 demonstrates the use of image map in an html code.

**Code Snippet 13**:

```
<!DOCTYPE html>
<html>
<body>
<img src="6.jpg" width="600" height="300" alt="cake" usemap="#cakemap" />
<map name="cakemap">
 <area shape="circle" coords="0,0,200,600" href="4.html" alt="cake" />
</map>
</body>
</html>
```

Figure 8.8 shows the output of an image map.

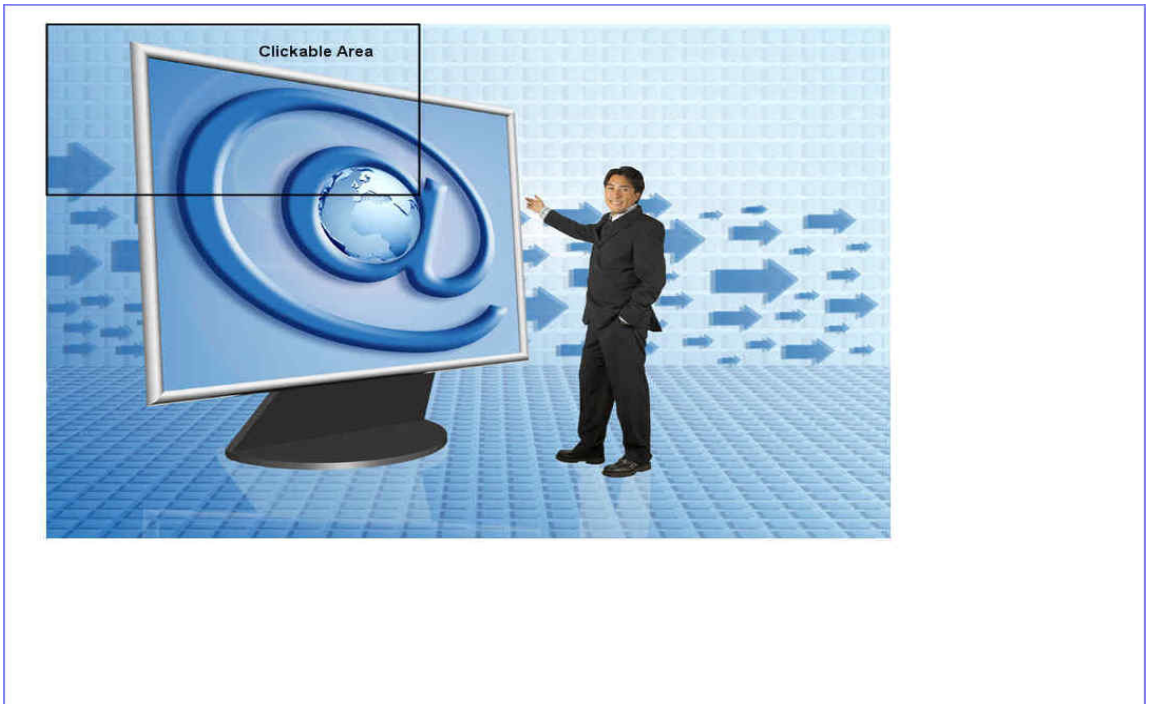



**Figure 8.8: Output of Image Map**

## *8.6 Divisions*

The  tag defines a division in an HTML Web page. It is used to group block-elements and format them with CSS. The new structural semantic tags reasonably reduce a lot of  tag's usage, but  tag is still important in the HTML5 world. The  tag can be used when there is no other semantically appropriate element left that suits the purpose in a Web page development. It can be commonly used for stylistic purposes such as wrapping some semantically marked-up content in a CSS-styled container.

Code Snippet 14 demonstrates an HTML code to show the use of  tag used for wrapping.

**Code Snippet 14**:

```
<body>
 <div id="wrapper">
  <header>
   <h1>Hello</h1>
   <nav>
    <! -- ... -->
   </nav>
  </header>
 </div>
</body>
```

Tips for using  tag in Web site development are as follows:

- The  tag is a block-level element.
- The  tag can contain any other tag.
- In HTML5, the  tag can be found inside any element that can contain flow elements, such as other , <address>, <section>, and <table>.

### *8.6.1 Division Positioning and Formatting*

Elements can be positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position of the property is set. They also work differently depending on the positioning method. There are five position properties in DIV elements namely, static, relative, absolute, fixed, and inherit. For easy usage, only three properties are used namely, absolute, relative, and fixed. Positioning can be applied to any block element. The default position for a block element (DIV) is static. Table 8.5 shows the various values that can be used in DIV element to position elements.

| Value | Description |
|---|---|
| static | Positions the element in order, as they appear in the document flow. It is the default value. |
| absolute | Positions the element relative to its first position. |
| fixed | Positions the element relative to the browser window. |

| Value | Description |
|---|---|
| relative | Positions the element relative to its normal position. |
| inherit | Positions the element with respect the value that is inherited from the parent element. |

Table 8.5: Values For Positioning With the DIV Element

Code Snippet 15 demonstrates the CSS code for `<div>` tags with different position properties.

**Code Snippet 15**:

```
.lCard{
width: 100px;
height:100px;
background-color:blue;
padding: 6px;
position:fixed;
left:450px;
top:100px;
}
.rCard{
width: 100px;
background-color:red;
padding: 7px;
position:relative;
top:93px;
left:300px;
}
.bCard{
width: 100px;
height:100px;
background-color:green;
```

```
padding: 6px;
position:absolute;
left:310px;
bottom:320px;
}
```

Include the div tag within the style section of the HTML body. Code Snippet 16 demonstrates HTML code for `<div>` tags.

**Code Snippet 16**:

```
<body>
     <div class="rCard">
     </div>
     <div class="bCard">
     </div>
     <div class="lCard">
     </div>
 </body>
```

Figure 8.9 shows the output of division positioning.

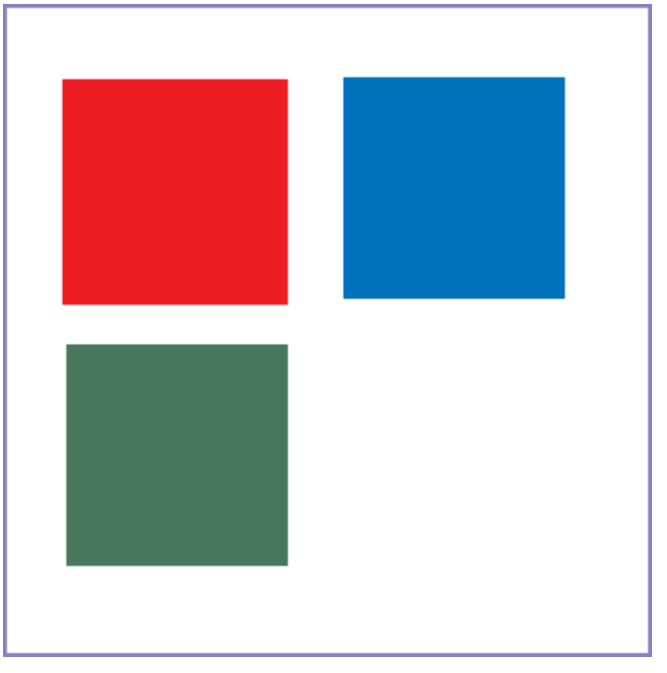


**Figure 8.9: Output of Division Positioning**

Multiple columns are created by using the `<div>` tag and CSS is used to format the divisions. Divisions can be formatted by using the same character, paragraph, and page formatting styles. In the Code Snippet, CSS is used to format each DIV.

## *8.7 Check Your Progress*

1. Which of the following navigation bar will load easily?

| | | | |
|---|---|---|---|
| (A) | Text-based | (C) | graphical |
| (B) | Image-based | (D) | Font-based |

2. Image maps are images with clickable _____ that link to another page.

| | | | |
|---|---|---|---|
| (A) | images | (C) | areas |
| (B) | texts | (D) | links |

3. Which tag is used to represents content that is tangentially related to the main text of a document?

| | | | |
|---|---|---|---|
| (A) | `<section>` | (C) | `<header>` |
| (B) | `<aside>` | (D) | `<article>` |

4. Match the following.

| Value | | Description | |
|---|---|---|---|
| (A) | static | (1) | The element is positioned relative to its normal position. |
| (B) | absolute | (2) | The element is positioned relative to the browser window. |
| (C) | fixed | (3) | The element is positioned relative to its first positioned. |
| (D) | relative | (4) | Elements renders in order, as they appear in the document flow. This is default. |

| | | | |
|---|---|---|---|
| (A) | a-2, b-1, c-3, d-4 | (C) | a-4, b-3, c-2, d-1 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-3, b-2, c-4, d-1 |

5. Match the following.

| Value | | Description | |
|---|---|---|---|
| (A) | meter | (1) | Used for displaying the publication date and time |
| (B) | mark | (2) | Used with JavaScript for displaying the progress of a task |
| (C) | pubdate | (3) | Used for displaying fractional value |
| (D) | progress | (4) | Used for defining marked or highlighted text |

| | | | |
|---|---|---|---|
| (A) | a-3, b-4, c-1, d-2 | (C) | a-4, b-3, c-2, d-1 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-2, b-1, c-4, d-3 |

6. Which tag is used to markup measurements or a scalar measurement within a known range?

| | | | |
|---|---|---|---|
| (A) | `<time>` | (C) | `<header>` |
| (B) | `<meter>` | (D) | `<progress>` |

## *8.7.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | C |
| 3. | B |
| 4. | C |
| 5. | A |
| 6. | B |

# Summary

- HTML5 has introduced two types of semantic tags. They are namely, text-level and structural. Structural semantic tags are as follows:
  - Section
  - Header
  - Footer
  - Aside
  - Nav
  - Article
- Text-level semantic tags are as follows:
  - Mark
  - Time
  - Meter
  - Progress
- Navigation is the most significant element in Web design. Since Web-layouts does not have any physical representation a user can depend on consistent navigation menu.
- Text-based navigation bars are created as stand-alone navigation bars that are not associated with icons. Text-based navigation bar is easy to create and can be displayed in any Web browsers.
- Graphical navigation bar is better than text-based navigation as it gives a visual appeal to the visiting users.
- The new structural semantic tags reasonably capture a lot of 's territory, but  tag still has a place in the HTML5 world. Div can be used when there is no other semantically appropriate element left that suits the purpose in a Web page development.

# *Try it Yourself*

1. ABC Inc. is a gaming company which is planning to start its online site for games. ABC Inc. needs a navigation bar which will give information about their products, company itself, support, and online gaming center, and so on.

   Navigation bar should have the following links:

   1. Home
   2. About us
   3. Products
   4. Gaming  Center
   5. Support
   6. Contacts

   You as a Web site designer have been assigned the task for developing the Web site for the company using the new semantic tag.

# MANY COURSES ONE PLATFORM

# Session - 9

# Creating Tables

Welcome to the Session, **Creating Tables**.

This session explores how to create tables, specify the size of the table, and the width of a column in a table. The session also describes the merging of table cells, how to define page layout for tables, and apply formatting to tables.

In this Session, you will learn to:

- ➔ Describe how to create and format tables
- ➔ Explain the table size and the width of a column
- ➔ Explain the process of merging table cells
- ➔ Explain the page layout for tables

## *9.1. Introduction*

Tables allow the user to view the data in a structured and classified format. Tables can contain any type of data such as text, images, links, and other tables. The user can create tables for displaying timetables, financial reports, and so on.

## *9.2. Creating and Formatting Tables*

A table is made up of rows and columns. The intersection of each row and column is called as a cell. A row is made up of a set of cells that are placed horizontally. A column is made up of set of cells that are placed vertically.

The user can represent the data in a tabular format by using the `<table>` element in HTML. The `<tr>` element divides the table into rows and the `<td>` element specifies columns for each row. By default, a table does not have a border. The `border` attribute of the `<table>` element specifies a border for making the table visible in a Web page.

Code Snippet 1 demonstrates how to create a table.

**Code Snippet 1:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Languages</title>
  </head>
   <body>
    <h2>Main Languages</h2>
     <table border="1">
      <tr>
       <td>English</td>
       <td>German</td>
      </tr>
      <tr>
       <td>French</td>
       <td>Italian</td>
      </tr>
    </table>
   </body>
</html>
```

The code uses the `<table>` element to create a table. The border attribute of `<table>` element gives a border to the table, which is 1 pixel wide. The `<tr>` element within the `<table>` element creates rows. The `<td>` element creates two cells with the values English and German in the first row and French and Italian in the second row.

Figure 9.1 displays a table created.

**Figure 9.1: Table Created**

## *9.2.1 Table Headings*

The user can specify the heading for each column in HTML. To specify the heading for columns in a table, use the `<th>` element.

The text included within the `<th>` element appears in bold. Code Snippet 2 demonstrates how to create a table with a heading.

**Code Snippet 2:**

```
<!DOCTYPE HTML>
  <html>
    <head>
      <title>List of Students </title>
    </head>
    <body>
      <h2>List of Students</h2>
      <table border="1">
```

```
          <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Place</th>
          </tr>
          <tr>
            <td>Mark</td>
            <td>17</td>
            <td>Madrid</td>
          </tr>
          <tr>
            <td>John</td>
            <td>19</td>
            <td>London</td>
          </tr>
        </table>
      </body>
</html>
```

In this code, the `<table>` element creates a table with a `border` of 1 pixel. The `<th>` element provides three column headings namely, **Name**, **Age**, and **Place**.

The second and the third row lists the details of the students in the three columns.

Figure 9.2 displays the output of the table with headings.

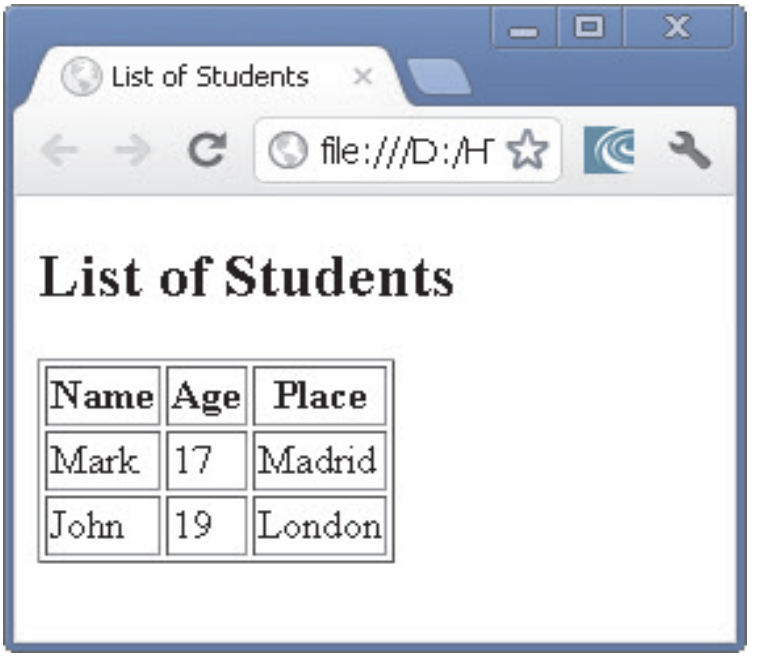



Figure 9.2: Table with Headings

## *9.2.2 Colspan Attribute*

The user might feel the need to span two or more cells while working with tables. Spanning refers to a process of extending a cell across multiple rows or columns. To span two or more columns, use the `colspan` attribute of the `<td>` and `<th>` elements.

The `colspan` attribute allows the user to span a cell along a horizontal row. The value of the `colspan` attribute specifies the number of cells across which a specific cell shall be expanded. Code Snippet 3 demonstrates how to create a table and span header cells across two cells vertically.

**Code Snippet 3:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Employee Details</title>
  </head>
  <body>
    <h2>Employee Details</h2>
    <table border="1">
```

```
      <tr>
        <th colspan="2">IT</th>
        <th colspan="2">Accounts</th>
      </tr>
      <tr>
        <th>Name</th>
        <th>Location</th>
        <th>Name</th>
        <th>Location</th>
      </tr>
      <tr>
        <td>David</td>
        <td>New York</td>
        <td>John</td>
        <td>London</td>
      </tr>
      <tr>
        <td>Katthy</td>
        <td>New Jersey</td>
        <td>Peter</td>
         <td>Los Angeles</td>
      </tr>
    </table>
  </body>
</html>
```

The code creates a table with a border of 1 pixel. The `<th>` element specifies two column headings namely, **IT** and **Accounts**. Each of these header cells horizontally span across the two cells by setting the colspan attribute of the `<th>` element to **2**. Each of these headings has two sub-headings namely, Name and Location, which specify the **name** and **location** of employees. The first and second rows display the details of the employees.

Figure 9.3 displays the employee details.

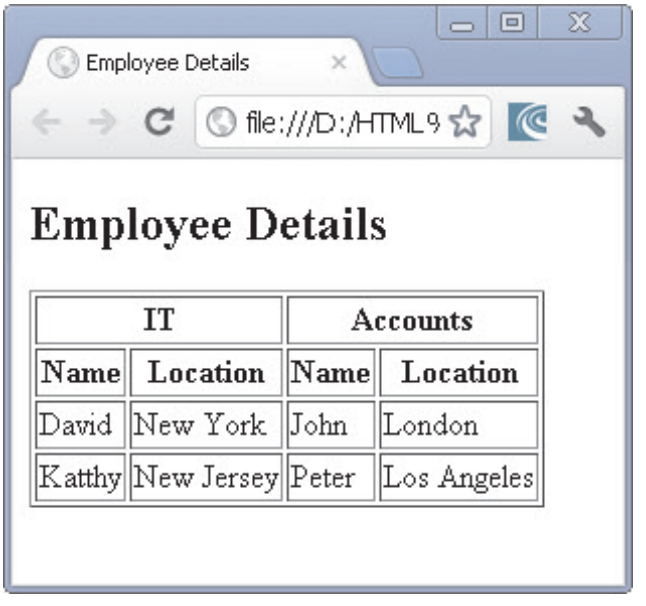



Figure 9.3: Employee Details

## *9.2.3 Rowspan Attribute*

The `rowspan` attribute spans a data cell across two or more rows. It allows the user to span a data cell along a vertical column. Like the `colspan` attribute, the `rowspan` attribute can be used within the `<td>` and `<th>` elements. Code Snippet 4 demonstrates how to span a cell across multiple rows.

**Code Snippet 4:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Automobile Gallery</title>
  </head>
  <body>
    <table border="1">
      <tr>
      <th>Manufacturer</th>
      <th>Model</th>
      <th>Price</th>
```

```
</tr>
    <tr>
      <th rowspan="3">Audi</th>
       <td>A4</td>
      <td>34.5</td>
    </tr>
    <tr>
      <td>A5</td>
      <td>42.6</td>
    </tr>
    <tr>
      <td>A6</td>
      <td>30.75</td>
    </tr>
    <tr>
      <th rowspan="2">BMW</th>
      <td>328i</td>
      <td>28.25</td>
    </tr>
    <tr>
      <td>530d</td>
      <td>47.5</td>
    </tr>
  </table>
</body>
</html>
```

The code creates a table with a border width of 1 pixel. The three `<th>` elements within the `<tr>` element specify column headings namely, **Manufacturer**, **Model**, and **Price.** The `rowspan` attribute of the `<th>` element combines the three rows of the **Manufacturer** column into a common brand namely **Audi**. The three different models and the respective prices of the **Audi** brand are displayed in three different rows. Similarly, the `rowspan` attribute of the `<th>` element combines the next two rows of the **Manufacturer** column into a common brand called **BMW**.

Figure 9.4 displays the `rowspan` attribute effect.

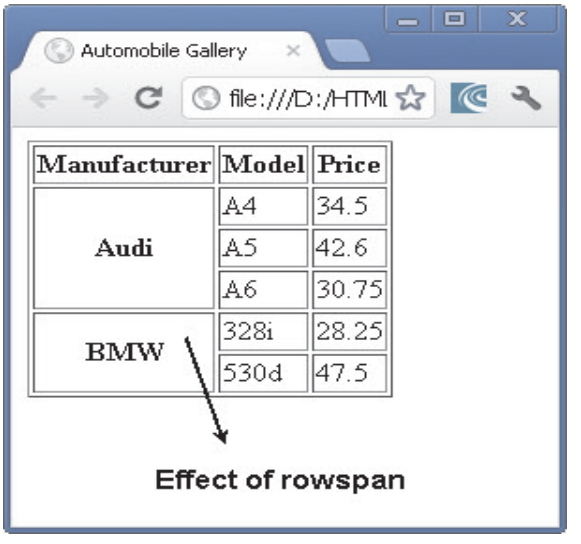



**Figure 9.4: rowspan Attribute Effect**

## *9.2.4 Horizontal Alignment*

Alignment determines the representation of text along the left, right, or center positions. In HTML, by default, the data within the table is aligned on the left side of the cell. Sometimes, the user might need to align the data to some other position for improving the readability or focusing on some data. HTML5 has deprecated the `align` attribute.

The four possible values for setting the horizontal alignment are as follows:

- **left**

  Aligns the data within a cell on the left side. This is the default value for table content.

- **center**

  Aligns the data within the cell on the center. This is the default value for table headings.

- **right**

  Aligns the data within the cell on the right side.

- **justify**

  Aligns the data within the cell by adjusting the text at the edges.

To set the alignment with `style` you can use the `text-align` attribute to specify the horizontal alignment.

Code Snippet 5 demonstrates how to center align the table data.

**Code Snippet 5:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Automobile Gallery</title>
  </head>
  <body>
    <table border="1">
      <tr>
      <th>Sr.No.</th>
      <th>Medicine Name</th>
      <th>Price</th>
    </tr>
    <tr style="text-align: center;">
      <td>1</td>
      <td>Captopril</td>
      <td>12.45</td>
    </tr>
    <tr style="text-align: center;">
      <td>2</td>
      <td>Ceftriaxone</td>
      <td>6.94</td>
    </tr>
    <tr style="text-align: center;">
      <td>3</td>
      <td>Ciprofloxacin</td>
      <td>56.21</td>
    </tr>
  </table>
  </body>
</html>
```

The code aligns the data within the row using a style within the `<tr>` element. The table content is center aligned by setting the value of the `text-align` attribute to center.

Figure 9.5 displays the horizontal alignment.

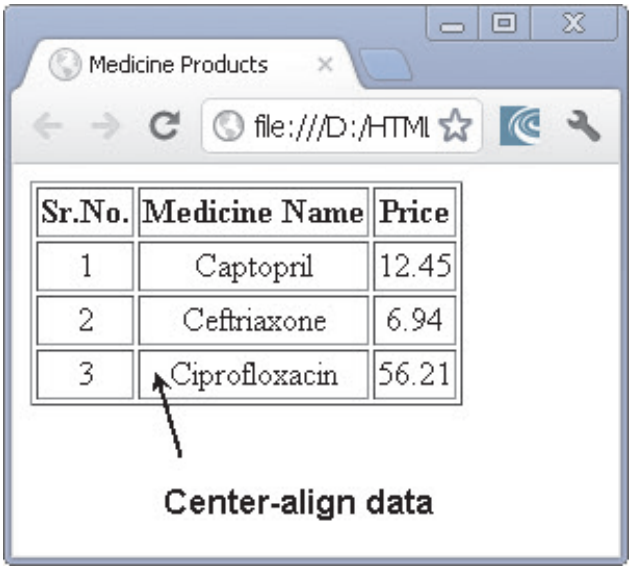



Figure 9.5: Horizontal Alignment

## *9.2.5 Vertical Alignment*

Users can vertically align the position of data earlier by using the `valign` attribute. HTML5 has deprecated the `valign` attribute. The possible values of vertical alignment are as follows:

➔ **top**

Vertically aligns the data within the cell at the top.

➔ **middle**

Vertically aligns the data within the cell at the center.

➔ **bottom**

Vertically aligns the data within the cell at the bottom.

To set the alignment with the style you can use the `text-align` attribute to specify the vertical alignment use the following syntax:

**Syntax:**

```
<td style= "text align: center; vertical align: middle">
```

The style can also be applied to individual rows, cells, or to the entire table.

Code Snippet 6 demonstrates how to align the data vertically within the table using the `style` attribute.

**Code Snippet 6:**

```
<!DOCTYPE HTML>
 <html>
  <head>
   <title>CelinaBatteries</title>
  </head>
   <body>
     <table border="1">
      <tr>
        <th>Sr.No.</th>
        <th>Product Id</th>
        <th>Product Description</th>
      </tr>
      <tr>
              <td  style="text-align:  center;  vertical-align:  middle">1
        </td>
            <td  style="text-align:  center;  vertical-align:  middle">P101
      </td>
       <td>1.5 Volts AA Ultra Alkaline</td>
      </tr>
      <tr>
              <td  style="text-align:  center;  vertical-align:  middle">2
        </td>
            <td  style="text-align:  center;  vertical-align:  middle">M105
      </td>
       <td>9 Volts pp3 Super Alkaline</td>
      </tr>
     </table>
   </body>
 </html>
```

The `text-align` attribute is set to the value `center`, which specifies that the data within the rows are centrally aligned. The `vertical-align` is used to specify the vertical alignment in the table.

Figure 9.6 displays the vertical alignment.

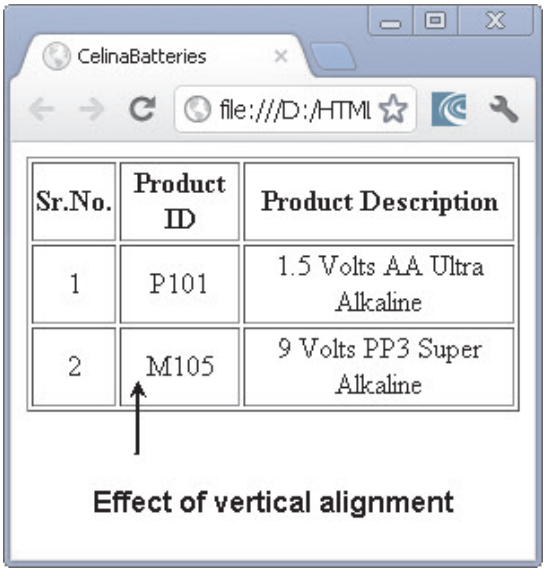



**Figure 9.6: Vertical Alignment**

## *9.2.6 Margin Attributes*

The data in a table might appear cluttered, which may affect the readability. This might make it difficult to comprehend data as the data. To overcome this issue, use the cell margin attributes.

Cell padding allows the user to control the look of the content on a page.

➔ **Padding**: Padding is the amount of space between the content and its outer edge. For tables, padding is specified as a space between the text and the cell border. Suppose if the user wants to set the padding attribute for the individual cells then he/she can use the padding attribute in a style as follows:

```
<td style="padding: 4px">
```

## *9.2.7 Caption Element*

The user can add a heading to a table in HTML. To specify the main heading for the table, use the `<caption>` element. The `<caption>` element defines a caption for the table. It is a sub-element of the `<table>` element. It must be present immediately after the `<table>` tag.

Unlike the `<th>` element that is used to specify a heading to an individual row or column, the `<caption>` element allows the user to specify a title for your entire table. There can be only one caption for a table.

Code Snippet 7 demonstrates how to specify a heading for a table.

**Code Snippet 7:**

```
<!DOCTYPE HTML>
 <html>
  <head>
    <title>Travel Expense Report</title>
  </head>
    <body>
       <table border="1">
          <caption>Travel Expense Report</caption>
            <tr>
               <th> </th>
               <th>Meals</th>
               <th>Hotels</th>
               <th>Transport</th>
            </tr>
            <tr>
                <td>25-Apr</td>
                <td>37.74</td>
                <td>112.00</td>
                <td>45.00</td>
            </tr>
            <tr>
               <td>26-Apr</td>
               <td>27.28</td>
               <td>112.00</td>
               <td>45.00</td>
            </tr>
            <tr>
               <td>Totals</td>
               <td>65.02</td>
               <td>224.00</td>
               <td>90.00</td>
            </tr>
       </table>
    </body>
 </html>
```

The code creates a table of border width of 1 pixel. The `<caption>` element that is used inside the `<table>` element specifies a caption to the entire table as Travel Expense Report.

Figure 9.7 displays the table captions.

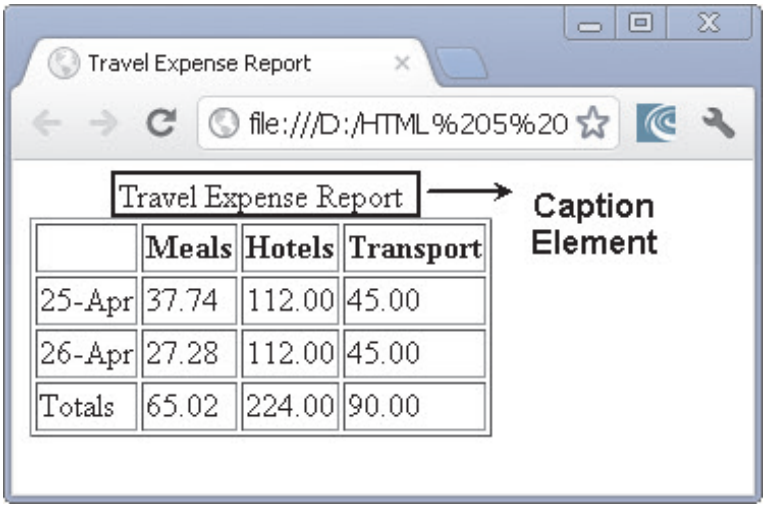



**Figure 9.7: Table Captions**

## *9.3 Table Size and Width of a Column*

The user can decide the size of the table based on his/her requirements while creating a Web site. The table size can be expanded when the user wants to add rows and columns in the table. The user can use the `<style>` section to set the default width for the table to 100% of the browser window.

For setting the width of a column in pixels, you can use style attribute in the `<td>` tag.

Code Snippet 8 demonstrates how to create a table with specific width for a column.

**Code Snippet 8:**

```
<!DOCTYPE HTML>
 <html>
  <head>
   <title>Tables</title>
  </head>
    <body>
     <h2>Table</h2>
      <table border="1">
      <tr>
         <td style ="width: 200px">Flowers</td>
```

```
      <td style ="width: 80px">Fruits</td>
       </tr>
       <tr>
         <td style ="width: 200px">Vegetables</td>
         <td style ="width: 80px">Trees</td>
       </tr>
      </table>
    </body>
</html>
```

The code creates a table of `border` width of 1 pixel. The `<style>` element is used to set table width to `100%`. The width of the columns is set by using the `style` attribute. Figure 9.8 displays the table size and column width.

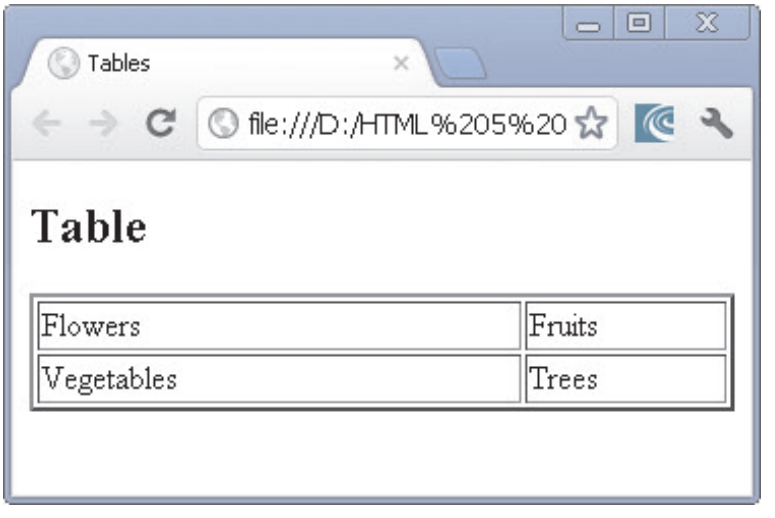



**Figure 9.8: Table Size and Column Width**

## *9.4 Merging Table Cells*

Suppose if the user wants to change the cells of a table to different height and width then colspan and rowspan attributes can be used. Consider a scenario, where the user wants to merge a cell into adjacent cells to the right-handside. The `colspan` attribute can be used to specify the number of columns to span. Similarly, the user can use the `rowspan` attribute to specify the number of rows.

Code Snippet 9 demonstrates creating a table having five columns and five rows, but many of the cells span multiple columns or rows.

**Code Snippet 9:**

```
<!DOCTYPE HTML >
 <html>
  <head>
   <title>Favorite Destination</title>
  </head>
    <body>
     <h2>Report</h2>
      <table border="1" width="100%" height="100%">
       <tr>
         <td colspan="2" rowspan="2">Results</td>
         <td colspan="3">Range</td>
       </tr>
       <tr>
         <td>18 to 20</td>
         <td>25 to 50</td>
         <td>over 50</td>
       </tr>
       <tr>
         <td rowspan="3">Your favorite vacation destination</td>
        <td>Dubai</td>
        <td>25%</td>
        <td>50%</td>
        <td>25%</td>
       </tr>
       <tr>
        <td>Bangkok</td>
        <td>40%</td>
        <td>30%</td>
        <td>30%</td>
       </tr>
       <tr>
        <td>Switzerland</td>
        <td>30%</td>
        <td>20%</td>
        <td>50%</td>
       </tr>
      </table>
    </body>
 </html>
```

The code creates a table having a `border` of 1 pixel. It also creates a table with five columns and five rows and uses the `colspan` and `rowspan` attributes respectively.

Figure 9.9 displays the merging table cells.

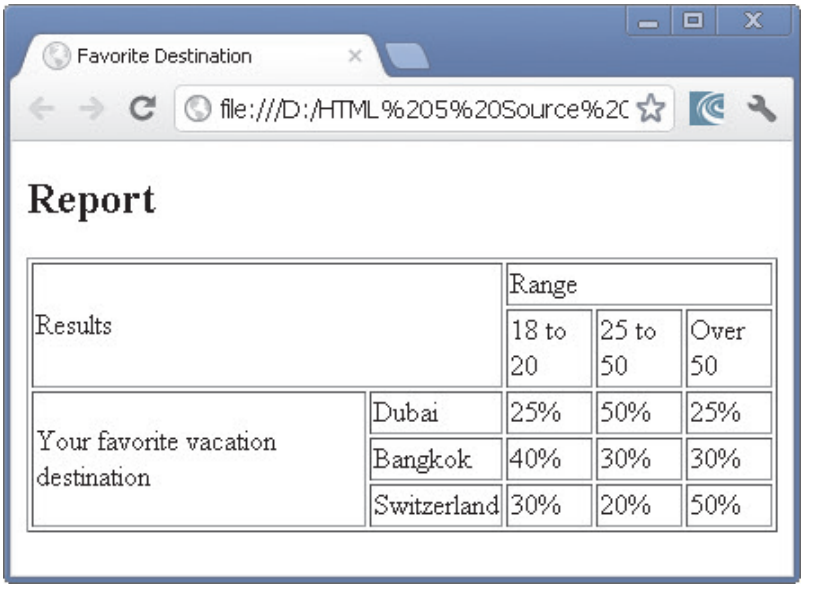



**Figure 9.9: Merging Table Cells**

## *9.5 Apply Borders by Using Styles*

Users can use CSS for applying borders as it is the best reliable and flexible method. The user must select the CSS method for Web sites that will be active for many years as the old formatting methods will not be used in future. You can format the table by using style based border for `<table>` and `<td>` tags. To evaluate the attributes used are as follows:

- The `border-width` attribute is used to control the thickness of the border and the values are specified in pixels.
- The `border-color` attribute is used to control the color of the border and specifies the color by either name, or RGB value, or hexadecimal number.
- The `border-style` attribute is used to control the line style. Users can choose between solid, dashed, groove, dotted, outset, ridge, inset, or none.

Suppose if the user wants to set all these attributes at one time then the user can use the `border` attribute and place the settings in the following order namely, `width`, `color`, and `style` respectively. The user can also format the sides of the border individually by replacing the `border` attribute with `border-bottom`, `border-top`, `border-right`, or `border-left` attribute. The user can apply these attributes to the entire table or individual cells and also create rules in the `<style>` area.

## *9.6 Tables for Page Layout*

Nowadays, there are many new techniques used for developing attractive Web pages. Tables are used for structuring the content. In other words, tables are used by the user to organize the data in an appropriate manner.

With tables the user can arrange the data horizontally or vertically according to his/her requirements. Community Web sites such as Facebook has different page layouts, the user uses the navigation tabs to move from one page to another. Similarly, the look and feel of each page is different.

While accessing Web sites such as Yahoo, Rediff, and so on users can view that the home page is very informative with a number of links, images, and so on. Each and every Web site has their unique way of presenting data to their customers or users. Many Web sites use pop-ups for providing information to their customers.

Code Snippet 10 demonstrates a simple example of using table for structuring the content of a Web page.

**Code Snippet 10:**

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Page Layout </title>
  </head>
  <style>
   #navlayout {
      width: 100%;
      float: left;
      margin: 0 0 3em 0;
      padding: 0;
      list-style: none;
      background-color: #f2f2f2;
      border-bottom: 1px solid #ccc;
      border-top: 1px solid #ccc; }
   #navlayout li {
      float: left; }
   #navlayout li a {
     display: block;
     padding: 8px 15px;
```

```
     text-decoration: none;
      font-weight: bold;
      color: #069;
      border-right: 1px solid #ccc; }
   #navlayout li a:hover {
      color: #c00;
      background-color: #fff; }
  </style>
 <body>
  <img src="../Images/flowers.jpg" width="133"
   height="100"  alt=""
    border="0">
    <h1>Blossoms Gallery</h1>
    <h5><i>The Best sellers for flowers since 1979</i></h5>
     <navlayout>
      <hr>
       <ul id="navlayout">
        <li><a href="#">Home</a></li>
       <li><a href="#">Contact Us</a></li>
       <li><a href="#">About Us</a></li>
       <li><a href="#"> FAQs</a></li>
       </ul>
     </navlayout>
      <table>
       <tr>
         <td>
          <b>Flowers are now in stock! </b>
          <i> We have just received a large shipment of flowers
              with prices as low as $19.
          </i>
         </td>
       </tr>
      </table>
 </body>
</html>
```

The code creates a page layout for a Web site. The data is arranged in a tabular format and an embedded style is used for defining the style. The style is defined using the `style` element placed immediately after the `<head>` section.

Defining a style in this manner helps to reuse the style in the same Web page.

The style is set using the ID selector methodology and is identified as `navlayout`. This will enable to apply the style to the content of all those elements whose id attribute has been set to `navlayout`.

Figure 9.10 displays the example of a page layout for using tables.

Figure 9.10: Page Layout for Tables

## *9.7 Check Your Progress*

1. Which of the following statements are true for tables?

| | | | |
|---|---|---|---|
| (A) | A table is made up of rows and columns | (C) | A row is made up of a set of cells that are placed horizontally |
| (B) | The intersection of each row and column is called as a cell | (D) | A column is not made up of a set of cells that are placed vertically |

2. Which of the following element is used to specify the heading for columns in a table?

| | | | |
|---|---|---|---|
| (A) | `<style>` | (C) | `<tr>` |
| (B) | `<td>` | (D) | `<th>` |

3. Identify the attribute that allows spanning of cell horizontally.

| | | | |
|---|---|---|---|
| (A) | rowspan | (C) | meta |
| (B) | colspan | (D) | div |

4. Which of the following option represent the possible values that can be used to vertically align the content within a table?

| | | | |
|---|---|---|---|
| (A) | top | (C) | middle |
| (B) | justify | (D) | bottom |

5. Andy is working on an online banking Web site. He wants to create a table for displaying the rate of interest allowed by the bank for different types of accounts. He also wants to give some width for the columns in the table. Which of the following code can he use to accomplish this task?

| | |
|---|---|
| (A) | `<table border=”1”>`<br>`<tr>`<br>`<td style=”height: 20px”>Savings</td>`<br>`<td style=”height: 80px”>Current</td>`<br>`</tr>`<br>`<tr>`<br>`<td style=”height:30px”>4%</td>`<br>`<td style=”height: 70px”>10%</td>`<br>`</tr>`<br>`</table>` |
| (B) | `<table border=”1”>`<br>`<tr>`<br>`<td style =”20px”>Savings</td>`<br>`<td style =”80px”>Current</td>`<br>`</tr>`<br>`<tr>`<br>`<td style =”30px”>4%</td>`<br>`<td style =”70%”>10px</td>`<br>`</tr>`<br>`</table>` |
| (C) | `<table border=”1”>`<br>`<tr>`<br>`<td style =”width: 100px”>Savings</td>`<br>`<td style =”width: 200px”>Current</td>`<br>`</tr>`<br>`<tr>`<br>`<td style =”width: 100px”>4%</td>`<br>`<td style =”width: 200px”>10%</td>`<br>`</tr>`<br>`</table>` |

(D)

```
<table border="1">
<tr>
<td style ="20px">Savings</td>
<td style ="80px">Current</td>
</tr>
<tr>
<td style ="70px">4%</td>
<td style ="30px">10%</td>
</tr>
</table>
```

## *9.7.1 Answers*

| | |
|---|---|
| 1. | A, B, C |
| 2. | D |
| 3. | B |
| 4. | A, C, D |
| 5. | C |

# Summary

- Tables allow the user to view your data in a structured and classified format.
- Padding is the amount of space between the content and its outer edge.
- The caption element defines a caption for a table. It is a sub-element of the <table> element.
- Spanning refers to a process of extending a cell across multiple rows or columns.
- The rowspan attribute spans a data cell across two or more rows.
- The colspan attribute allows the user to specify the number of columns a cell should span.
- The border attribute of the table element allows the user to specify a border for making the table visible in a Web page.
- Tables allow the user to organize the data. It enables the developer to design a Web page having an attractive page layout.

# *Try it Yourself*

1. Samson works for an advertising agency and is headquartered at Hong Kong. He is very fond of learning latest technologies that are coming up in the market. He wants to create an HTML5 Web site for his company. The company offers a number of services such as designing of logos, new media, promotions, and branding. Samson wants to develop their company Web site home page and also wants to add all the details about the company with additional graphics, text, animations, and arrange the content in an organized manner. He has decided to use tables for the Home page. Help him to develop the application.

2. Carol is working on an online shopping Web site. This Web site has a wide range of collection for men, women, and kids. The Web site deals with many branded products such as bags, shoes, clothes, jewelry, beauty products, and many more. The Web site was created earlier using HTML 4 version. Now, she has decided to upgrade the Web site with HTML5. She wants to change the appearance and layout of the entire Web site by adding page layouts to the Web site using the new features of HTML5 so that the Web site is appealing and attractive for the new generation. Help her to design the Web site using new features of HTML5.

# Onlinevarsity

# Session - 10

## HTML Forms

Welcome to the Session, **HTML Forms**.

This session describes the new features in HTML5 forms, working of new input types, new form attributes, new form field attributes, and new form elements.

In this Session, you will learn to:

- Describe HTML5 forms
- Explain the working of new input types in HTML5
- Explain the new Form attributes
- Explain the new Form elements

## *10.1 Introduction*

HTML, the core language of the World Wide Web is being used for designing Web pages for nearly a decade. With the continuous evolvement in the Web, Web designers and programmers find it difficult to make pages more flexible and dynamic in behavior. The dynamic behavior is particularly expected when displaying Web forms to the users. For example, HTML5 allows creating `combo boxes`, adding `placeholder` text on the page, providing client-side validations through JavaScript, and achieving dynamic behavior using Document Object Model (DOM) API.

Although Internet Explorer 9 and Google Chrome 19 support most HTML5 features, there are some that are supported only in Opera 11.

## *10.2 Introduction to HTML5 Forms*

Among the other features of HTML5, there has been a great enhancement to Web forms. HTML5 Web forms are those sections on the Web page that contain special elements called as controls. The controls, such as `check boxes`, `radio buttons`, and `text boxes` provide a visual interface to the user to interact with them. A user provides data through these controls that is sent to the server for further processing. Thus, it is very important to create forms with lot of consideration for usability and accessibility by the users.

In HTML5, creation of form is made easier for Web developers by standardizing them with rich form controls. It also provides client-side validations that are now handled natively by the browsers. This not only reduces the load time of the pages, but also removes the need of the repetitive JavaScript codes to be included on the page.

Even the visual appearance of the forms is improved when displayed on different devices, such as iPhone, ipad, touch screens, and browsers. This enhances the users experience while interacting with them.

## *10.3 New Features in HTML5 Forms*

HTML5 Web forms bring great improvements related to form creation for the Web developers and also for users interacting with them. The following are the changes introduced in HTML5 forms:

- New form elements
- New input types
- New attributes
- Browser-based validation
- CSS3 styling techniques
- Forms API

**Note** - The state of HTML5 is changing continuously to improve. Thus, support for new form elements, attributes, and input types can vary across different browsers.

## *10.3.1 New Elements*

HTML5 has introduced a range of new elements that are expanding the options for more number of elements related to input on the forms.

Table 10.1 lists the new elements in HTML5.

| Element | Description |
|---|---|
| `progress` | Represents the completion progress of a task on the page |
| `meter` | Represents a scale of known range |
| `datalist` | Represents a set of options used with list attribute to make a drop-down control |
| `output` | Represents the result of a calculation |

**Table 10.1: New Elements in HTML5**

## *10.3.2 New Input Types*

The `input` element is a data field that allows the user to edit the data on the form. It has an attribute named `type` which controls the data type and characteristics of the `input` element.

Table 10.2 lists the new input types supported by HTML5 that specify the kind of input expected from the users on the Web page.

| Type | Description |
|---|---|
| `email` | Represents the completion progress of a task on the page |
| `search` | Represents a scale of known range |
| `url` | Represents a set of options used with list attribute to make a drop-down control |
| `tel` | Represents the result of a calculation |
| `number` | Represents a numeric value in the input field |
| `range` | Represents a numeric value to be selected from a range of numbers |
| `date` | Represents a calendar which is shown at every click on the field |
| `week` | Represents date in year-week format |
| `month` | Represents a value with year-month format |
| `time` | Represents a value in hours and minutes format |
| `datetime` | Represents a full date and time input field with a time zone |
| `datetime-local` | Represents a full date and time with no time zone |

| Type | Description |
|---|---|
| `color` | Represents a predefined interface for selecting color |

**Table 10.2: New Input Types Supported by HTML5**

## *10.3.3 New Attributes*

HTML5 has introduced several new attributes that can be used with `form` and `input` elements. Attributes help the elements to perform their tasks.

Table 10.3 lists the new attributes in HTML5.

| Type | Description |
|---|---|
| `placeholder` | Represents a hint that help users to enter the correct data in the field |
| `required` | A Boolean attribute that validates the entry in the field |
| `multiple` | A Boolean attribute that allows multiple values to be entered in the field |
| `autofocus` | Focuses the input element on page load |
| `pattern` | Represents a regular expression for validating the field's value |
| `form` | Allows the elements to reference the form by including the form name |

**Table 10.3: New Attributes in HTML5**

## *10.3.4 Browser-based Validation*

HTML4 supported the use of custom JavaScript or libraries to perform validation on the client-side browsers. These validations ensure that the input fields are checked before the form is submitted to the server for further processing.

The new attributes in HTML5, such as `required` and `pattern` can be used with the input elements to perform validation. This relieves the Web developers from writing the custom JavaScript code for performing client-side validation on the Web pages. HTML5 also provides advanced validation techniques that can be used with JavaScript to set custom validation rules and messages for the input elements.

## *10.3.5 CSS Styling Techniques*

A Web developer can enhance the form elements with the pseudo-class selectors, such as `:required`, `:valid`, and `:invalid`.

For example, the input fields which cannot be left blank while submitting the form can be displayed with an outline. To achieve this, input field with required attribute can be styled using CSS. Applying CSS styles make it easier for user to navigate and complete the form.

Code Snippet 1 shows the CSS code for formatting non-empty and incorrect data input in the `input` element fields on the form.

**Code Snippet 1**:

```
<style>
 input:required {
 outline: 1px red solid;
 color: green ;
 }
 input:required:valid {
 background-size:10px 10px;
 background-position: right top;
 background-repeat: no-repeat;
 }
 input:required:invalid {
 background-size:10px 10px;
 background-position: right top;
 background-repeat: no-repeat;
 }
</style>
</head>
<body>
 <form method="get" action="try.php">
  Name: <input type="text" name="name" required="true" /><br/>
  Email: <input type="email" name="emailid" required="true" />
         <input type="submit" value="submit" />
 </form>
```

## *10.3.6 Forms API*

HTML5 has introduced JavaScript API for forms. This API is used to customize validations and processing performed on the forms. The new form's API provides new methods, events, and properties to perform complex validations combining fields or calculations.

Table 10.4 lists the events and methods.

| Events and Methods | Description |
|---|---|
| `setCustomValidity (message)` | Sets the custom error message that is displayed when the form is submitted by the user |
| `checkValidity()` | Checks the validity of the e-mail address entered by the user |
| `oninvalid` | Allows script to run only when the element is invalid |
| `onforminput` | Allows script to run when the form gets an input from the user |
| `onformchange` | Represents a regular expression for validating the field's value |
| `form` | Allows script to run when the form changes |

**Table 10.4: Events and Methods**

## *10.4 Working with New Input Types*

The `type` attribute of the `input` element determines what kind of input will be displayed on the user's browser. The default input is `type="text"`.

The registration form in the session is using the following input types:

- ➔ `text`
- ➔ `label`
- ➔ `radio`
- ➔ `textarea`
- ➔ `checkbox`
- ➔ `submit`

HTML5 has introduced more data-specific user interface elements. Now, you will see the new input types in detail.

## *10.4.1 E-mail Addresses*

The `type="email"` is used for specifying one or more e-mail addresses. To allow multiple addresses in the e-mail field, separate each address with comma-separator.

In the registration form, the input type is changed from `text` to `email` as shown in Code Snippet 2.

**Code Snippet 2**:

```
<form method="get" action="test.html">
   <label for="emailid">Email:</label>
   <input type="email" value="" id="emailid" name="emailaddress"
       maxlength="255"  />
   <input type="submit" value="submit"/>
</form>
```

In the code, `<label>` tag defines a label for the element on the form. The `for` attribute of `<label>` tag binds it with the related element, that is `email` element, on the form. The value of `for` attribute must match with the value of `id` attribute assigned to the element.

Also, the `email` contains two attributes, namely `id` and `name`. The `id` attribute specifies a unique id for the element. The value of the `id` attribute should be unique within the document. It can be used as a reference for styles in style sheet or to access elements using DOM API in JavaScript.

The `name` attribute specifies a name for the `input` element. It can be used for referencing the elements in a JavaScript or form data after a form is submitted to the server for processing. The look of the input is still like a plain text field, but changes are applied behind the scenes. Browsers, such as Firefox, Chrome, and Opera will display a default error message if user submits the form with some unrecognizable contents.

Figure 10.1 shows the error message for incorrect e-mail address in Chrome.

Figure 10.1: Error Message for Incorrect E-mail Address in Chrome

**Note** - If the browser does not provide the support for new input types, such as email, then they are displayed as plain text field.

## *10.4.2 URL*

The `type="url"` input element is used for specifying a Web address. The look of the `url` field is a normal text field. Code Snippet 3 shows the code of `url` input type.

**Code Snippet 3**:

```
<label for="url">Enter your Web page address:</label>
 <input type="url" value="" id="urlname" name="urltext"
        maxlength="255"  />
 <input type="submit" value="submit"/>
```

Browsers, such as Opera, Firefox, and Chrome supports validation for the `url` input type. While validating the URL, browsers only checks for entry with forward slash `(/)`. For example, a URL such as **x://mysite.com** will be considered as valid, even though `x://` is not a real protocol.

Figure 10.2 shows the error message for incorrect URL in Chrome.

**Figure 10.2: Error Message for Incorrect URL in Chrome**

## *10.4.3 Telephone Number*

The `type="tel"` input element is used for accepting telephone numbers. As compared to `email` and `url` types, the `tel` type does not impose a particular pattern. It supports characters, numbers, and special characters except new lines or carriage returns. The reason for not imposing any pattern for `tel` type is that different countries support various lengths and punctuation in the phone numbers. Thus, there cannot be a standard format for them.

A user can enforce a pattern for `tel` input type by using `placeholder` or `pattern` attribute. A JavaScript code can also be provided for performing client-side validation on the `tel` input type.

Code snippet 4 shows the code for including `tel` input type on the registration form.

**Code Snippet 4**:

```
<label for="telno">Telephone Number:</label>
 <input type="tel" value="" id="telno" name="telephone_no"
   maxlength="10"  />
```

## *10.4.4 Number*

The input `type="number"` is used for accepting only number values. The `input` element displayed for `number` type is a spinner box. The user can either type a number or click the up or down arrow to select a `number` in the spinner box.

Code Snippet 5 shows the code for including number input type on the form.

**Code Snippet 5**:

```
<label for="stud_age">Age:</label>
  <input type="number" value="15" id="stud_age"
       name="studentage" min="15" max="45" step="1" />
  <input type="submit" value="submit"/>
```

In the code, the `number` input type has attributes, such as `min` and `max` to specify the minimum and maximum value for the input.

Figure 10.3 shows the `number` input type in Opera.

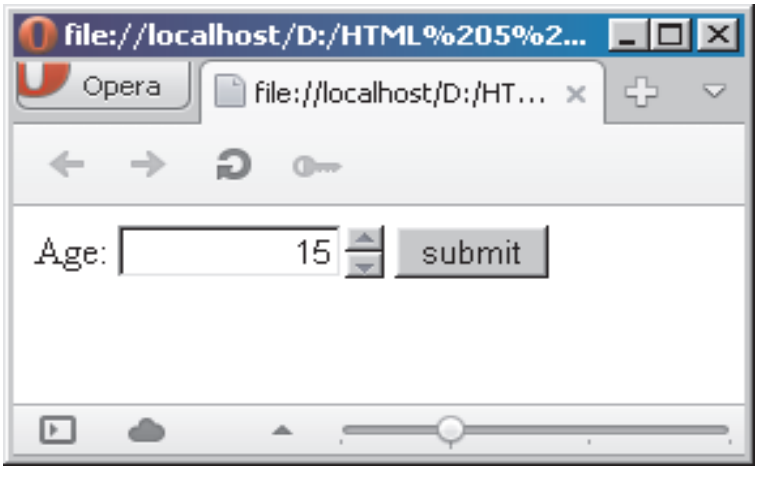



**Figure 10.3: Number Input Type in Opera**

## *10.4.5 Range*

The input `type="range"` is similar to `number` type and displays a slider control on the page. The range type is used when the exact value is not required in the input. In other words, the value from this type is not accurate. For example, an online survey form asking the clients to rate the products may not receive exact values in the ratings.

Code Snippet 6 shows the code for including range `input` type with attributes `min` and `max`.

**Code Snippet 6**:

```
<label>Survey for packages offered[scale: 1-10]:</label>
  <input type="range" name="rating" min="1" max="10" />
  <input type="submit" value="submit"/>
```

In the code, the range input type contains attributes, such as `min`, `max`, `step`, and value. The `min` and `max` attributes are used to specify the minimum and maximum value allowed for a range and are set to 1 and 10 respectively. The `step` attribute specify the intervals for incrementing the value. The value of step attribute is 1 by default. The `value` attribute specifies the default value for the range. The default value is the midpoint of the range specified.

Figure 10.4 shows the range input type in Opera.

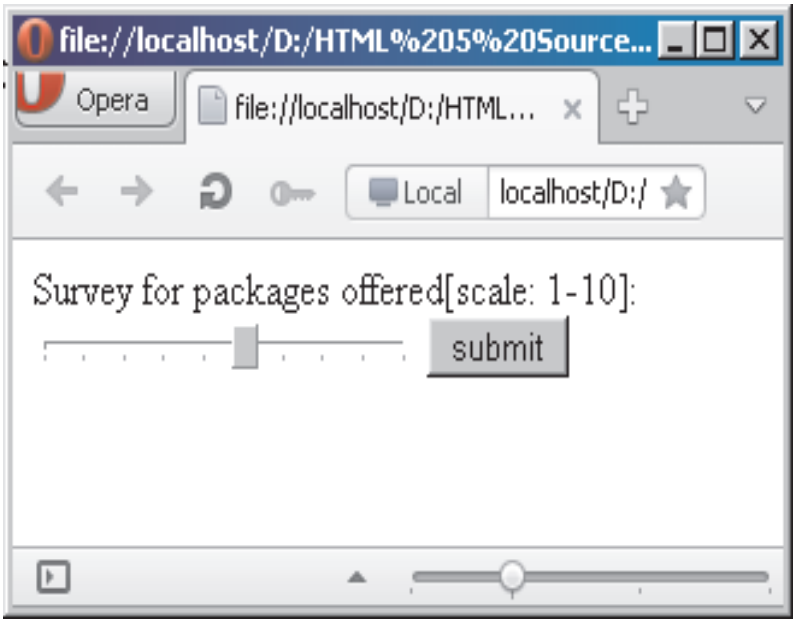



Figure 10.4: Range Input Type in Opera

Figure 10.5 shows the value for the range input type in the URL after the form is submitted by the user. The rating selected by the user can be seen in the Address Bar of the browser.

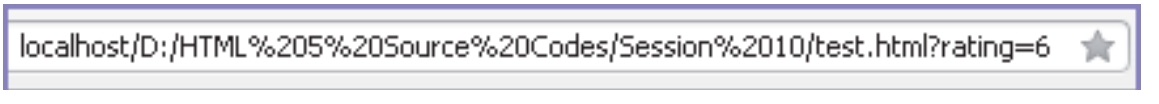



Figure 10.5: Value for the Range Input Type in URL

## *10.4.6 Date and Time*

HTML5 has introduced several new input types for date and time. The format for all these date and time types is according to the ISO standards. At present only Opera provides the support for date element by displaying a calendar control.

➔ **Date**

This input type contains only date in year, month, and day format. The time part is not support by date type.

Code Snippet 7 shows the code of the `date` input type.

**Code Snippet 7**:

```
<label for="stdate">Date:</label>
<input type="date" id="stdate" name="startdate"
   min="2000-01-01"/>
<input type="submit" value="Submit" id="btnSubmit"></input>
```

In the code, all `date` input types have `min` and `max` attributes to set the range for the dates. The default value for `date` input type is based on the browsers. Different browsers take different dates as default. Thus, it is advisable to set the minimum and maximum value for the `date` type.

Figure 10.6 shows the `date` input type.

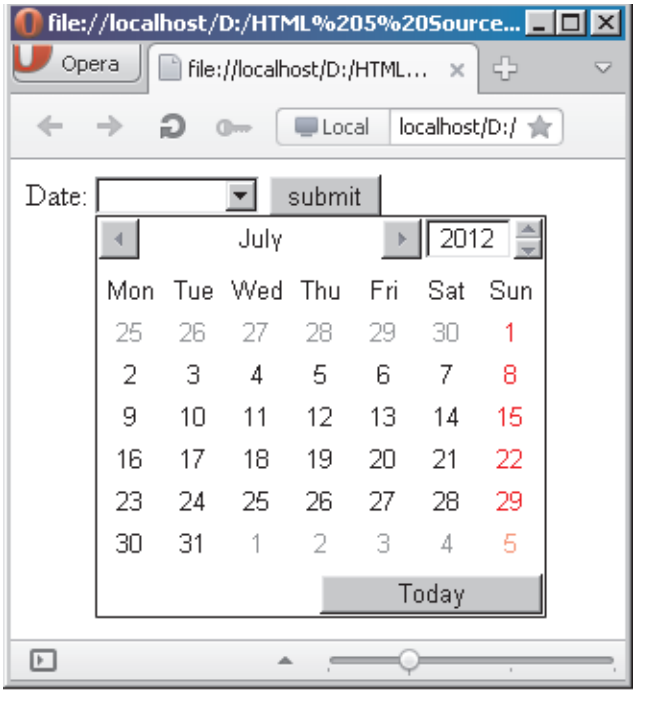



Figure 10.6: Date Input Type

Figure 10.7 shows the value sent for the `date` input type after the form is submitted by the user.

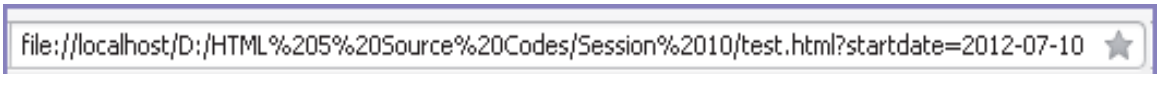



Figure 10.7: Value Sent for the Date Input Type

### ➔ Month

The `type="month"` includes only the year and month in the input.

Code Snippet 8 shows the syntax of `month` input type.

**Code Snippet 8**:

```
<label for="stmonth">Month:</label>
  <input type="month"  id="stmonth" name="startmonth" />
  <input type="submit" value="submit"/>
```

Browser such as Opera will display the date picker for selecting month. On selecting any day from the calendar, the whole month is selected.

Figure 10.8 shows the date picker for the `month` input type.

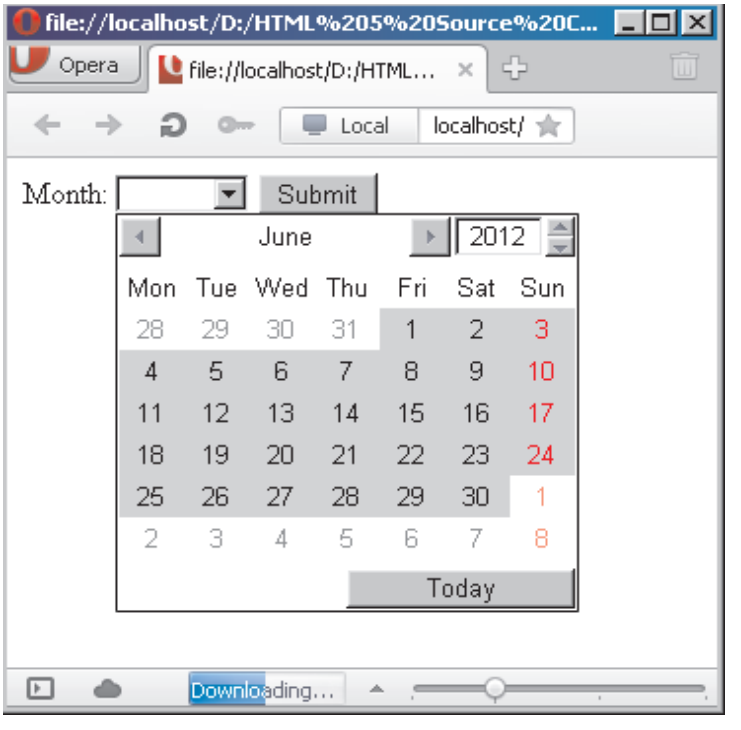



**Figure 10.8: Date Picker for the Month Input Type**

### ➔ Week

The input `type="week"` provides a similar interface as displayed for date type and selects the entire week.

Code Snippet 9 shows the syntax of the `week` input type.

**Code Snippet 9**:

```
<label>Week:</label>
  <input type="week" name="week" />
  <input type="submit" value="submit"/>
```

Figure 10.9 shows the `week` input type in Opera.

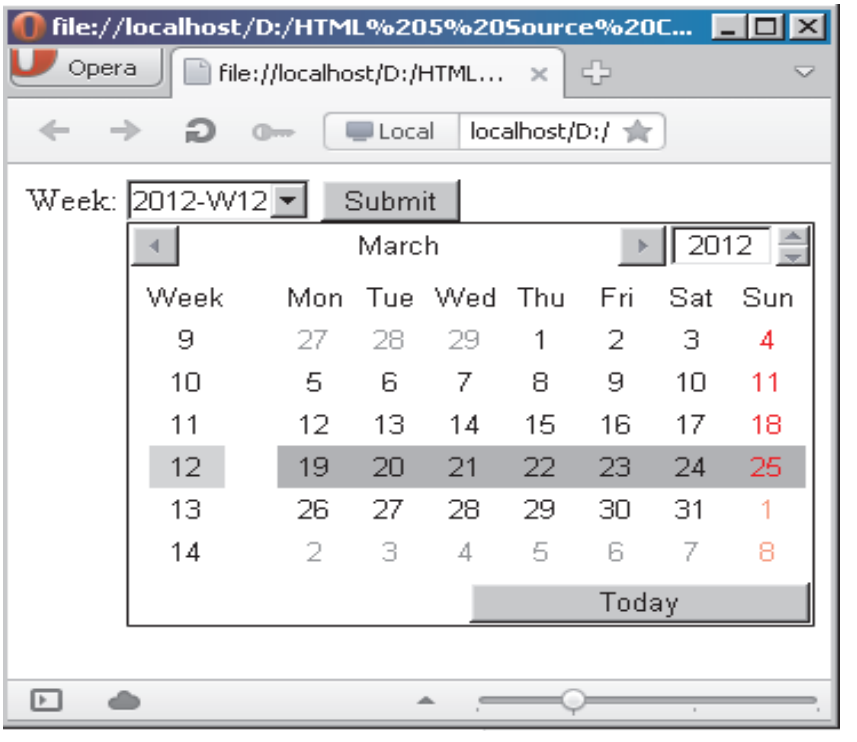



**Figure 10.9: Week Input Type in Opera**

In this figure, the 12th March is selected and the combo box also displays the 2012-W12 as the 12th week is selected. Similar to date input types, there are new time input types introduced in HTML5.

### ➔ Time

The input `type="time"` displays a time of day in hours and minutes format (24-hour clock).

Code Snippet 10 shows the syntax of `time` input type.

**Code Snippet 10**:

```
<label>Time:</label>
  <input type="time" name="time" />
  <input type="submit" value="submit"/>
```

Figure 10.10 shows the `time` input type in Opera.

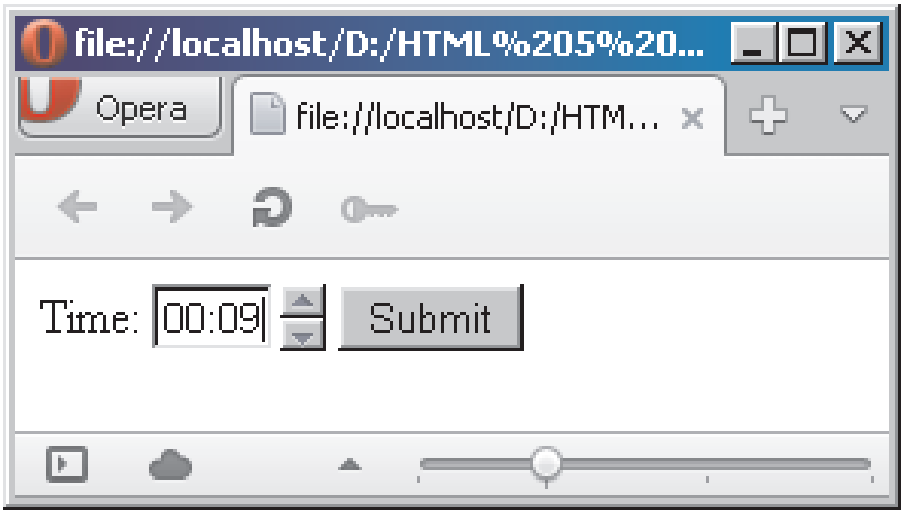



Figure 10.10: Time Input Type in Opera

➔ **Datetime**

The input `type="datetime"` includes full date and time in the input. The input includes a date part and a time part which is represented as Coordinated Universal Time (UTC). Thus, UTC time will be displayed with 'T' followed by a 'Z'. For example, an input, such as 2012-07-03T12%3A05Z is interpreted as 2012-07-03T12:01.

Code Snippet 11 shows the syntax of `datetime` input type.

**Code Snippet 11**:

```
<label for="mydatetime">Date-Time:</label>
  <input type="datetime" name="mydatetime" />
  <input type="submit" value="submit"/>
```

Figure 10.11 shows the `datetime` input type in Opera.

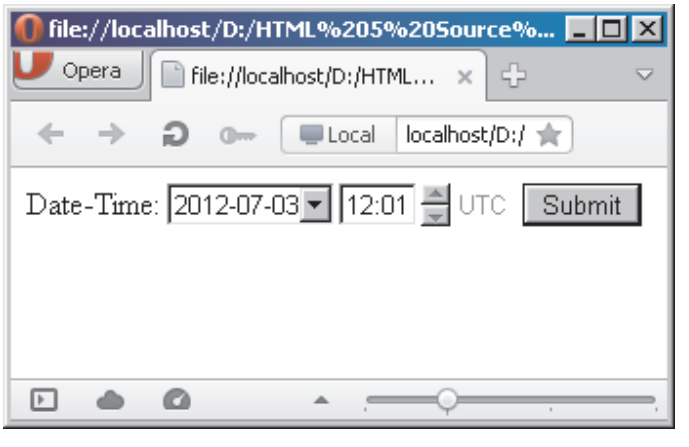



Figure 10.11: Datetime Input Type in Opera

➔ **Datetime-local**

This input type is similar to `datetime` input type, except that the time zone is omitted for input `type="datetime-local"`.

## *10.4.7 Color*

HTML5 provides a predefined interface for selecting the colors using input `type="color"`. The input value from the color input field is a hexadecimal number. For example, `#00FF00` represents a hexadecimal RGB color value.

At present, the color input type is supported on Opera browser and some new smart phones.

Code Snippet 12 shows the usage of `color` input type to display a color picker on the Web page.

**Code Snippet 12**:

```
<label>Color:</label>
<input type="color" name="mycolor" />
<input type="submit" value="submit"/>
```

Figure 10.12 shows the `color` input type in Opera.

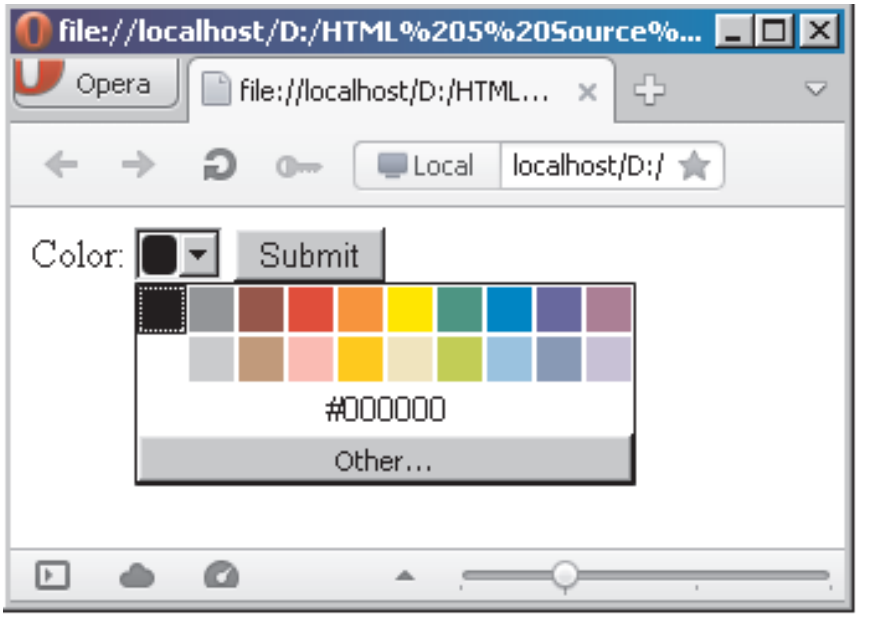



Figure 10.12: Color Input Type in Opera

## *10.5 New Form Attributes*

Earlier, Web developers needed to write JavaScript snippets for performing the validations on the data entered by the users in form fields. HTML5 has provided several new attributes that perform the validations without writing JavaScript snippets for them. These attributes perform the following tasks:

- ➔ Check data provided by users with the regular expression pattern assigned to the fields
- ➔ Inform users with appropriate errors
- ➔ Check that the required fields are not left empty by the users
- ➔ Enable multiple values for the fields, if provided

These attributes can be used to support scripting drawbacks, without actually hard coding them in the Web pages. Browsers that do not understand these new attributes will ignore them.

### *10.5.1 Required*

This is a `boolean` attribute that informs the browser to submit the form only when the required fields are not left empty by the users. The input type elements, such as `button`, `range`, and `color` cannot be set for `required` attribute as they have a default value.

Different Web browsers such as Opera, Chrome, and Firefox provide different error messages, such as 'This is a required field', or 'Please fill out this field' for required attribute.

Code Snippet 13 shows assignment of `required` attribute to the name field on the registration form.

**Code Snippet 13**:

```
<label>Name: <em> <img src="star.jpg" width="9" height="10"
             alt="" border="0"> </em>
</label> <br>
  <input type="text" value="" name="first" size="8" tabindex="1"
   required ="true"/>
  <input type="text" value="" name="last" size="14" tabindex="2"
   required="true"/>
  <input type="submit" value="submit"/>
```

Figure 10.13 shows the message of `required` attribute in Opera.

**Figure 10.13: Message of Required Attribute in Opera**

## *10.5.2 Placeholder*

This attribute displays a short hint or text inside a form element. This informs the user about what data needs to be entered in that field. The `placeholder` text toggles, which means it appears in the field and disappears when the user clicks inside the field. Earlier, Web developers provided this functionality through JavaScript snippets which is now done in the browsers with the help of `placeholder` attribute. At present, all the browsers such as Chrome, Safari, Opera, and Firefox support the `placeholder` attribute.

If the size of the hint exceeds the field size, then use `title` attribute to describe text for the field.

Code Snippet 14 shows the assignment of `placeholder` attribute to the name field on the registration form.

**Code Snippet 14**:

```
<label>Name: <img src="required_star.gif" height="10px"
 width="10px"/></label> <br>
 <input type="text" value="" name="first" size="8" tabindex="1"
  required="true" placeholder="First Name"/>
 <input type="text" value="" name="last" size="14" tabindex="2"
  required="true" placeholder="Last Name" /><br/>
```

Figure 10.14 shows the message of `placeholder` attribute in Opera.

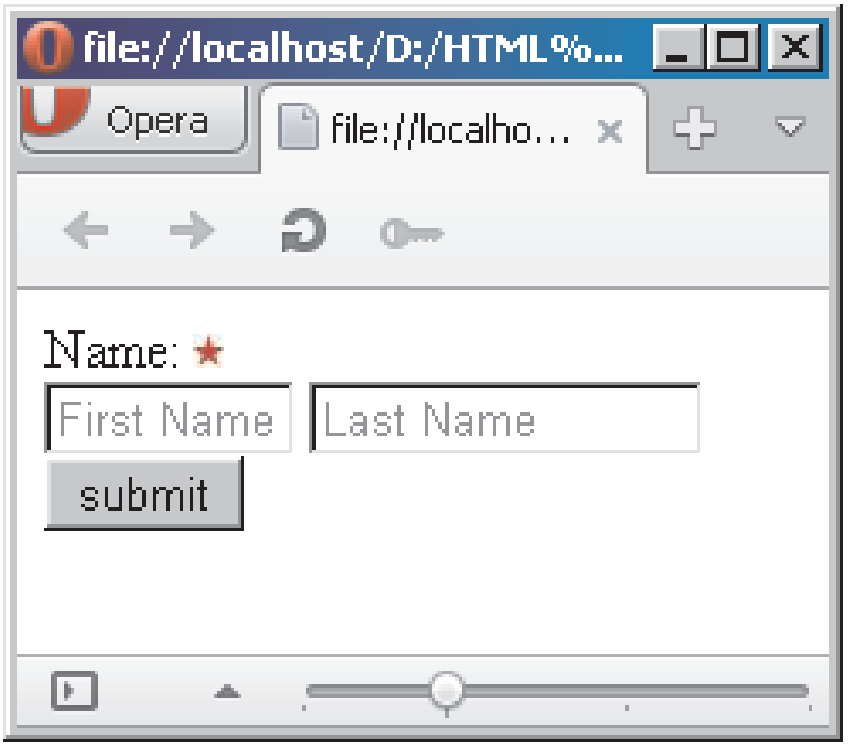



**Figure 10.14: Message of Placeholder Attribute in Opera**

## *10.5.3 Pattern*

This attribute uses regular expressions for validating the fields. The data entered by the user must match with the `pattern` specified in the regular expression. This helps to limit the input accepted from the user.

While including regular expressions through `pattern` attribute, it informs the users about the excepted `pattern` for the data. This can be achieved in the current browsers using the `title` attribute, which is displayed as a tool tip when the users move the pointer over the field.

You need to validate a zip code of five numbers on the form. There is no pre-defined input type to restrict the input to numbers of specified length. Thus, `pattern` attribute can be used to create user-defined check values for the field. Also, a `title` attribute can be used to customize the error message displayed for the field.

Code Snippet 15 shows the assignment of `pattern` attribute to the phone number field on the registration form.

**Code Snippet 15**:

```
 <label>Phone number:<img src="required_star.gif" height="10px"
   width="10px"/></label>
 <input type="tel" value="" size="4" maxlength="5" tabindex="11"
  required="true" placeholder="Code"pattern="[+0-9]{1,4}"
  title="Format:(+)99(99)"/>
 <label>-</label>
 <input type="tel" value="" size="10" maxlength="12"
    tabindex="13" required="true" placeholder="Number"
pattern="[0-9]{8,}"  title="Minimum 8 numbers"/>
```

In the code, `[+0-9]` pattern indicates that only special character `'+'` as well as numbers are allowed. Also, `{1, 4}` refers to the length of the numbers, that is between `1` and `4`. Similarly, `{8, }` means minimum eight numbers are allowed in the `tel` input type field.

Figure 10.15 shows the message of `pattern` attribute in Opera.

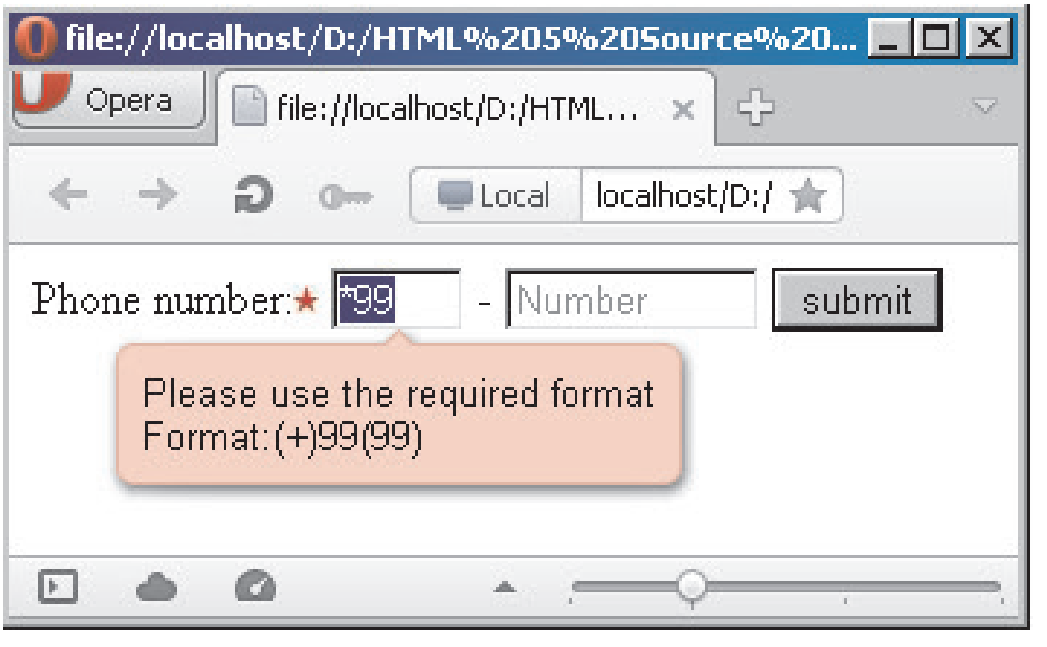



Figure 10.15: Message of Pattern Attribute in Opera

## *10.5.4 Multiple*

This is a `boolean` attribute that allows multiple values for some input types. This was available only for `select` input type in the earlier version of HTML.

HTML5 allows multiple attribute with input types, such as email and file. If assigned, it allows selection of multiple files, or include several e-mail addresses in the email field separated by comma separator.

At present, browsers such as Chrome, Opera, and Firefox support multiple attribute for email and file element.

Code Snippet 16 shows the assignment of multiple attribute to the e-mail address field on the registration form.

**Code Snippet 16**:

```
<label>Email Address:<img src="required_star.gif" height="10px" width="10px"/></label>

<input type="email" value="" name="emailid" maxlength="255" tabindex="5" required="true" placeholder="Email Address" multiple/>
```

In the code, multiple attribute will allow insertion of multiple e-mail addresses in the field. Every e-mail address will be validated individually by the browser.

Figure 10.16 shows the validation of multiple e-mail address.

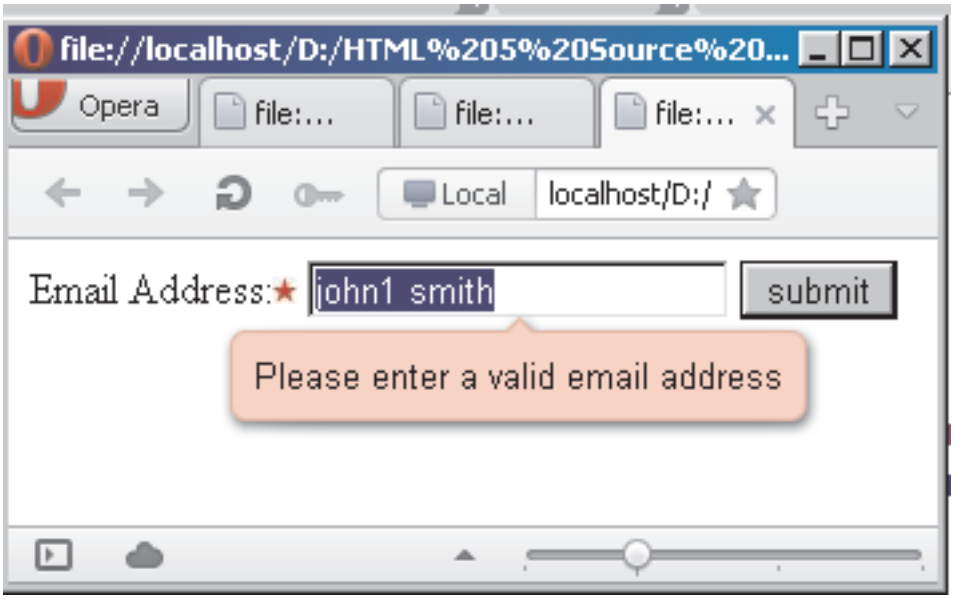



Figure 10.16: Validation of Multiple E-mail Address

## *10.5.5 Autofocus*

Earlier, Web developers were using JavaScript code to set the focus to the input field on page load. The purpose was to force the focus over the input field, even if the user selected some other element while page is still loading. As a result of the JavaScript code, control moves to the input field upon completion of page load. This way, regardless of what the user selected, the focus would always be on the input field.

To provide an easier solution for this behavior, HTML5 has introduced `autofocus` attribute for the form elements. The `autofocus` attribute will focus on the input field on page load. However, depending upon the situation, it will not move the focus away if the user has selected some other field. Only one element can be focused with `autofocus` attribute on a particular page while loading.

Code Snippet 17 shows the assignment of `autofocus` attribute to the first name field on the registration form.

**Code Snippet 17**:

```
<label>Name:</label>
 <br>
  <input type="text" value="" name="first" size="8" tabindex="1"
       placeholder ="First Name" autofocus/>
  <input type="submit" value="submit"/>
  <br>
  <label>First Name</label>
```

Figure 10.17 shows the behavior of `autofocus` attribute.

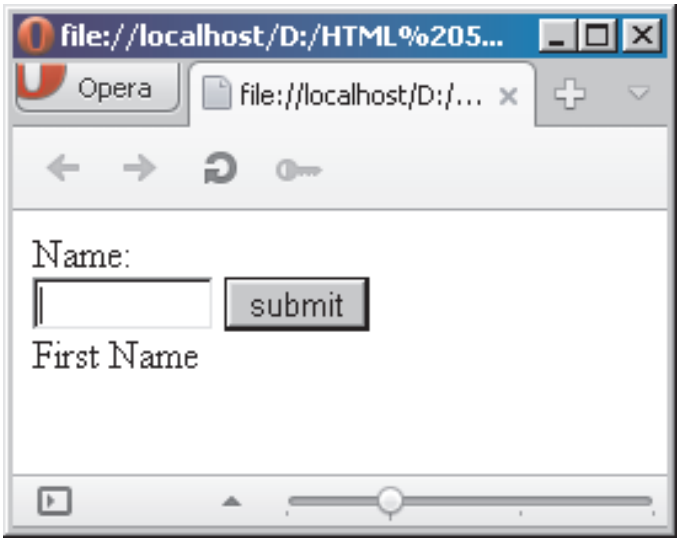



**Figure 10.17: Behavior of Autofocus Attribute**

## *10.5.6 Form*

Earlier, all the form controls need to be provided between the opening and closing `<form>` tag. In HTML5, elements can be inserted at any place in the document and they can reference the form using the `form` attribute.

Code Snippet 18 shows the association of an element with the form on the Web page.

**Code Snippet 18**:

```
<body>
 <input type="text" name="mytext" id="mytext" form="myform"/>
 . . .
 . . .
  <form id="myform">
 . . .
 . . .
  </form>
</body>
```

In the code, the form is declared with an `id` attribute. The value of the `id` attribute is assigned to the input element using `form` attribute.

## *10.5.7 Autocomplete Attribute*

Many browsers help user in filling forms by providing data in the input fields, such as `email` and `tel`, based on their earlier input. In many situations, the autocomplete behavior may not be secure, especially with certain fields accepting password or credit card number data.

HTML5 offers an `autocomplete` attribute which provides control on prefilled values displayed in the fields. It must be specified on the `form` element which applies for all input fields or on particular input fields. The `input` element that can support autocomplete are `text`, `url`, `tel`, `password`, `datepickers`, `range`, and `color`.

The `autocomplete` feature comprises two states namely, **on** and **off**. The **on** state indicates that the data that is not sensitive can be remembered by the browser. Similarly, the **off** state indicates that the data will not be remembered. Such data may be sensitive and not safe for storing with the browsers. Hence, user needs to explicitly provide the data each time while filling the form.

By default, many browsers have the `autocomplete` feature enabled in them. The browsers that do not support autocompletion, can be turned **on** or **off** for this behavior by specifying `autocomplete` attribute either on the form or specific input elements.

Figure 10.18 shows the behavior of `autocomplete` attribute in Chrome.

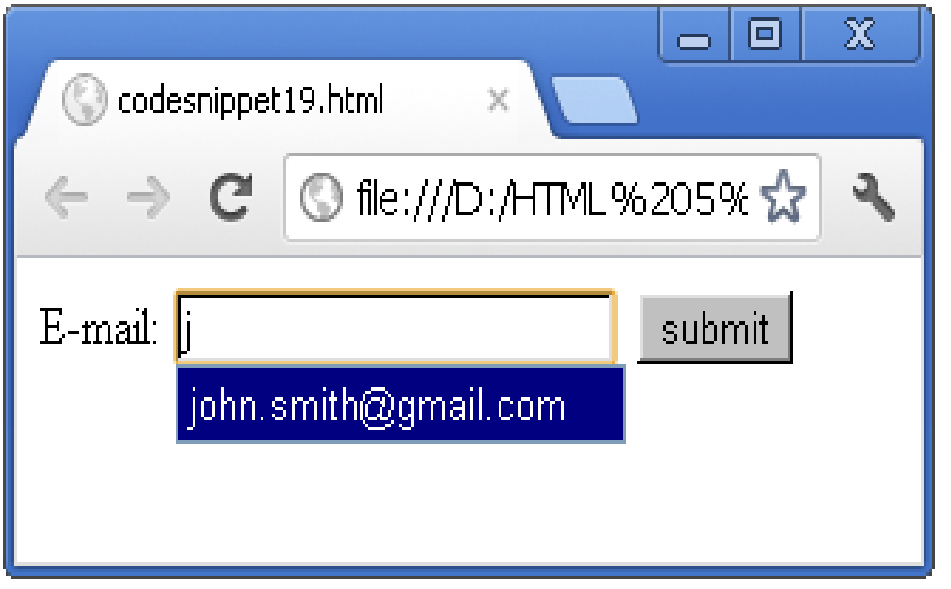



**Figure 10.18: Behavior of Autocomplete Attribute in Chrome**

Code Snippet 19 demonstrates to disable the default behavior of `autocomplete` attribute.

**Code Snippet 19**:

```
E-mail: <input type="email" name="email" autocomplete="off" />
        <input type="submit" value="submit"/>
```

## *10.6 New Form Elements*

HTML5 has introduced some brand new elements that can be incorporated in the Web pages. These new elements are specifically designed to be used with the JavaScript. When combined with JavaScript, these new elements can be more functional.

At present, all the browsers do not provide the support for these new elements. If the control is not supported by the browser, then it displays element as a text field. Opera provides the support for all the new form elements.

The new form elements introduced in HTML5 are as follows:

- Datalist
- Progress
- Meter
- Output

## *10.6.1 Datalist*

Datalist is a form-specific element. It provides a text field with a set of predefined list of options that are displayed in a drop-down list. When the text field receives focus, a list of options is displayed to the user.

The `<datalist>` element is very similar to standard `<select>` element available in earlier HTML. The only difference in `datalist` is that it allows the user to enter data of their choice or select from the suggested list of options.

The lists of options for the `<datalist>` element are placed using the `option` element. Then, the `datalist` is associated with an input element using the `list` attribute. The value of the `list` attribute is the value of `id` attribute provided with the `<datalist>` element. The same `datalist` can be associated with several input fields. At present, only Opera browser provides the support for the `datalist`.

Code Snippet 20 shows the syntax of providing the `<datalist>` element on the form.

**Code Snippet 20**:

```
<label> Select the mode of payment: </label>
 <input type="text"  name="payment" list="paymentlist" />
  <datalist id="paymentlist">
        <option value="Cash-on-Delivery">
        <option value="Net Banking">
        <option value="Credit Card">
        <option value="Debit Card">
        <option value="e-Gift Voucher">
  </datalist>
 <input type="submit" value="submit"/>
```

In the code, a `datalist` requires `value` attribute to be added with the `<option>` tag. Values nested between the opening and closing `<option>` tag will not be displayed in the `datalist` menu.

Figure 10.19 shows the `<datalist>` element in Opera.

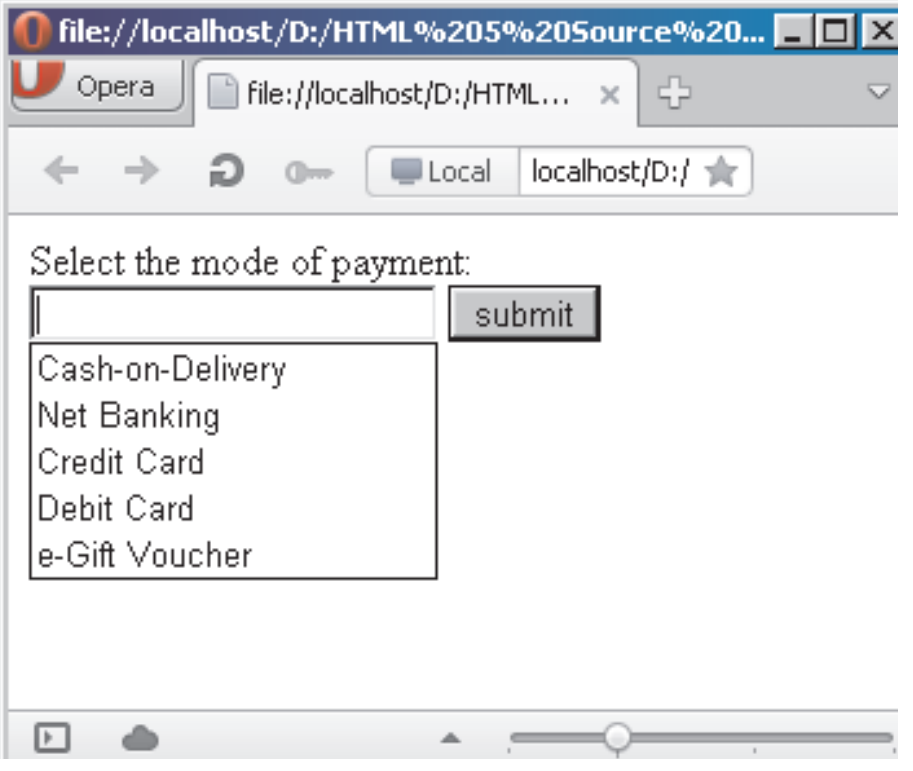



**Figure 10.19: Datalist Element in Opera**

As shown in figure 10.19, the datalist is displayed with simple text field that shows the suggested list of options in a drop-down list on focus.

## *10.6.2 Progress*

The `progress` element represents the current status of a task, which gradually changes as the task heads for completion. This is not a form-specific element.

For example, when the user downloads any file from a particular Web page, the download task is represented as a progress bar.

Code Snippet 21 shows the syntax for providing `progress` element on the form.

**Code Snippet 21**:

```
<label> Downloading status: </label>
 <progress value="35" max="100" ></progress>
 <input type="submit" value="submit"/>
```

In the code, the `progress` element contains two attributes namely, `max` and `value`. The `max` attribute declares the maximum value for the task to be processed for its completion.

The `value` attribute indicates how much task has been processed so far. Figure 10.20 shows the `progress` element in Opera.

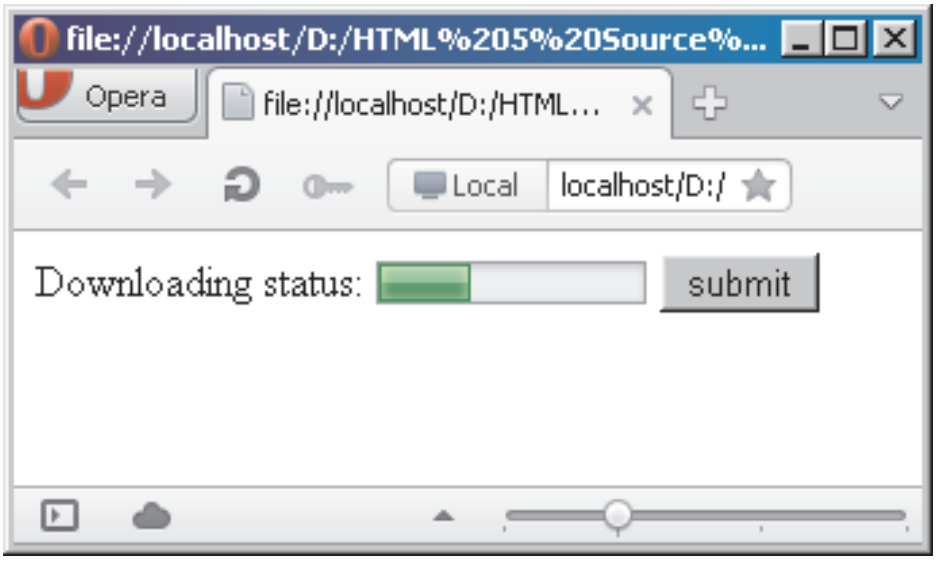



Figure 10.20: Progress Element in Opera

## *10.6.3 Meter*

The `meter` element represents a measurement scale for a known range. The known range have the definite minimum and maximum values to measure the data on the scale. For example, a `meter` element can be used to represent measurements, such as disk usage space, fraction value, or significance of a query result. All these have a known maximum value defined for them. The `meter` element cannot indicate age, height, or weight, as maximum values for them cannot be specified.

Code Snippet 22 shows the code of the `meter` element.

**Code Snippet 22**:

```
<label> Total score of marks: </label>

0   <meter min="0" max="400" value="180" title="numbers scored" low="120"
high="300"> </meter>   400<br/>

<input type="submit" value="submit"/>
```

In the code, the `meter` element contains six attributes that are used to determine the measurements in the known range.

The `min` and `max` attribute specifies the minimum and maximum value that sets bounds for the range. The default value for the `max` attribute is `1`. The `value` attribute specifies the current measured value. The `high` and `low` attributes specifies the range of values that can be considered as `high` or `low` for the given range.

For example, in the given range of scores, the range of values below `120` will be considered `low`, but anything above `300` will be considered as `high`.

There is another attribute named `optimum` which refers to the ideal value for the measurement.

Figure 10.21 shows the `meter` element in Opera.

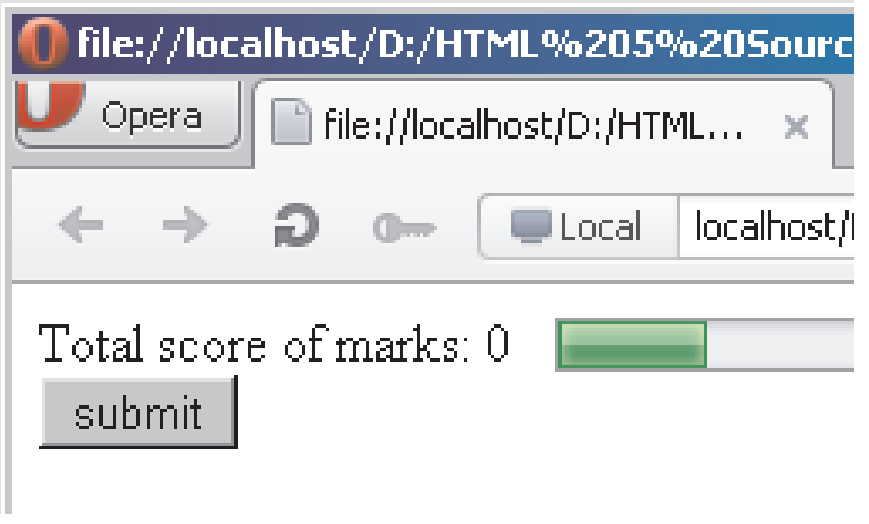



Figure 10.21: Meter Element in Opera

## *10.6.4 Output*

The `output` element displays the results of a calculation on a form. The result values displayed in the `output` element are processed from the other form elements. For example, the `output` element might be used to display the total cost on the purchase items after calculating discount amount in a registration form or purchase order form.

Code Snippet 23 shows the calculation of data from other form elements to be displayed in the `output` element.

**Code Snippet 23**:

```
<form oninput="x.value = parseInt(type.value)*
 parseInt(duration.value)">
 <label>Membership Type:</label>
  <select name="type">
   <option value="400">Gold - $400</option>
   <option value="500">Silver - $500</option>
   <option value="600">Platinum - $600</option>
  </select>
```

```
        <label>Duration [years]:</label>
      <input type="number" value="0" name="duration"
       min="1"max="5" step="1" />
      <label> Annual Payment Fees: $.</label>
    <output name="x" for="type duration"></output>
```

In the code, `for` attribute relates the `output` element with the elements whose values are taken for calculation. The form `oninput` event handles the `input` event which is fired whenever the value of the elements change on receiving input from the user. A JavaScript code can also be written to update the values for the `output` element.

Figure 10.22 shows the result of calculation for `output` element.

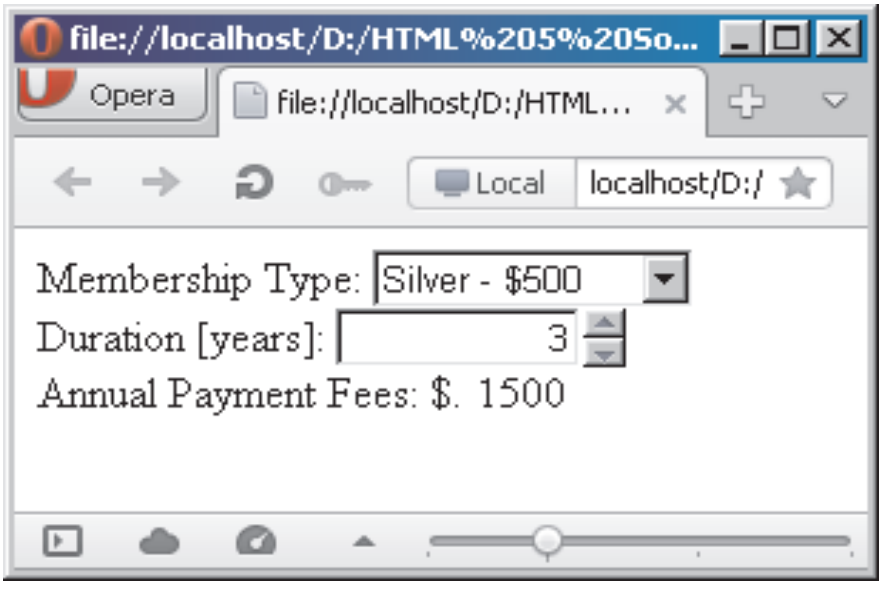



**Figure 10.22: Result of Calculation for Output Element**

## *10.7 Check Your Progress*

1. Which of the following element is a data field that allows the user to edit the data on the form?

| | | | |
|---|---|---|---|
| (A) | required | (C) | output |
| (B) | input | (D) | progress |

2. Which of the following attribute displays a short hint or text inside a form element?

| | | | |
|---|---|---|---|
| (A) | meter | (C) | caption |
| (B) | type | (D) | placeholder |

3. The _____ element represents the current status of a task, which gradually changes as the task heads for completion.

| | | | |
|---|---|---|---|
| (A) | datalist | (C) | progress |
| (B) | pattern | (D) | required |

4. Which of the following statements are true about Datalist?

| | |
|---|---|
| (A) | Provides a text field with a set of predefined list of options that are displayed in a drop-down list |
| (B) | Is a form-specific element |
| (C) | `<datalist>` element is very similar to standard `<select>` element |
| (D) | Does not allow the user to enter data of their choice or select from the suggested list of options |

5. Which of the following code disables the default behavior of autocomplete attribute?

| | |
|---|---|
| (A) | E-mail: <input type="email" name="email" autocomplete="on" /><br><input type="submit" value="submit"/> |
| (B) | E-mail: <input type="email" name="email" autocomplete="yes" /><br><input type="submit" value="submit"/> |
| (C) | E-mail: <input type="email" name="email"/><br><input type="submit" value="submit"/> |
| (D) | E-mail: <input type="email" name="email" autocomplete="off" /><input type="submit" value="submit"/> |

## *10.7.1 Answers*

| | |
|---|---|
| 1. | B |
| 2. | D |
| 3. | C |
| 4. | A, B, C |
| 5. | D |

# Summary

- HTML5 provides a great enhancement to Web forms.
- Creation of form is made easier for Web developers by standardizing them with rich form controls.
- HTML5 introduces new form elements such as new input types, new attributes, browser-based validation, CSS3 styling techniques, and forms API.
- HTML5 provides new input types that are data-specific user interface elements such as email, url, number, range, date, tel, and color.
- The new form elements introduced in HTML5 are namely, datalist, progress, meter, and output.
- HTML5 has provided several new attributes that performs the validations without writing JavaScript snippets for them.
- In HTML5, one can use the submit input type for form submission.

# *Try it Yourself*

1. Samson works for an advertising agency named Creative Designers which is headquartered at Hong Kong. He is very fond of learning latest technologies that are coming up in the market. He wants to create an HTML5 Web site for his advertising agency. The Web site should display the list of products for sale such as books, computers, vehicles, cameras, laptops, musical instruments, and so on. Only registered users can purchase products from the site. Help him to develop the application.

# Session - 11

# HTML5 Audio and Video

Welcome to the Session, **HTML5 Audio and Video**.

This session describes the supported media types, audio, and video elements. It further describes the accessibility of audio elements, video elements, and how to deal with non-supporting browsers.

In this Session, you will learn to:

- Describe the need for multimedia in HTML5
- List the supported media types in HTML5
- Explain the audio elements in HTML5
- Explain the video elements in HTML5
- Explain the accessibility of audio and video elements
- Describe how to deal with non-supporting browsers

## *11.1 Introduction*

Traditionally, Web browsers were capable of handling only graphics and text. Suppose, if a user had to play some video, then, a distinct program, plug-in, or an ActiveX control had to be installed. Earlier, Web designers and Web developers used to set up Web pages to play audio and video on the Web using Adobe Flash player.

## *11.2 Multimedia in HTML5*

Multimedia is a combination of various elements such as video, graphics, sound, and text. A common way of inserting a multimedia content on Web pages is by embedding a video or audio file in the Web page.

Consider the earlier situations where a Web site developer did not have the facility of including videos or audios directly on their Web site until and unless the browser had the required plug-in installed. These days, Web site developers want their visitors to not only download, but also, view movies online on their Web site. This is possible by adding the new features of HTML5 which can provide this facility.

HTML5 has made lives easier by introducing `<audio>` and `<video>` elements. In other words, HTML5 has provided the developers with the features to embed media on the Web pages in a standard manner. Thus, the user need not depend on Flash to access the audio and video files.

## *11.3 Supported Media Types in Audio and Video*

There are various video and audio codecs which are used for handling of video and audio files. The codec is a device or a program used for encoding and decoding digital data stream. These different codecs have different level of compression quality.

For storing and transmitting coded video and audio together, a container format is used. There are a number of container formats which includes Ogg (.ogv), the Audio Video Interleave (.avi), Flash Video (.flv), and many others. WebM is a new open source video container format supported by Google. Different browsers support different container format.

Table 11.1 lists the common audio and video formats.

| Container | Video Codec | Audio Codec |
|---|---|---|
| Mp4 | H.264 | AAC |
| Ogg | Theora | Vorbis |
| WebM | VP8 | Vorbis |

**Table 11.1: Common Audio and Video Formats**

## *11.3.1 Audio Formats*

There are the three supported file formats for the `<audio>` element in HTML5. Table 11.2 lists the audio file formats supported by the Web browsers.

| Browsers Support | MP3 | Wav | Ogg |
|---|---|---|---|
| Opera 10.6 | No | Yes | Yes |
| Apple Safari 5 | Yes | Yes | No |
| Google Chrome 6 | Yes | Yes | Yes |
| FireFox 4.0 | No | Yes | Yes |
| Internet Explorer 9 | Yes | No | No |

**Table 11.2: Audio File Formats Supported by the Web Browsers**

## *11.3.2 Video Formats*

There are the three supported file formats for the `<video>` element in HTML5. Table 11.3 lists the video file formats supported by the Web browsers.

| Browsers Support | MP4 | WebM | Ogg |
|---|---|---|---|
| Opera 10.6 | No | Yes | Yes |
| Apple Safari 5 | Yes | No | No |
| Google Chrome 6 | Yes | Yes | Yes |
| FireFox 4.0 | No | Yes | Yes |
| Internet Explorer 9 | Yes | No | No |

**Table 11.3: Video File Formats Supported by the Web Browsers**

## *11.4 Audio Elements in HTML5*

The `<audio>` element will help the developer to embed music on the Web site and allow the user to listen to music. The `<audio>` element is one of the best features in HTML5. This feature allows the user to enable a native audio file within the Web browser. The `<audio>` tag specifies the audio file to be used in the HTML document. The `src` attribute is used to link to the audio file.

Code Snippet 1 displays the embedding of an audio file in the Web page using the `<audio>` tag. The music is played in the background when the page is loaded on the browser.

**Code Snippet 1**:

```
<!DOCTYPE html>
 <html>
  <head>
    <title>audio element</title>
  </head>
    <body>
      <audio src="d:\sourcecodes\audio.mp3"
       controls autoplay   loop>
       html5 audio not supported
      </audio>
    </body>
 </html>
```

The `src` attribute is mandatory, the `<audio>` tag includes several other options.

Figure 11.1 displays the `<audio>` element.

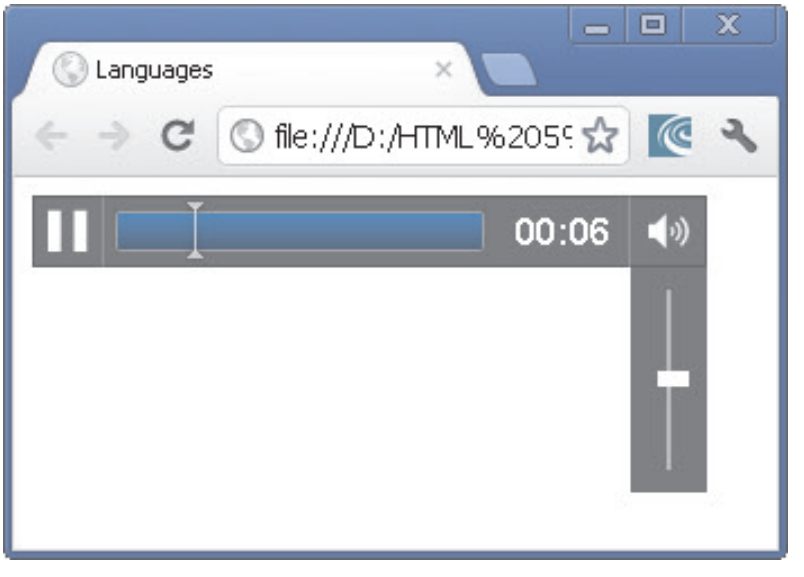



**Figure 11.1:** `<audio>` **Element**

The existing HTML5 specification does not specify the formats supported by the browser in the `<audio>` tag. The audio formats frequently used are **wav**, **ogg**, and **mp3**.

The users can also use the `<source>` tag to specify the media along with the media type and other attributes. An audio element can contain multiple source elements and the browser will identify the first supported format.

## *11.4.1 Audio Tag Attributes*

HTML tags normally consist of more than one attribute. Attributes provide additional information to the browser about the tag. HTML5 has a number of attributes for controlling the look and feel of various functionalities. According to HTML5 specifications, the `<audio>` element has the following attributes. Table 11.4 lists the `<audio>` tag attributes.

| Audio Attributes | Description |
|---|---|
| `autoplay` | This attribute identifies whether to start the audio or not once the object is loaded. The attribute accepts a boolean value which when specified will automatically start playing the audio as soon as possible without stopping |
| `autobuffer` | This attribute starts the buffering automatically |
| `controls` | This attribute identifies the audio playback controls that should be displayed such as resume, pause, play, and volume buttons |
| `loop` | This attribute identifies whether to replay the audio once it has stopped |
| `preload` | This attribute identifies whether the audio has to be loaded when the page loads and is ready to execute. This preload attribute is ignored if autoplay exists |
| `src` | This attribute specifies the location or the URL of the audio file that has to be embedded |

**Table 11.4: <audio> Tag Attributes**

## *11.4.2 Creating Audio Files*

Suppose, if the user plays the audio in older browsers then the `<embed>` tag will be used. The `<embed>` tag has two attributes, `src` and `autostart`. The `src` attribute is used to specify the source of the audio and the `autostart` attribute controls the audio and determines whether the audio should play as soon as the page loads.

Code Snippet 2 demonstrates the use of `<embed>` tag in the `<audio>` element.

**Code Snippet 2**:

```
<!DOCTYPE HTML>
<html>
<body>
   <audio autoplay loop>
       <source src="sampaudio.mp3">
       <source src="sampaudio.ogg">
       <embed src="sampaudio.mp3">
   </audio>
</body>
</html>
```

The `<audio>` element in HTML5 supports multiple formats. The content included within the `<embed>` tag is automatically played by default. Suppose, if the user does not want to play the audio file automatically then he/she can set the value of the `autostart` attribute to `"false"`.

Code Snippet 3 demonstrates the use of the `autostart` attribute.

**Code Snippet 3**:

```
<embed src="mpaudio.mp3" autostart="false" >
```

The `<embed>` tag also supports another attribute named `loop`. The `loop` attribute determines whether the audio clip will be replayed continuously or not. If the value of the `loop` attribute is set to `true` or `infinite` then the music will be played continuously. If the `loop` attribute is not specified then it is same as setting the value to `false`.

Code Snippet 4 demonstrates the use of some of the audio formats supported by HTML5.

**Code Snippet 4**:

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

This code snippet shows the **ogg** and **wav** formats supported by the `<audio>` tag. While adding the `<audio>` element in the code, the user can specify error messages to check if the browser is supporting the `<audio>` tag or not.

## *11.5 Video Elements in HTML5*

The `<video>` element is a new feature added in HTML5. The user can use the `<video>` element for embedding the video content on the Web page. The easiest way to specify the video is by using the `src` attribute which give the URL of the video file to be used. Suppose, if the browser does not support the `<video>` element then the content between the start tag and end tag is displayed on the browser.

Code Snippet 5 demonstrates the use of the `<video>` element.

**Code Snippet 5**:

```
<!DOCTYPE HTML>
 <html>
   <head>
   </head>
     <body>
      <video src="D:\Source codes\movie.mp4">
       Your browser does not support the video.
      </video>
     </body>
</html>
```

In the code, the `src` attribute is used for specifying the location of the mp4 video file format used by the `<video>` tag. While adding the `<video>` element in the code, the user can specify messages between the `<video>` and `</video>` tag to check if the browser is supporting the `<video>` tag or not.

Figure 11.2 displays the `<video>` element.

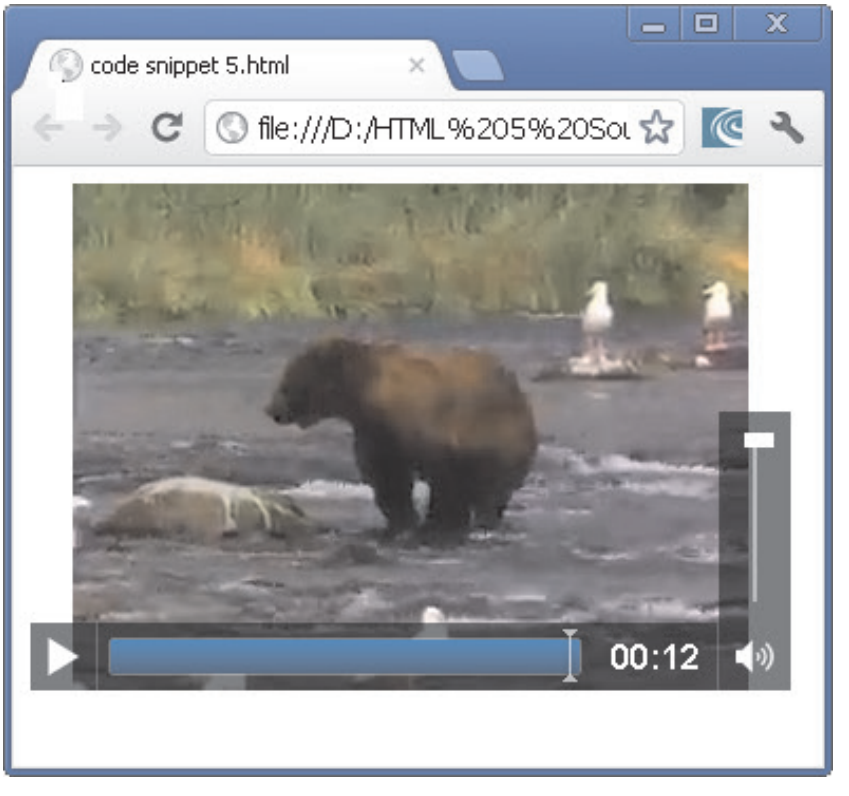



**Figure 11.2:** `<video>` **Element**

## *11.5.1 Video Tag Attributes*

The HTML5 specification provides a list of attributes that can be used with the `<video>` element. Table 11.5 lists the `<video>` tag attributes.

| Video Attributes | Description |
|---|---|
| `autoplay` | Specifies that the browser will start playing the video as soon as it is ready |
| `muted` | Allows to mute the video initially, if this attribute is existing |
| `controls` | Allows displaying the controls of the video, if the attribute exists |
| `loop` | Specifies that the browser should repeat playing the existing video once more if the loop attribute exists and accepts a boolean value |
| `preload` | Specifies whether the video should be loaded or not when the page is loaded |
| `src` | Specifies the location of the video file to be embedded |

**Table 11.5: <video> Tag Attributes**

**Note -** The muted attribute is not supported in Safari and Internet Explorer.

## *11.5.2 Preloading the Video*

The `<video>` element comprises a `preload` attribute that allows the browser to download or buffer the video while the Web page containing the video is being downloaded. If the video is preloaded, then it decreases the initial delay once the user has started the playback. The `preload` attribute has the following values:

➔ **None**

This attribute allows the browser to load only the page. The video will not be downloaded while the page is being loaded.

➔ **Metadata**

This attribute allows the browser to load the metadata when the page is being loaded.

➔ **Auto**

This is the default behavior as it allows the browser to download the video when the page is loaded. The browser can avoid the request.

Code Snippet 6 demonstrates the use of `none` and `metadata` values for the `preload` attribute.

**Code Snippet 6**:

```
<!DOCTYPE HTML>
<html>
 <head>
 </head>
  <body>
  <video width="160" height="140" src="D:\Source Codes\movie.mp4"
    controls preload="none" muted>
   Your browser does not support the video.
  </video>
  <video width="160" height="140" src="D:\ Source Codes\movie.mp4"
    controls preload="metadata" muted>
   Your browser does not support the video.
  </video>
  </body>
</html>
```

In the code, the `preload` attribute specifies `none` and `metadata` values.

Figure 11.3 displays the effect of `none` and `metadata` values.

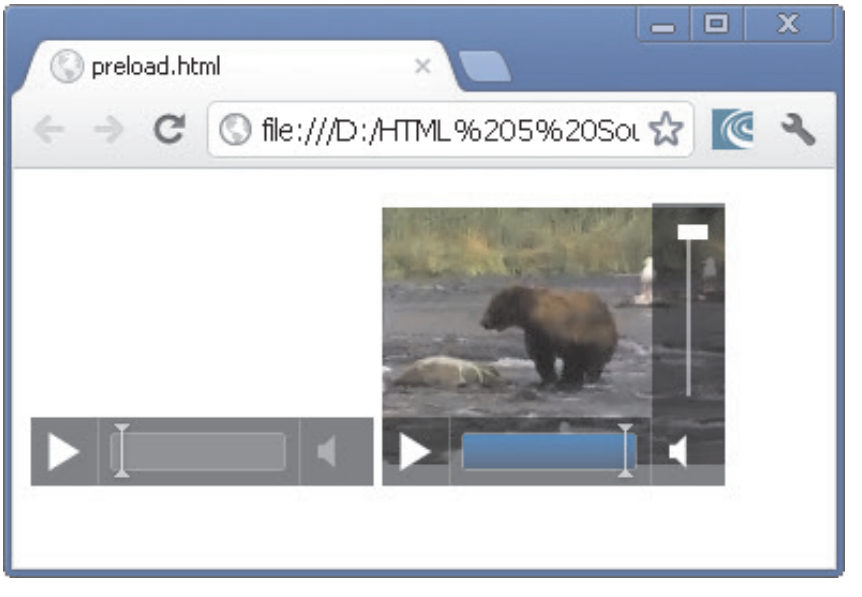



**Figure 11.3: Effect of none and metadata Values**

## *11.5.3 Setting the Video Size*

The user can specify the size of the video with the `height` and `width` attribute of the `<video>` element. Suppose, if these attributes are not provided then the browser sets the video with the key dimensions of the video. This will result in changing the page layout as the Web page is adjusted to accommodate the video.

Code Snippet 7 demonstrates how to apply the `height` and `width` attributes to the `<video>` element.

**Code Snippet 7**:

```
<!DOCTYPE HTML>
 <html>
   <head>
   </head>
   <title> Video Size</title>
     <style>
      video{
             background-color: black;
               border: medium double black;
            }
     </style>
     <body>
     <video src="D:\Source Codes\movie.mp4"
      controls preload="auto" width="360" height="340">
      Your browser does not support the video.
     </video>
      </body>
 </html>
```

In the code, the `style` attribute is used to specify the `background-color` and `border style` of the video. The code also specifies the `preload`, `height`, and `width` attributes for the `<video>` element.

Figure 11.4 displays the `width`, `height`, and `style` effect.

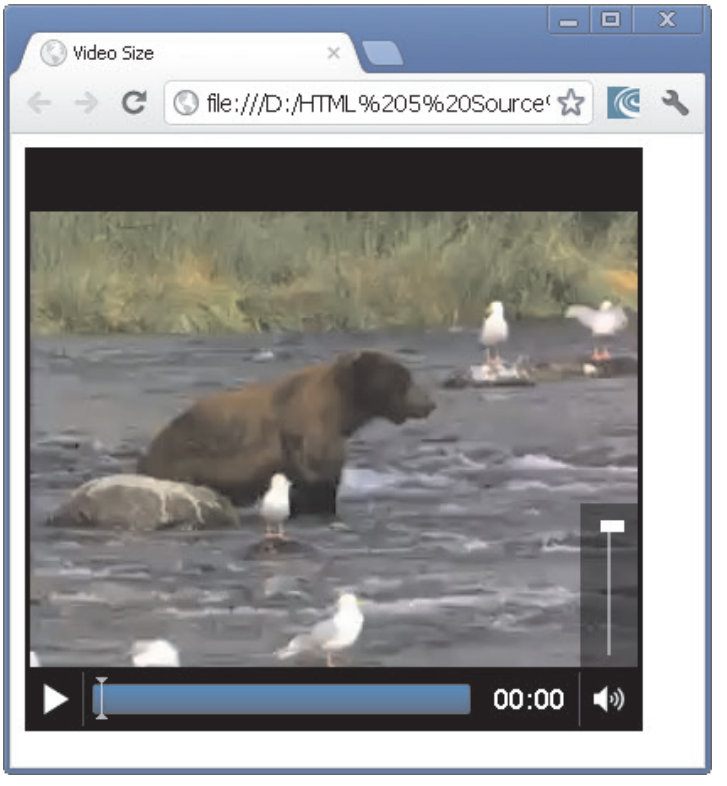



**Figure 11.4: Width, Height, and Style Effect**

## *11.5.4 Converting the Video Files*

The `<video>` element used in HTML5 is a great feature but how will the user get the video files in a correct format. There are many problems with browser vendors for supporting the various video formats on the Web sites. The following are some of the video formats supported by the significant browsers:

➔ **Ogg/Theora**

This is an open source, royalty-free, and patent-free format available. This format is supported by browsers such as Opera, Chrome, and Firefox.

➔ **WebM**

This is a royalty-free and patent-free format supported by Google. This format is supported by browsers such as Opera, Chrome, and Firefox. Free WebM Encoder 1.2 is a simple utility that allows you to convert video files to the WebM format.

➔ **H.264/MP4**

H.264 or MP4 formats are supported on iPhone and Google Android devices.

There is a simple way to encode H.264 is by using the Handbrake. Handbrake is an open-source, GPL-license application and is accessible by using Mac OS X, Windows, and Linux. Handbrake has two versions, command-line and graphical versions.

This format is available for free till 2015. This format is supported by browsers such as Internet Explorer, Chrome, and Safari.

➔ **Micro Video Controller**

This converter creates all files that the user requires for HTML5 `<video>` element that works on the cross browser.

## *11.6 Accessibility of Audio and Video Elements*

Enterprises across the world are employing people with varied skills and abilities. They may even include people with limited abilities or disabilities such as people with visual, cognitive, or mobility impairments. Accessibility is the level of ease with which computers can be used and be available to a wide range of users, including people with disabilities.

Applications can be accessed through various sources. If the application considers the requirements of the target audience, then it will be appreciated and used by number of users.

There are various types of users who will view the application containing the media content. Therefore, while developing an application a lot of assumptions are to be considered and some of them are as follows:

➔ Users can check the content on laptop, mobile, tablet, or desktop

➔ Users can listen to the audio by using headphones or speakers

➔ Users can understand the language in which the media was delivered

➔ Users can successfully play and download the media

These assumptions meet the requirements of a vast majority of users accessing the application. However, not all users will fall in this category. Therefore, another set of assumptions are to be considered for these users and they are as follows:

➔ Users who have hearing and visual impairment and thus, cannot listen to the audio or view the video

➔ Users who are not familiar with the language that the content is delivered

➔ Users who use keyboards and screen readers to access the content on Web

➔ Users who cannot view or hear the media content because of their working environment or due to device restrictions

HTML5 provides powerful features to make applications accessible to such users.

WebVTT (Web Video text Tracks) is a file format used to mark up the external text tracks. This format allows the user to give a textual description of the content in the video. This description is then used by different accessibility devices to define the content to those users who cannot see it.

## *11.6.1 The Track Element*

The track element provides an easy, standard way to add captions, subtitles, chapters, and screen reader descriptions to the `<audio>` and `<video>` elements.

Track elements are also used for other types of timed metadata. The source data for this track element is in a form of a text file that is made up of a list of timed cues. A cue is a pointer at an accurate time point in the length of a video. These cues contain data in formats such as Comma-Separated Values (CSV) or JavaScript Object Notation (JSON).

The track element is not supported in many major browsers. This track element is now available in IE 10 and Chrome 18+.

Table 11.6 lists the track element attributes.

| Container | Description |
|---|---|
| `src` | Contains the URL of the text track data |
| `srclang` | Contains the language of the text track data |
| `kind` | Contains the type of content the track definition is used for |
| `default` | Indicates that this will be the default track if the user does not specifies the value |
| `label` | Specifies the title to be displayed for the user |

**Table 11.6: Track Element Attributes**

Code Snippet 8 demonstrates how a track element is used in combination with `<video>` element for providing subtitles.

**Code Snippet 8**:

```
<video controls>
    <source src="myvideo.mp4" type="video/mp4">
    <source src="myvideo.webm" type="video/webm">
 <track src="eng.vtt"
    label="English p subtitles"  kind="subtitles"
    srclang="en" >
</video>
```

This code specifies the `src`, `label`, and `srclang` attributes in the track element. Here, the `srclang` is set to `en` that is English language.

Code Snippet 9 demonstrates a track element used in combination with `<video>` element providing subtitles in another language.

**Code Snippet 9**:

```
<video controls>
    <source src="myvideo.mp4" type="video/mp4">
    <source src="myvideo.webm" type="video/webm">
    <track src="de.vtt" srclang="de" label="German p subtitles"
               kind="subtitles">
</video>
```

This code specifies the `src`, `label`, and `srclang` attributes in the track element. Here, the `srclang` is set to `de` which represents French language.

## *11.6.2 Accessibility for Audio and Video Element*

There are some accessibility supports for `<audio>` and `<video>` elements. These are as follows:

➔ **Audio Support**

The following are the accessibility support for `<audio>` elements:

- **Firefox** - This browser exposes controls with accessibility APIs, however individual controls do not interact with keyboard. The access to keyboard is provided by the Firefox specific shortcuts.
- **Opera** - This browser has only keyboard support.

- **IE 9** - This browser expose controls with accessibility APIs, however individual controls do not interact with keyboard.

➔ **Video Support**

The following are the accessibility support for `<video>` elements:

- **Firefox** - This browser cannot interact with individual controls.
- **Opera** - This browser has only keyboard support.
- **IE 9** - This browser does not allow individual controls to interact with keyboard.

## *11.7 Non-Supporting Browsers*

There are many browsers that do not support HTML5 elements. Browsers such as Firefox, IE9, Chrome, Opera, and Safari support the `<audio>` and the `<video>` elements. Google chrome 17 and lower version has no support for `<audio>` elements. Similarly, Safari browser does not support `<audio>` element in HTML5. Internet Explorer 8 and earlier versions do not support the `<audio>` and the `<video>` elements.

## *11.8 Check Your Progress*

1. _____ is a combination of various elements such as video, graphics, sound, and text.

| | | | |
|---|---|---|---|
| (A) | Audio | (C) | Video |
| (B) | Multimedia | (D) | Autoplay |

2. Which of the following element allows the user to enable a native audio file within the Web browser?

| | | | |
|---|---|---|---|
| (A) | Audio | (C) | Video |
| (B) | Text | (D) | Autoplay |

3. _____ is a new open source video container format supported by Google.

| | | | |
|---|---|---|---|
| (A) | Flv | (C) | Avi |
| (B) | Ogg | (D) | WebM |

4. _____ is the level of ease with which computers can be used and be available to a wide range of users, including people with disabilities.

| | | | |
|---|---|---|---|
| (A) | Browser | (C) | Accessibility |
| (B) | Element | (D) | Autoplay |

5. Which of the following browser supports accessibility for audio and video elements?

| | | | |
|---|---|---|---|
| (A) | Firefox | (C) | Netscape Navigator |
| (B) | IE9 | (D) | Opera |

## *11.8.1 Answers*

| | |
|---|---|
| 1. | B |
| 2. | A |
| 3. | D |
| 4. | C |
| 5. | A, B, D |

# Summary

- Multimedia is a combination of various elements such as video, graphics, sound, and text.
- There are various media types used for audio and video files on different Web sites.
- The <audio> element will help the developer to embed music on the Web site and allow the user to listen to music.
- Users can play the audio in older browsers using the <embed> tag.
- The <video> element is used for embedding the video content on the Web page.
- Preload attribute identifies whether the audio has to be loaded when the page loads and is ready to execute.
- WebM is a new open source video container format supported by Google.

# *Try it Yourself*

1. Julia works for a music online Web site named, Crayon Developers and is headquartered at Tokyo, Japan. The Web site contains the information about the different genre of music such as jazz, opera, rock, and many more. Now, she wants to add the list of newly launched albums to the Web site. She wants to add some audio files to the Web site for the listeners that will create a different impact on the customers. Help her to develop the application.

2. Ching Chow works for an online music company named, MovieBuzz and is headquartered at Yokohama, China. She has created her company's Web site on HTML 4 now she wants to upgrade her company's Web site to HTML5. The Web site contains the information about the latest Hollywood movies and review of the released movies. She wants to add the new movie releases to her collection of movies where the visitors can view some video clips of movies. Help her to develop the application.

# Session - 12

# Introduction to JavaScript

Welcome to the Session, **Introduction to JavaScript**.

This session describes the scripting languages. The session also explores the JavaScript language and versions used in the language. It further describes the variables, data types, methods, built in functions, event handling, and jQuery mobile.

In this Session, you will learn to:

- Explain scripting
- Explain the JavaScript language
- Explain the client-side and server-side JavaScript
- List the variables and data types in JavaScript
- Describe the JavaScript methods to display information
- Explain escape sequences and built in functions in JavaScript
- Explain events and event handling
- Explain jQuery
- Describe how to use the jQuery Mobile

## *12.1 Introduction*

Consider an organization that provides a Web site that allows its customers to view their products. The company has received frequent customer feedbacks to provide the shopping facility online. Therefore, the company has decided to add the shopping facility in their Web site by creating dynamic Web pages. These Web pages will allow the user to shop for the products online. Here, the main task of the developer is to validate the customer's inputs while they shop online. For example, details such as credit card number, email, and phone number entered by the customer must be in a proper format. Further, the developer also needs to retrieve the chosen products and their quantity to calculate the total cost.

The developer can handle all these critical tasks by using a scripting language. A scripting language refers to a set of instructions that provides some functionality when the user interacts with a Web page.

Figure 12.1 displays the need for scripting.

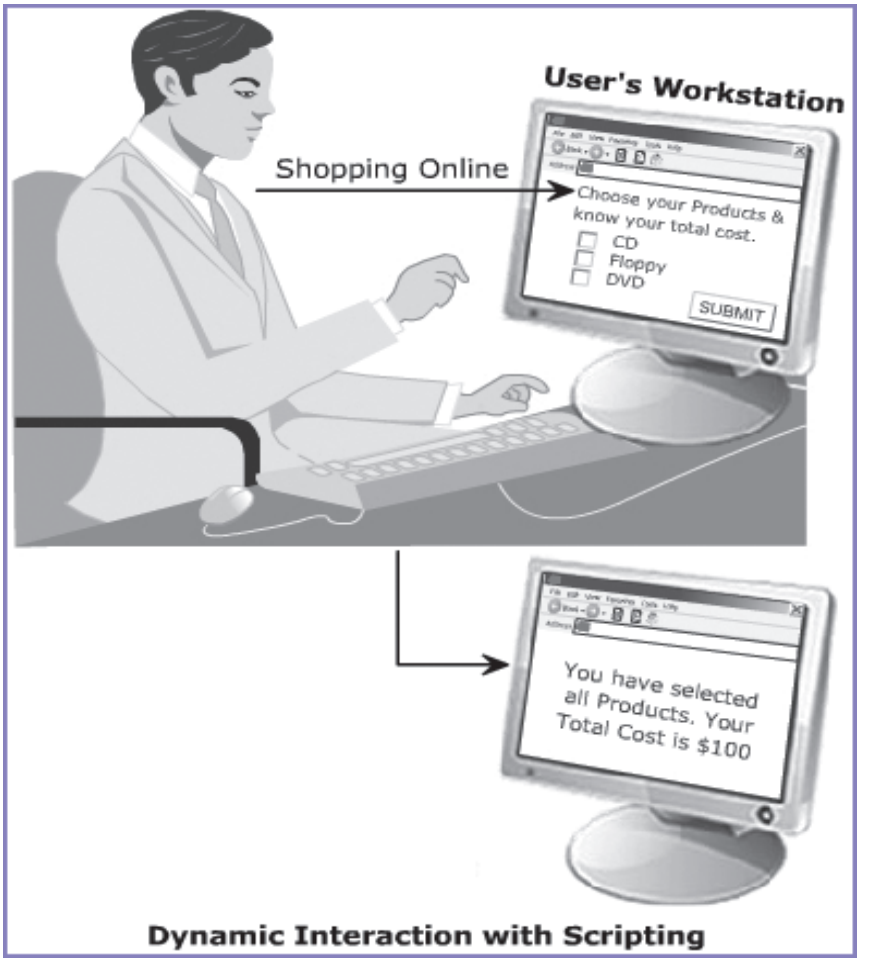



**Figure 12.1: Need for Scripting**

## *12.2 Scripting*

Scripting refers to a series of commands that are interpreted and executed sequentially and immediately on occurrence of an event. This event is an action generated by a user while interacting with a Web page. Examples of events include button clicks, selecting a product from a menu, and so on. Scripting languages are often embedded in the HTML pages to change the behavior of the Web pages according to the user's requirements.

There are two types of scripting languages. They are as follows:

➔ **Client-side Scripting**

Refers to a script being executed on the client's machine by the browser.

➔ **Server-side Scripting**

Refers to a script being executed on a Web server to generate dynamic HTML pages.

Figure 12.2 displays the types of scripting.

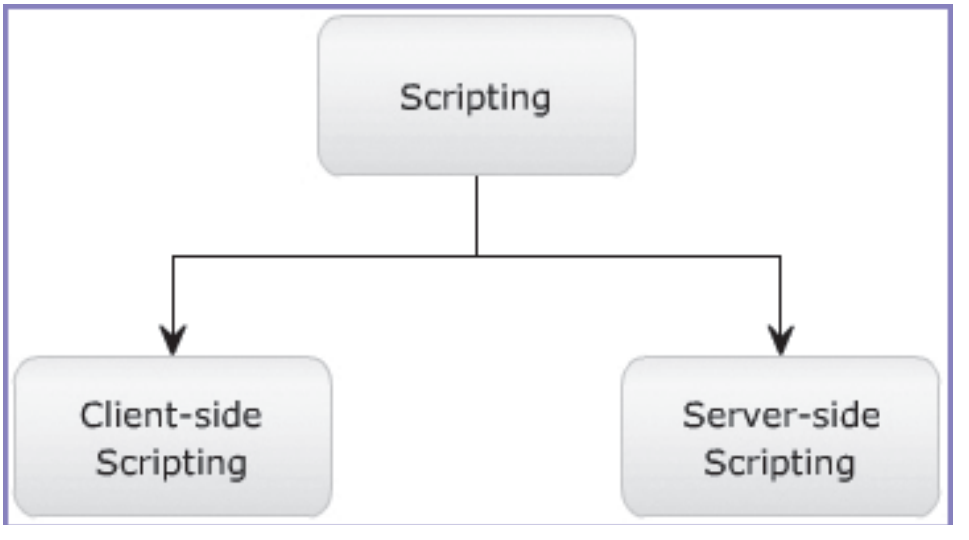



**Figure 12.2: Types of Scripting**

**Note** - In earlier days, script languages were called batch languages or job control languages.

## *12.3 JavaScript*

JavaScript is a scripting language that allows you to build dynamic Web pages by ensuring maximum user interactivity.

JavaScript language is an object-based language, which means that it provides objects for specifying functionalities. In real life, an object is a visible entity such as a car or a table. Every object has some characteristics and is capable of performing certain actions. Similarly, in a scripting language, an object has a unique identity, state, and behavior.

The identity of the object distinguishes it from the other objects of the same type. The state of the object refers to its characteristics, whereas the behavior of the object consists of its possible actions.

The object stores its identity and state in fields (also called variables) and exposes its behavior through functions (actions).

Figure 12.3 displays the objects.

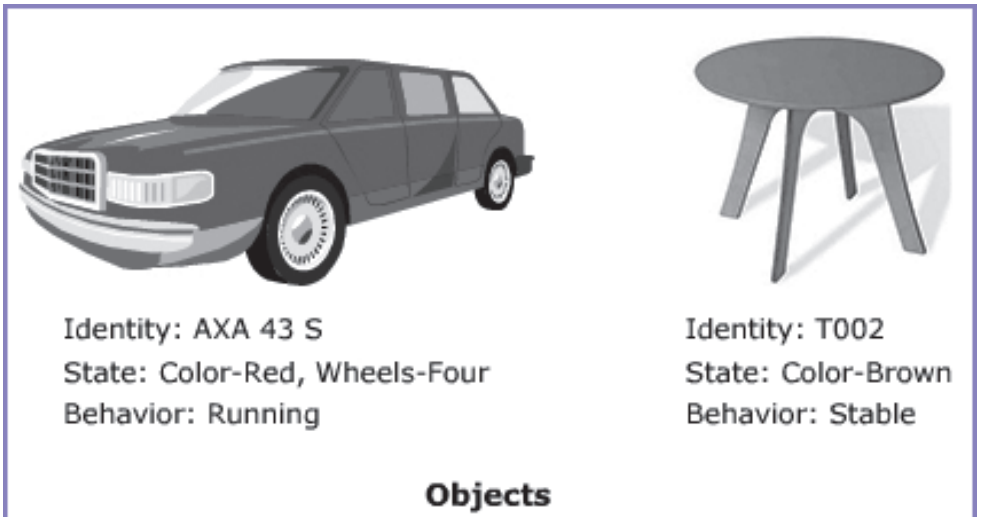



Figure 12.3: Objects

## *12.4 Versions of JavaScript*

The first version of JavaScript was developed by Brendan Eich at Netscape in 1995 and was named JavaScript 1.0. Netscape Navigator 2.0 and Internet Explorer 3.0 supported JavaScript 1.0. Over the period, it gradually evolved with newer versions where each version provided better features and functionalities as compared to their previous versions.

Table 12.1 lists the various versions of JavaScript language.

| Version | Description |
|---|---|
| 1.1 | Is supported from 3.0 version of the Netscape Navigator and Internet Explorer |
| 1.2 | Is supported by the Internet Explorer from version 4.0 |
| 1.3 | Is supported by the Internet Explorer from version 5.0, Netscape Navigator from version 4.0, and Opera from version 5.0 |
| 1.4 | Is supported by servers of Netscape and Opera 6 |
| 1.5 | Is supported by the Internet Explorer from version 6.0, Netscape Navigator from version 6.0, and Mozilla Firefox from version 1.0 |
| 1.6 | Is supported in the latest versions of the Internet Explorer and Netscape Navigator browsers. It is also supported by Mozilla Firefox from version 1.5 |

| Version | Description |
|---|---|
| 1.7 | Is supported in the latest versions of the Internet Explorer and Netscape Navigator browsers. It is also supported by Mozilla Firefox from version 2.0 |

Table 12.1: Various Versions of JavaScript Language

**Note** - The latest version, JavaScript 2.0, is into the development phase.

## *12.5 Client-side JavaScript*

JavaScript is a scripting language, which can be executed on the client-side and on the server-side. A client-side JavaScript (CSJS) is executed by the browser on the user's workstation. A client-side script might contain instructions for the browser to handle user interactivity. These instructions might be to change the look or content of the Web page based on the user inputs. Examples include displaying a welcome page with the username, displaying date and time, validating that the required user details are filled, and so on.

A JavaScript is either embedded in an HTML page or is separately defined in a file, which is saved with `.js` extension. In client-side scripting, when an HTML is requested, the Web server sends all the required files to the user's computer. The Web browser executes the script and displays the HTML page to the user along with any tangible output of the script.

Figure 12.4 displays the client-side JavaScript.

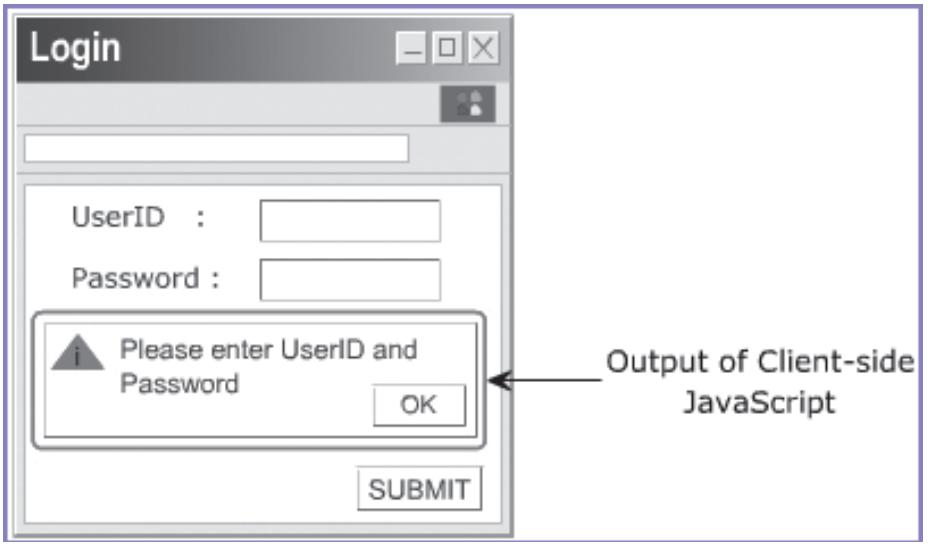



Figure 12.4: Client-side JavaScript

## *12.6 Server-side JavaScript*

A server-side JavaScript (SSJS) is executed by the Web server when an HTML page is requested by a user. The output of a server-side JavaScript is sent to the user and is displayed by the browser. In this case, a user might not be aware that a script was executed on the server to produce the desirable output.

A server-side JavaScript can interact with the database, fetch the required information specific to the user, and display it to the user. This means that server-side scripting fulfills the goal of providing dynamic content in Web pages. Unlike client-side JavaScript, HTML pages using server-side JavaScript are compiled into bytecode files on the server. Compilation is a process of converting the code into machine-independent code. This machine-independent code is known as the bytecode, which is an executable file. The Web server runs this executable to generate the desired output.

Figure 12.5 displays the server-side JavaScript.

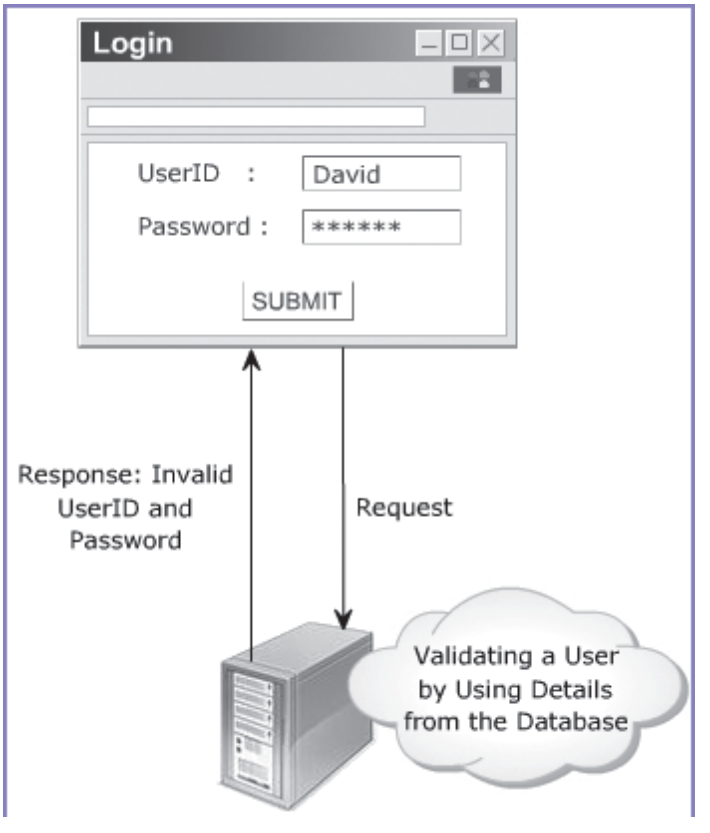



**Figure 12.5: Server-side JavaScript**

## *12.7 <Script> Tag*

The `<script>` tag defines a script for an HTML page to make them interactive. The browser that supports scripts interprets and executes the script specified under the `<script>` tag when the page loads in the browser. You can directly insert a JavaScript code under the `<script>` tag. You can define multiple `<script>` tags either in the `<head>` or in the `<body>` elements of an HTML page. In HTML5, the type attribute specifying the scripting language is no longer required as it is optional.

Code Snippet 1 demonstrates the use of the `<script>` tag.

**Code Snippet 1**:

```
<!DOCTYPE html>
 <html>
   <head>
    <script>
       document.write("Welcome to the Digital World");
    </script>
    </head>
     <body>
      ...
     </body>
 </html>
```

There are two main purposes of the `<script>` tag, which are as follows:

- ➔ Identifies a given segment of script in the HTML page
- ➔ Loads an external script file

## *12.8 Variables in JavaScript*

A variable refers to a symbolic name that holds a value, which keeps changing. For example, age of a student and salary of an employee can be treated as variables. A real life example for variables includes the variables used in algebraic expressions that store values.

In JavaScript, a variable is a unique location in the computer's memory that stores a value and has a unique name. The name of the variable is used to access and read the value stored in it. A variable can store different types of data such as a character, a number, or a string. Therefore, a variable acts as a container for saving and changing values during the execution of the script.

## *12.8.1 Declaring Variables*

Declaring a variable refers to creating a variable by specifying the variable name. For example, you can create a variable named `studName` to store the name of a student. Here, the variable name `studName` is referred to as an identifier. In JavaScript, the var keyword is used to create a variable by allocating memory to it. A keyword is a reserved word that holds a special meaning in JavaScript.

You can initialize the variable at the time of creating the variable or later. Initialization refers to the task of assigning a value to a variable. Once the variable is initialized, you can change the value of a variable as required.

Variables allow keeping track of data during the execution of the script. While referring to a variable, you are referring to the value of that variable. In JavaScript, you can declare and initialize multiple variables in a single statement. Figure 12.6 displays how to declare variables.

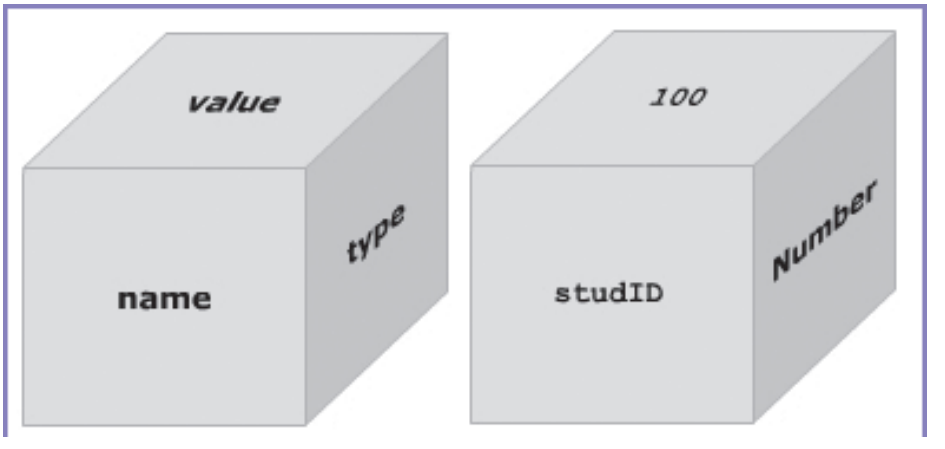



**Figure 12.6: Declare Variables**

The syntax demonstrates how to declare variables in JavaScript.

**Syntax**:

```
var <variableName>;
```

where,

`var`: Is the keyword in JavaScript.

`variableName`: Is a valid variable name.

The syntax demonstrates how to initialize variables in JavaScript.

**Syntax**:

```
<variableName> = <value>;
```

where,

`=`: Is the assignment operator used to assign values.

`value`: Is the data that is to be stored in the variable.

The syntax demonstrates how to declare and initialize multiple variables in a single statement, which are separated by commas.

**Syntax**:

```
var <variableName1> = <value1>, <variableName2> = <value2>;
```

Code Snippet 2 declares two variables namely, `studID` and `studName` and assign values to them.

**Code Snippet 2**:

```
var studID;
var studName;
studID = 50;
studName = "David Fernando";
```

This code assigns values to `studID` and `studName` variables by using the assignment operator `(=)`. The value named `David Fernando` is specified within double quotes.

Code Snippet 3 demonstrates how to declare and initialize multiple variables in a single statement in JavaScript.

**Code Snippet 3**:

```
var studName = David, studAge = 15;
```

## *12.8.2 Variable Naming Rules*

You cannot refer to a variable until it is created in JavaScript. JavaScript is a case-sensitive language. This means that if you specify `X` and `x` as variables, both of them are treated as two different variables. Similarly, in JavaScript, there are certain rules, which must be followed while specifying variables names. These rules for a variable name are as follows:

- Can consist of digits, underscore, and alphabets.
- Must begin with a letter or the underscore character.
- Cannot begin with a number and cannot contain any punctuation marks.
- Cannot contain any kind of special characters such as +, *, %, and so on.
- Cannot contain spaces.
- Cannot be a JavaScript keyword.

It is recommended to give meaningful names to variables such that the name determines the kind of data stored in the variable.

## *12.9 Data Types in JavaScript*

A Web page designer can store different types of values such as numbers, characters, or strings in variables. However, the Web page designer must know what kind of data a particular variable is expected to store. To identify the type of data that can be stored in a variable, JavaScript provides different data types.

A Web page designer need not specify the data type while declaring variables. Due to this, JavaScript is referred to as the loosely typed language. This means that a variable holding a number can also hold a string value later. The values of variables are automatically mapped to their data types when the script is executed in the browser.

Data types in JavaScript are classified into two broad categories namely, primitive and composite data types. Primitive data types contain only a single value, whereas the composite data types contain a group of values.

### *12.9.1 Primitive Data Types*

A primitive data type contains a single literal value such as a number or a string. A literal is a static value that you can assign to variables.

Table 12.2 lists the primitive data types.

| Primitive Data Type | Description |
|---|---|
| `boolean` | Contains only two values namely, true or false |
| `null` | Contains only one value namely, null. A variable of this value specifies that the variable has no value. This null value is a keyword and it is not the same as the value, zero |
| `number` | Contains positive and negative numbers and numbers with decimal point. Some of the valid examples include 6, 7.5, -8, 7.5e-3, and so on |
| `string` | Contains alphanumeric characters in single or double quotation marks. The single quotes is used to represent a string, which itself consists of quotation marks. A set of quotes without any characters within it is known as the null string |

**Table 12.2: Primitive Data Types**

### *12.9.2 Composite Data Types*

A composite data type stores a collection of multiple related values, unlike primitive data types. In JavaScript, all composite data types are treated as objects. A composite data type can be either predefined or user-defined in JavaScript.

Table 12.3 lists the composite data types.

| Composite Data Type | Description |
|---|---|
| Objects | Refers to a collection of properties and functions. Properties specify the characteristics and functions determine the behavior of a JavaScript object |
| Functions | Refers to a collection of statements, which are instructions to achieve a specific task |
| Arrays | Refers to a collection of values stored in adjacent memory locations |

**Table 12.3: Composite Data Types**

## *12.10 Methods*

JavaScript allows you to display information using the methods of the `document` object. The `document` object is a predefined object in JavaScript, which represents the HTML page and allow managing the page dynamically. Each object in JavaScript consists of methods, which fulfills a specific task. There are two methods of the `document` object, which displays any type of data in the browser. These methods are as follows:

- `write()`: Displays any type of data.
- `writeln()`: Displays any type of data and appends a new line character.

The syntax demonstrates the use of `document.write()` method, which allows you to display information in the displayed HTML page.

**Syntax**:

```
document.write("<data>" + variables);
```

where,

`data`: Specifies strings enclosed in double quotes.

`variables`: Specify variable names whose value should be displayed on the HTML page.

The syntax demonstrates the use of `document.writeln()` method, which appends a new line character.

**Syntax**:

```
document.writeln("<data>" + variables);
```

Code Snippet 4 demonstrates the use of `write()` method.

**Code Snippet 4**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title> JavaScript language </title>
  <script>
   document.write("<p> JavaScript:");
      document.writeln("is a scripting");
      document.write("and a case-sensitive language.");
  </script>
  </head>
  <p>
  JavaScript: is a scripting and a case-sensitive language.
  </p>
</html>
```

The code uses the `writeln()` method to display the text after the colon without leaving a space. It finally appends a new line character after the text. Then, the text within the `write()` method is displayed on the same line after leaving a space.

The same paragraph is displayed in the body of the HTML page. Note that the text in the `p` element appears on different lines. In HTML, the text on the second line, and a case sensitive language will not be displayed in the new line in the browser even though the **ENTER** key is pressed while writing the code. Rather, it will be displayed on the same line with a space. The `writeln()` method also follows this same format.

Figure 12.7 displays the use of `write()` and `writeln()` methods.

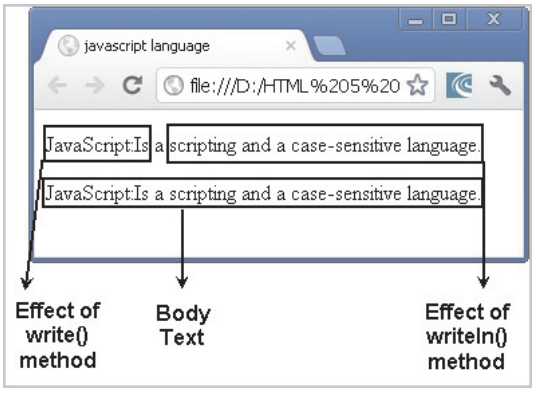



Figure 12.7: Use of write() and writeln() Methods

## *12.11 Using Comments*

A Web page designer might code complex script to fulfill a specific task. In JavaScript, a Web page designer specifies comments to provide information about a piece of code in the script. Comments describe the code in simple words so that somebody who reads the code can understand the code. Comments are small piece of text that makes the program more readable. While the script is executed, the browser can identify comments as they are marked with special characters and do not display them.

JavaScript supports two types of comments. These are as follows:

➔ **Single-line Comments**

Single-line comments begin with two forward slashes `(//)`. You can insert single-line comments as follows:

```
// This statement declares a variable named num.
     var num;
```

➔ **Multi-line Comments**

Multi-line comments begin with a forward slash followed by an asterisk `(/*)` and end with an asterisk followed by a forward slash `(*/)`. You can insert multiple lines of comments as follows:

```
/* This line of code
declares a variable */
var num;
```

## *12.12 Escape Sequence Characters*

An escape sequence character is a special character that is preceded by a backslash (`\`). Escape sequence characters are used to display special non-printing characters such as a tab space, a single space, or a backspace. These non-printing characters help in displaying formatted output to the user to maximize readability.

The backslash character specifies that the following character denotes a non-printing character. For example, `\t` is an escape sequence character that inserts a tab space similar to the **Tab** key of the keyboard. In JavaScript, the escape sequence characters must always be enclosed in double quotes.

There are multiple escape sequence characters in JavaScript that provides various kind of formatting.

Table 12.4 lists the escape sequence characters.

| Escape Sequence | Non-Printing Character |
|---|---|
| `\b` | Back space |
| `\f` | Form feed |
| `\n` | New line |
| `\r` | Carriage return |
| `\t` | Horizontal tab |
| `\'` | Single quote |
| `\"` | Double quote |
| `\\` | Backslash |
| `\aaa` | Matches a Latin-1 encoding character using octal representation, where aaa are three octal numbers. For example, \251 represents the copyright symbol |
| `\xaa` | Matches a Latin-1 encoding character using hexadecimal representation, where aa are two hexadecimal numbers. For example, \x61 represents the character 'a' |
| `\uaaaa` | Represent the Unicode encoding character, where aaaa are four hexadecimal numbers. For example, the character \ u0020 represents a space |

**Table 12.4: Escape Sequence Characters**

Code Snippet 5 demonstrates the use of escape sequence characters in JavaScript.

**Code Snippet 5**:

```
<script>
 document.write("You need to have a \u0022credit card\u0022, if you
     want to shop on the \'Internet\'.");
</script>
```

The code uses a Unicode encoding character namely, `\u0022`, which represents double quotes. These open and close double quotes will contain the term `credit card`. Similarly, the word `Internet` will be placed in single quotes. The single quotes are specified using the backslash character.

**Note** - An encoding scheme specifies how to represent character data in terms of their acceptable range, maximum number of characters, and patterns. Unicode is a character set that contains all the international characters required for processing information. Latin 1 is the encoding scheme for English and Western European languages on the Internet.

## *12.13 Built-in Functions*

A function is a piece of code that performs some operations on variables to fulfill a specific task. It takes one or more input values, processes them, and returns an output value. JavaScript provides built-in functions that are already defined to fulfill a certain task. Table 12.5 lists the built-in functions.

| Function | Description | Example |
|---|---|---|
| `alert()` | Displays a dialog box with some information and OK button | `alert("Please fill all the fields of the form");`<br>Displays a message box with the instruction |
| `confirm()` | Displays a dialog box with OK and Cancel buttons. It verifies an action, which a user wants to perform | `confirm("Are you sure you want to close the page?");`<br>Displays a message box with the question |
| `parseInt()` | Converts a string value into a numeric value | `parseInt("25 years");` |
| `parseFloat()` | Converts a string into a number with decimal point | `parseFloat("10.33");`<br>Returns 10.33 |
| `eval()` | Evaluates an expression and returns the evaluated result | `eval("2+2");`<br>Returns 4 |
| `isNaN()` | Checks whether a value is not a number | `isNan("Hello");`<br>Returns true |
| `prompt()` | Displays a dialog box that accepts an input value through a text box. It also accepts the default value for the text box. | `prompt("Enter your name", "Name");`<br>Displays the message in the dialog box and Name in the text box. |

**Table 12.5: Built-in Functions**

**Note** - The `\n` character, when used in the `alert()` function, prints the information on a new line. This does not happen when the `\n` character is used with the `write` methods of the `document` object.

Code Snippet 6 demonstrates the use of some of the built-in functions in JavaScript. It performs the addition operation using JavaScript.

**Code Snippet 6**:

```
<!DOCTYPE HTML>
 <html>
   <head>
   <title> JavaScript language </title>
    <script>
       var value = "";
        var numone = prompt("enter first value to perform the
                     multiplication operation", value);
         var numtwo = prompt("enter second value to perform the
                     multiplication operation", value);
         var result = eval(numone *  numtwo);

       document.write("The result of multiplying: " + numone + "
         and " +
         numtwo + " is: " + result + "." );

    </script>
    </head>
 </html>
```

In the code, it takes the first value from the user and stores in the `numOne` variable. Then, it takes the second value from the user and stores in the `numTwo` variable. It multiplies the values and stores the output in the `result` variable and then displays the output on the Web page.

Figure 12.8 displays the input first number.

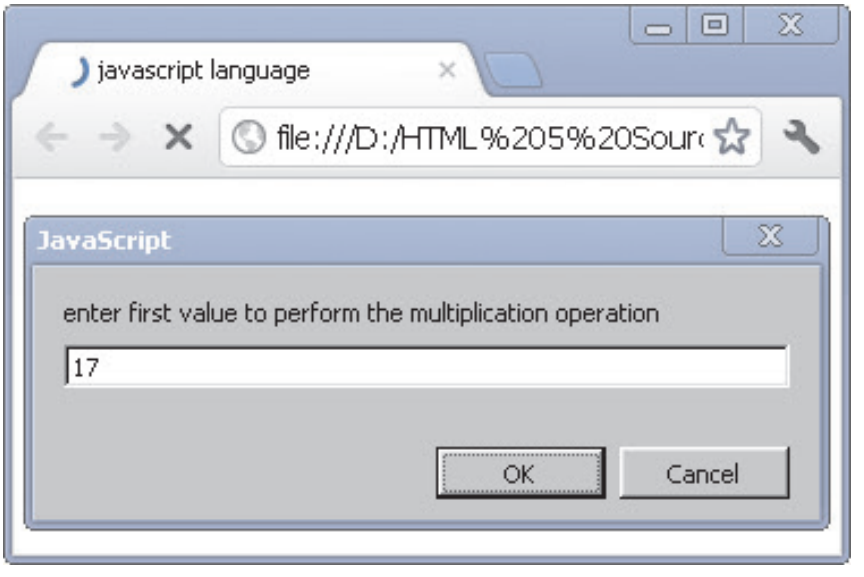



Figure 12.8: Input First Number

Figure 12.9 displays the input second number.

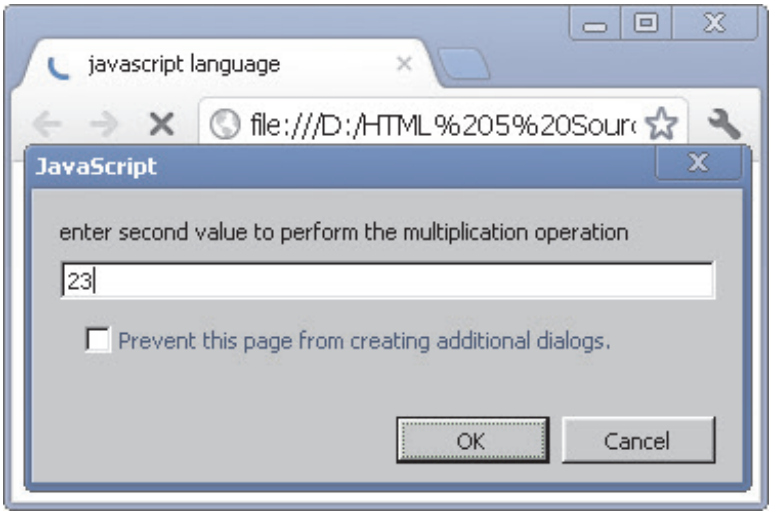



Figure 12.9: Input Second Number

Figure 12.10 displays the result.

Figure 12.10: Result

## *12.14 Events*

Consider a scenario where you want to design an Employee registration Web form. This form allows the users to fill in the appropriate details and click the submit button. When the user clicks the submit button, the form data is submitted to the server for validation purposes. In this case, when the user clicks the button, an event is generated. The submission of form refers to the action performed on click of the button.

An event occurs when a user interacts with the Web page. Some of the commonly generated events are mouse clicks, key strokes, and so on. The process of handling these events is known as event handling. Figure 12.11 displays the event.

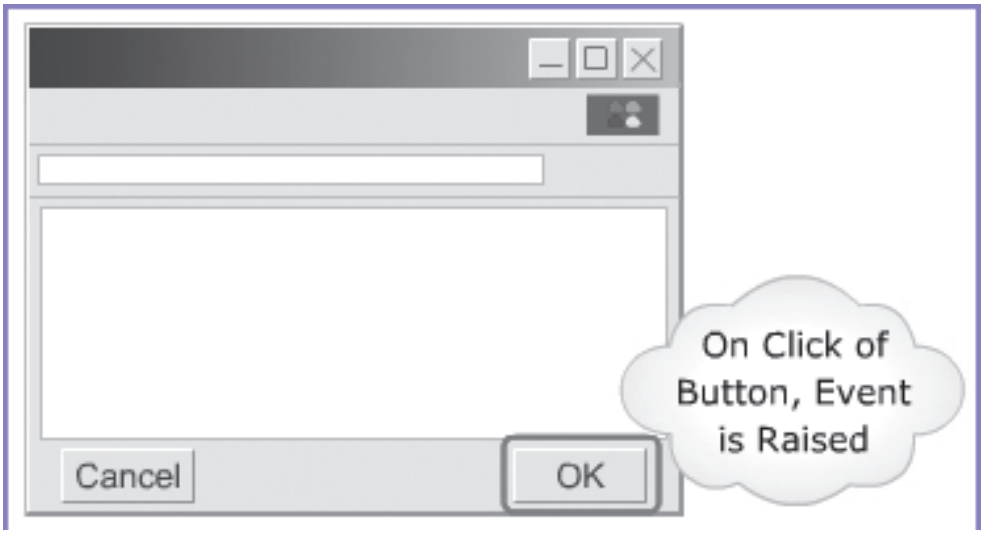



**Figure 12.11: Event**

### *12.14.1 Event Handling*

Event handling is a process of specifying actions to be performed when an event occurs. This is done by using an event handler. An event handler is a scripting code or a function that defines the actions to be performed when the event is triggered.

When an event occurs, an event handler function that is associated with the specific event is invoked. The information about this generated event is updated on the `event` object. The `event` object is a built-in object, which can be accessed through the `window` object.

It specifies the event state, which includes information such as the location of mouse cursor, element on which an event occurred, and state of the keys in a keyboard.

Figure 12.12 displays the event handling.

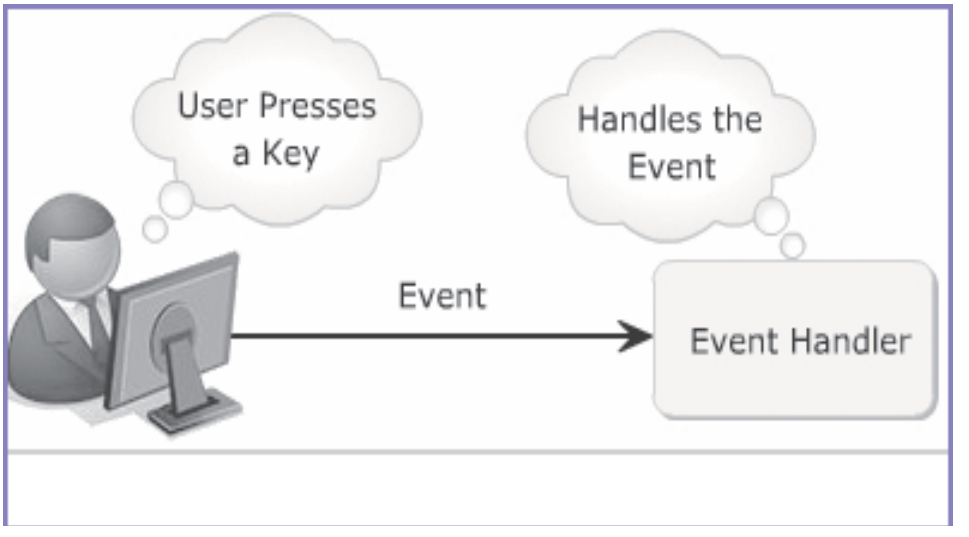



**Figure 12.12: Event Handling**

## *12.14.2 Event Bubbling*

Event bubbling is a mechanism that allows you to specify a common event handler for all child elements. This means that the parent element handles all the events generated by the child elements. For example, consider a Web page that consists of a paragraph and a table. The paragraph consists of multiple occurrences of italic text. Now, you want to change the color of each italic text of a paragraph when the user clicks a particular button. Instead of declaring an event handler for each italic text, you can declare it within the P element. This allows you to apply colors for all the italic text within the paragraph. This helps in reducing the development time and efforts since it minimizes the code. Figure 12.13 displays the event bubbling.

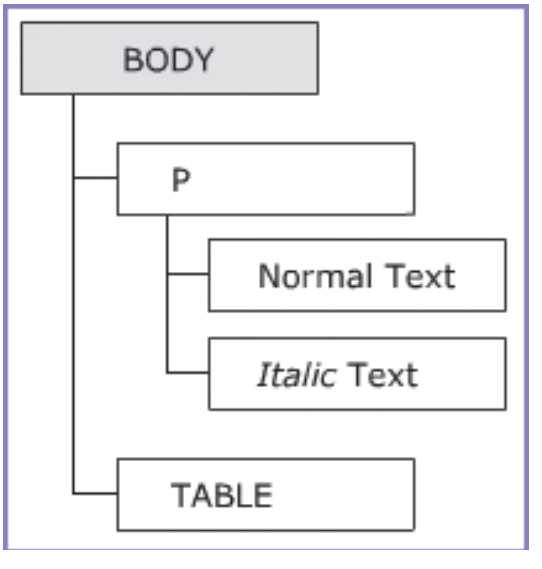



**Figure 12.13: Event Bubbling**

## *12.14.3 Life Cycle of an Event*

An event's life starts when the user performs an action to interact with the Web page. It finally ends when the event handler provides a response to the user's action. The steps involved in the life cycle of an event are as follows:

- The user performs an action to raise an event.
- The event object is updated to determine the event state.
- The event is fired.
- The event bubbling occurs as the event bubbles through the elements of the hierarchy.
- The event handler is invoked that performs the specified actions.

## *12.14.4 Keyboard Events*

Keyboard events are the events that occur when a key or a combination of keys are pressed or released from a keyboard. These events occur for all keys of a keyboard.

The different keyboard events are as follows:

- **Onkeydown**

  Occurs when a key is pressed down.

- **Onkeyup**

  Occurs when the key is released.

- **Onkeypress**

  Occurs when a key is pressed and released.

Code Snippet 7 demonstrates how to create a JavaScript code that defines the event handlers.

**Code Snippet 7**:

```
function numericonly()
 {
   if(!event.keyCode >=48 && event.keyCode<=57))
     event.returnValue=false;
 }
function countWords()
 {
    var message = document.getElementByID('txtMessage').value;
    message= message.replace(/\s+/g, ' ');
    var numberOfWords = message.split(' ').length;
    document.getElementById('txtTrack').value = words Remaining:
    ' +
    eval(50 - numberOfWords);
    if(numberOfWords > 50)
    alert("too many words.');
 }
```

In the code, the function `numericOnly()` declares an event handler function, `numericOnly()`. The `event.keyCode` checks if the Unicode character of the entered key is greater than `48` and less than `57`. This checks that only numeric values are entered. It also declares an event handler function, `countWords()`. It retrieves the text specified in the `txtMessage` control. `split()function` splits the specified string when a space is encountered and returns the length after splitting. It also calculates and displays the number of remaining words to complete the count of `50` words. If the number of words is greater than `50`, an alert box is displayed.

## *12.14.5 Mouse Events*

Mouse events occur when the user clicks the mouse button. Table 12.6 lists the mouse events.

| Events | Description |
|---|---|
| `onmousedown` | Occurs when the mouse button is pressed |
| `onmouseup` | Occurs when the mouse button is released |
| `onclick` | Occurs when the mouse button is pressed and released |
| `ondblclick` | Occurs when the mouse button is double-clicked |

| Events | Description |
|---|---|
| onmousemove | Occurs when the mouse pointer is moved from one location to other |
| onmouseover | Occurs when the mouse pointer is moved over the element |
| onmouseout | Occurs when the mouse pointer is moved out of the element |

Table 12.6: Mouse Events

Code Snippet 8 demonstrates the use of different mouse events.

**Code Snippet 8**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <title> Reservation </title>
  <script src="form.js">
  </script>
 </head>
      <body>
       <h2> Hotel Reservation Form</h2>
       <form id="frmreservation">
      <table>
        <tr>
    <td> <label for="txtName">Name:</label></td>
    <td> <input id="txtName" type="text" /></td>
        </tr>
        <tr>
        <td> Arrival Date: </td>
        <td> <input id="txtArrival" type="text" /></td>
```

```
      </tr>
      <tr>
      <td> Departure Date: </td>
      <td><input id="txtDepartture" type="text" /></td>
      </tr>
      <tr>
      <td> Number of Person: </td>
      <td><input id="txtPerson" type="text" maxlength="3"
     size="3"></td>
      </tr>
      <tr>
      <td><img id="imgSubmit" width="120px" height="30px"
     src="submit.jpg" alt="Submit",
         onmousedown="showImage(this, 'submitdown.jpg');"
         onmouseup="showImage(this,
     'submit.jpg');",onclick="frmReservation.submit();"/>
      </td>
         <td><img id="imgSubmit" width="120px" height="30px"
      src="reset.jpg" alt="Reset",
         onmousedown="showImage(this, 'resetdown.jpg');"
         onmouseup="showImage(this,
      'reset.jpg');",onclick="frmReservation.reset();"/>
       </td>
         </tr>
     </table>
     </form>
     </body>
</html>
```

In the code, an image is displayed when Submit button is clicked.

It will also display the `submit.jpg` image when the mouse is released from Submit button. It also submits the form data when the Submit button is clicked. Further it displays the image when Reset button is clicked and it displays the `reset.jpg` image when the mouse is released from Reset button. It will reset the form data when the Reset button is clicked.

Code Snippet 9 demonstrates the loading of images in a JavaScript file.

**Code Snippet 9**:

```
function showImage(object,url)
{
 object.src=url;
}
```

Figure 12.14 displays the output of mouseup.

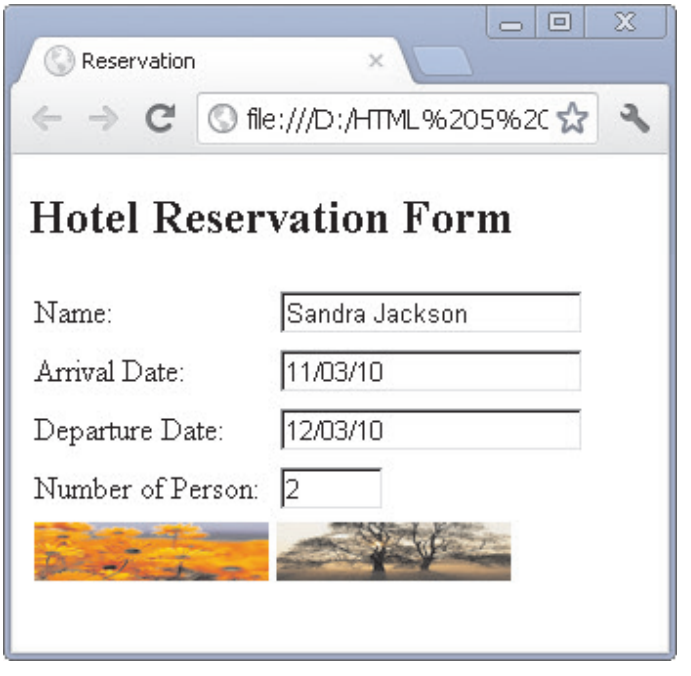



**Figure 12.14: Output of MouseUp**

Figure 12.15 displays the output on mousedown.

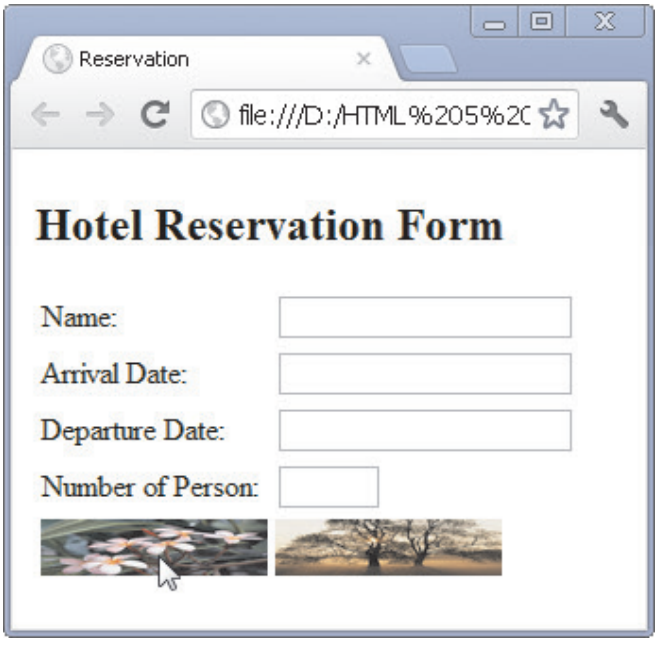



Figure 12.15: Output On MouseDown

## *12.14.6 Focus and Selection Events*

The focus events determine the activation of various elements that uses the `input` element. It allows you to set or reset focus for different `input` elements. The selection events occur when an element or a part of an element within a Web page is selected. Table 12.7 lists the focus and selection events.

| Events | Description |
|---|---|
| `onfocus` | Occurs when an element receives focus |
| `onblur` | Occurs when an element loses focus |
| `onselectstart` | Occurs when the selection of an element starts |
| `onselect` | Occurs when the present selection changes |
| `ondragstart` | Occurs when the selected element is moved |

Table 12.7: Focus and Selection Events

Code Snippet 10 demonstrates the use of focus and selection events.

**Code Snippet 10**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title> Reservation </title>
  <script>
      function showStyle (field)
      {
       field.style.backgroundColor = '#FFFFCC';
      }
      function hideStyle (field)
      {
       field.style.backgroundColor = '#FFFFFF';
      }
      function setFontStyle (field)
      {
       field.style.fontWeight = 'bold';
       field.style.fontFamily = 'Arial';
      }
 </script>
 </head>
      <body>
      <h2> Feedback Form</h2>
      <form id="frmreservation">
     <table>
       <tr>
       <td> <label for="txtName">Name:</label></td>
       <td> <input id="txtName" type="text" onfocus="showStyle (this
              );" onblur="hideStyle (this);" onselect=setFontStyle (
         this); />
```

```
   </td>
   </tr>
   <tr>
   <td><label for="txtEmail">E-mail:</label></td>
   <td><input id="txtEmail" type="text" onfocus="showStyle(this);"
      onblur="hideStyle(this);" onselect=setFontStyle(this); />
   </td>
   </tr>
   <tr>
   <td><label for="txtComment">Comment:</label></td>
   <td><textarea id="txtComment" cols="15" rows="3"
     onfocus="showStyle(this);" onblur="hideStyle(this);"
      onselect=setFontStyle(this);></textarea>
   </td>
  </tr>
  <tr>
  <td><input id="btnSubmit" type="button" type="button"
      value="Submit" /></td>
  <td><input id="btnReset" type="reset" /></td>
  </tr>
  </table>
   </form>
 </body>
</html>
```

In the code, a specified style is displayed when the element receives and loses focus. It also displays the specified font style when the element is selected. It also declares an event handler function and specifies the background color for the field. It sets the font style for text to bold and the text should appear in `Arial` font.

Figure 12.16 displays the focus and selection events.

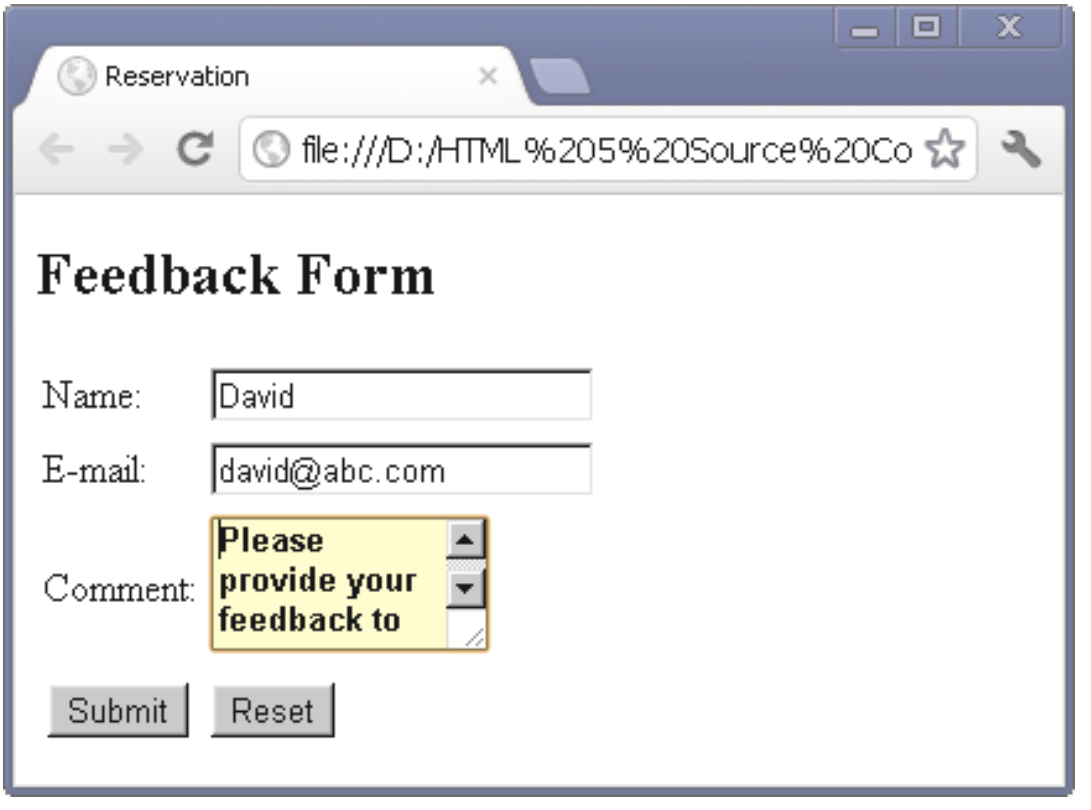



Figure 12.16: Focus and Selection Events

## *12.15 jQuery*

jQuery is a short and fast JavaScript library developed by John Resig in 2006 with a wonderful slogan: **Write less and do more**. It simplified the client side scripting of HTML. jQuery also simplifies HTML files animation, event handling, traversing, and developing AJAX based Web applications. It helps in rapid Web application development. jQuery is designed for simplifying several tasks by writing lesser code. The following are the key features supported by jQuery:

- **Event Handling**: jQuery has a smart way to capture a wide range of events, such as user clicks a link, without making the HTML code complex with event handlers.

- **Animations**: jQuery has many built-in animation effects that the user can use while developing their Web sites.

- **DOM Manipulation**: jQuery easily selects, traverses, and modifies DOM by using the cross-browser open source selector engine named Sizzle.

- **Cross Browser Support**: jQuery has a support for cross-browser and works well with the following browsers:
  - Internet Explorer 6 and above
  - Firefox 2.0 and above
  - Safari 3.0 and above
  - Chrome
  - Opera 9.0 and above

- **Lightweight**: jQuery has a lightweight library of 19 KB size.

- **AJAX Support**: jQuery helps you to develop feature-rich and responsive Web sites by using AJAX technologies.

- **Latest Technology**: jQuery supports basic XPath syntax and CSS3 selectors.

## *12.15.1 Using jQuery Library*

There is an easy way to use jQuery library. To work with jQuery perform the following steps:

1. Download the jQuery library from the **http://jquery.com/** Web site
2. Place the **jquery-1.7.2.min.js** file in the current directory of the Web site

The user can include jQuery library in their file.

Code Snippet 11 shows how to use a jQuery library.

**Code Snippet 11**:

```
<!DOCTYPE HTML>
<html>
<head>
<title>The jQuery Example</title>
  // Using jQuery library
<script src="jquery-1.7.2.min.js">
  // The user can add our JavaScript code here
</script>
```

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

## *12.15.2 Calling jQuery Library Functions*

Users can do many tasks while jQuery is reading or manipulating the DOM object. The users can add the events only when the DOM object is ready. If the user wants the event on their page then the user has to call the event in the `$(document).ready()` function. All the content inside the event will be loaded as soon as the DOM is loaded but before the contents of the page are loaded. The users also register the ready event for the document. Place the `jquery-1.7.2.min.js` file in the current directory and specify the location of this file in the `src` attribute.

Code Snippet 12 shows how to call jQuery library function and ready event in DOM.

**Code Snippet 12**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title>The jQuery Example</title>
  <script src=" jquery-1.7.2.min.js">
  </script>
  <script>
   $(document).ready(function() {
   $("div").click(function() {
    alert("Welcome to the jQuery world!");
   });
   });
  </script>
 </head>
 <body>
```

```
<div id="firstdiv">
  Click on the text to view a dialog box.
 </div>
 </body>
</html>
```

The code includes the jQuery library and also registers the ready event for the document. The ready event contains the click function that calls the click event.

Figure 12.17 displays the output of jQuery.

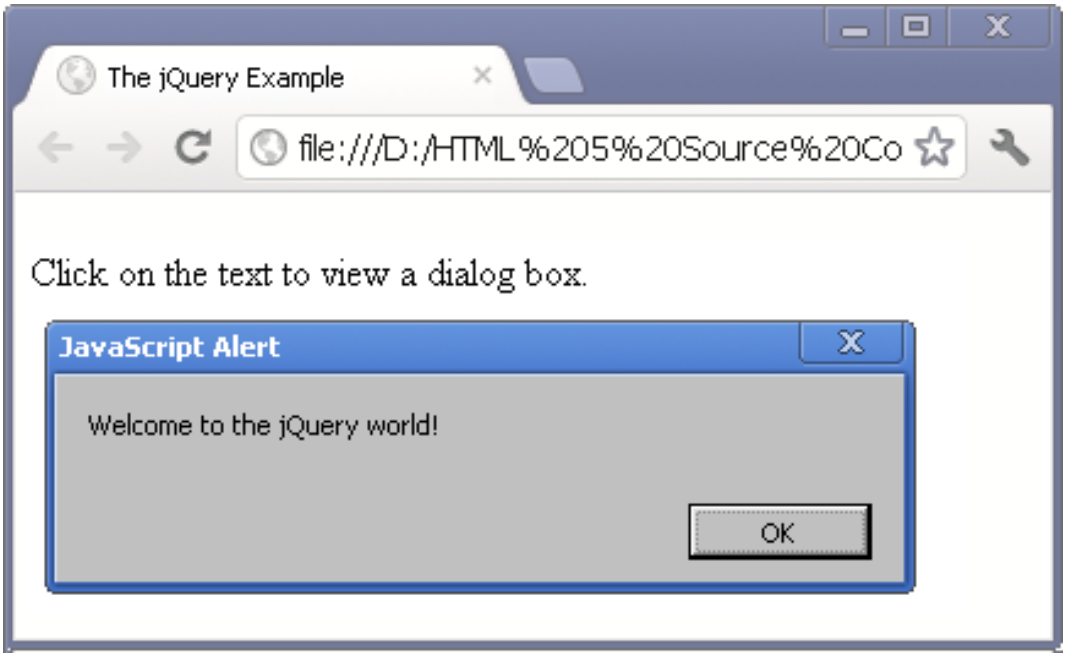



**Figure 12.17: Output of jQuery**

## *12.16 jQuery Mobile*

jQuery mobile is a Web User Interface (UI) development framework that allows the user to build mobile Web applications that works on tablets and smartphones. The jQuery mobile framework provides many facilities that include XML DOM and HTML manipulation and traversing, performing server communication, handling events, image effects, and animation for Web pages. The basic features of jQuery mobile are as follows:

➔ **Simplicity**

This framework is easy to use and allows developing Web pages by using markup driven with minimum or no JavaScript.

➔ **Accessibility**

The framework supports Accessible Rich Internet Applications (ARIA) that helps to develop Web pages accessible to visitors with disabilities.

➔ **Enhancements and Degradation**

The jQuery mobile is influenced by the latest HTML5, JavaScript, and CSS3.

➔ **Themes**

This framework provides themes that allow the user to provide their own styling.

➔ **Smaller Size**

The size for jQuery mobile framework is smaller for CSS it is 6KB and for JavaScript library it is 12KB.

For executing the jQuery mobile, you have to download the Opera Mobile Emulator from the **http://www.opera.com/developer/tools/mobile/** Web site. Download the emulator and install it on your machine and then write the code in the Coffee cup editor.
Code Snippet 13 shows an example of a jQuery mobile.

**Code Snippet 13**:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Page Title</title>
  <link rel="stylesheet" href="jquery.mobile-1.0a3.min.css" />
  <script src="jquery-1.5.min.js"></script>
  <script src="jquery.mobile-1.0a3.min.js"></script>
 </head>
 <body>
  <div data-role="page">
  <div data-role="header">
  <h1>Car Rental</h1>
  </div>
  <div data-role="content">
```

```
<p>Choose from the listed car models</p>
 <ul data-role="listview" data-inset="true">
     <li><a href="#">Ford</a></li>
     <li><a href="#">Ferrari</a></li>
     <li><a href="#">BMW</a></li>
     <li><a href="#">Toyota</a></li>
     <li><a href="#">Mercedes-Benz</a></li>
 </ul>
 </div>
 <div data-role="footer">
  <h4>&copy; DriveCars 2012.</h4>
 </div>
 </div>
 </body>
 </html>
```

The jQuery mobile application should have the following three files:

- CSS file
- jQuery library
- jQuery Mobile library

In the code, three files are included the CSS (`jquery.mobile-1.0a3.min.css`), jQuery library (`jquery-1.5.min.js`), and the jQuery mobile library (`jquery.mobile-1.0a3.min.js`). A user can also download the jQuery libraries from **http://code.jquery.com/** Web site.

The jQuery Mobile takes HTML tags and renders them on mobile devices. To work with this, HTML has to make use of data attributes. jQuery use these attributes as indicators for rendering it on the Web pages. jQuery also looks for div using a particular data-role values such as page, content, header, and footer are used in this code. There are multiple div blocks added to the code for page, content, header, and footer. Similarly, to display the different car models a data-role listview is added to enhance the look and feel of the mobile Web page.

A user need to install the Opera Mobile Emulator from the Opera Web site.

After installing the **Opera Mobile Emulator**, perform the following steps to apply settings to the emulator:

1. Select **All Programs ➔ Opera Mobile Emulator ➔ Opera Mobile Emulator**.

   The Opera Mobile Emulator dialog box will be displayed.

2. In the **Profile** tab, select the **Samsung Galaxy Tab**.
3. In the **Resolution** drop-down, select the **WVGA Portrait(480x800)**.
4. Click **Update**.
5. Click **Launch**. The **Samsung Galaxy** tab is displayed.

For executing the jQuery mobile code given in Code Snippet 12 in the **CoffeeCup** editor, perform the following steps:

1. Add the **Opera Mobile Emulator** in the **CoffeeCup** editor by clicking **Tools ➔ Additional Browsers ➔ Test with Additional Browser 1** and give the location of the **Opera Mobile Emulator** installed on your system. After adding the emulator, you can see the emulator added to the additional browsers list.
2. Open the jQuery file in the **CoffeeCup** editor and save.
3. Click **Tools ➔ Additional Browser ➔ Test with Additional Browser 1**.

   Figure 12.18 displays the **Opera Mobile Emulator**.

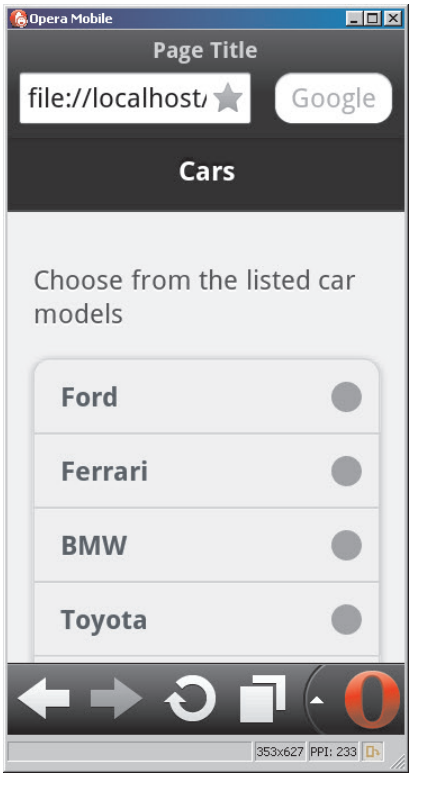



Figure 12.18: Opera Mobile Emulator

## *12.17 Check Your Progress*

1. Identify the following types of scripting languages.

| | | | |
|---|---|---|---|
| (A) | Client-side | (C) | C |
| (B) | C++ | (D) | Server-side |

2. _____ is a scripting language, which can be executed on the client-side and on the server-side. A client-side JavaScript (CSJS) is executed by the browser on the user's workstation.

| | | | |
|---|---|---|---|
| (A) | Java | (C) | jQuery |
| (B) | VBScript | (D) | JavaScript |

3. The _____ tag defines a script for an HTML page to make them interactive.

| | | | |
|---|---|---|---|
| (A) | `<script>` | (C) | `<title>` |
| (B) | `<head>` | (D) | `<html>` |

4. Which of the following is a piece of code that performs some operations on variables to fulfill a specific task?

| | | | |
|---|---|---|---|
| (A) | Code | (C) | Function |
| (B) | Script | (D) | Variable |

5. Which of the following is a process of specifying actions to be performed when an event occurs?

| | | | |
|---|---|---|---|
| (A) | Event Handling | (C) | Scripting |
| (B) | Event Bubbling | (D) | Function |

## *12.17.1 Answers*

| | |
|---|---|
| 1. | A, D |
| 2. | D |
| 3. | A |
| 4. | C |
| 5. | A |

# Summary

- Scripting refers to a series of commands that are interpreted and executed sequentially and immediately on an occurrence of an event.
- JavaScript is a scripting language, which can be executed on the client-side and on the server-side. A client-side JavaScript is executed by the browser on the user's workstation.
- A variable refers to a symbolic name that holds a value, which keeps changing.
- A primitive data type contains a single literal value such as a number or a string.
- A function is a piece of code that performs some operations on variables to fulfill a specific task.
- Event handling is a process of specifying actions to be performed when an event occurs.
- Event bubbling is a mechanism that allows you to specify a common event handler for all child elements.
- jQuery mobile is a Web User Interface development framework that allows the user to build mobile Web applications that works on tablets and smartphones.

# *Try it Yourself*

1. Joaquina works for an online shopping Web site named, Zambia Developers and is headquartered at Tokyo, Japan. She has created a Web site on HTML5. The Web site contains a registration form where the customer has to enter their details. Joaquina wants to add some validations on the registration form so that the details entered by the customers are in a proper format such as the name, mobile number, and so on. So she has decided to add JavaScript code for validation of the data present in Web site. Help her to develop the application.

2. Robert is working for Jazz Developers Private Ltd. located at Washington, USA. He has worked on many mobile projects. Recently, their company has received an order for creating mobile application for online shopping Web site. The Web site should be created in such a manner that it is accessible by all mobile devices so that the customer can order the products online. Robert has decided to create the application using jQuery with HTML5. Help him to develop the application.

# Session - 13
## Basics of jQuery

Session Objectives:

This session introduces the jQuery library along with its basic concepts and implementation in Bootstrap. It also describes the different jQuery selectors and various ways of creating functions. The session also explains JavaScript Object Notation (JSON) along with its usage in jQuery. It further explains conversions to JSON and event delegation in jQuery.

At the end of this session, you will be able to:

- → Explain the basics of jQuery and its usage with Bootstrap
- → Outline selectors and functions in jQuery
- → Describe JSON and its usage in jQuery
- → Explain conversions to JSON
- → Describe the concept of event delegation and its usage in jQuery

## *13.1 Introduction to jQuery and Integrating with Bootstrap*

jQuery refers to a lightweight and concise JavaScript (JS) library that aims to make JS programming easier and swifter. It is a feature-rich and open-source library that simplifies several complicated tasks such as manipulating and traversing an HTML/CSS page, handling events, animating objects, and making Asynchronous JavaScript and XML (AJAX) calls.

The library offers an easy-to-use Application Programming Interface (API) that is compatible across several browsers. It wraps several common tasks into built-in methods. A Web developer can invoke the desired method by writing just a single line of JS code instead of writing several lines of it. This is how jQuery accomplishes the purpose of 'get more by writing less'.

### *13.1.1 Benefits of Using jQuery over JS*

There are many JS frameworks available but jQuery is perhaps the most extendable one. Big brands such as Google, IBM, and Microsoft are using it. Following are the benefits of using jQuery over JS or other JS libraries:

- **Unobtrusive JS**: jQuery helps in designing Webpages through unobtrusive JS, an approach that separates functionality between the presentation and structure of a Webpage. In other words, inserting an event code in the script tag keeps the behavior separate from the Webpage's structure or content.
- **Lightweight**: The reduced library size is in Kilobytes (KB), which is perhaps smaller than that of a digital photo file. The jQuery version 2.x is smaller than the version 1.x. The former version is more widely used, as it supports Internet Explorer (IE) 6/7/8. The browser needs to download it once and cache it for using it in all Webpages of a site. In case a Web developer relies upon a Content Delivery Network (CDN), the browser tends to cache the library across several Websites instead of downloading for each.
- **Ease of Use**: This is due to the simple yet robust syntax and significantly reduced lines of JS code. The syntax makes it simple to select the page elements that need a change. For this, the library uses CSS3 selector specification, which eliminates the need to learn the jQuery syntax.
- **Big and Focused Library**: Unlike other JS libraries, Web developers can perform a variety of functions. Further, to make JS tasks easier, jQuery offers several utility functions.

  With these functions, Web developers can parse data, trim strings, and search for array elements based on a filter. However, the library is focused, which means it contains only essential features. A Webpage downloads only those features that it requires.
- **Extensibility**: jQuery comes with a plugin framework due to which it is simple to extend jQuery. The framework includes both third-party and official plugins. So, if a feature does not exist in the library, a Web developer may include it through a plugin.
- **Browser Compatibility**: Just as in few other JS libraries, jQuery flawlessly resolves cross-browser issues. JS implementations tend to vary amongst browsers, which can take much time of Web developers to get the code working across browsers. However, the makers of jQuery have already implemented the code to display Webpages seamlessly across all major browsers.

- **Strong Community**: Although a new library, jQuery has an exclusive community through which the dedicated developers tend to improve its functionality. Through this community, there are several plugins available to accelerate the Web development process. Most of the plugins are secured and efficient.
- **AJAX Support**: jQuery easily allows creating AJAX templates for gaining the benefits of smoother interfaces and efficient loading. A user can perform an action on a Webpage without reloading the whole page.
- **Comprehensive Documentation**: Official Website of jQuery has in-depth documentation and tutorials, which helps even a beginner in programming to start easily.

### *13.1.2 Using Bootstrap and jQuery Together*

Web developers can also jQuery with Bootstrap through the jQuery UI Bootstrap theme, as shown in Figure 13.1.

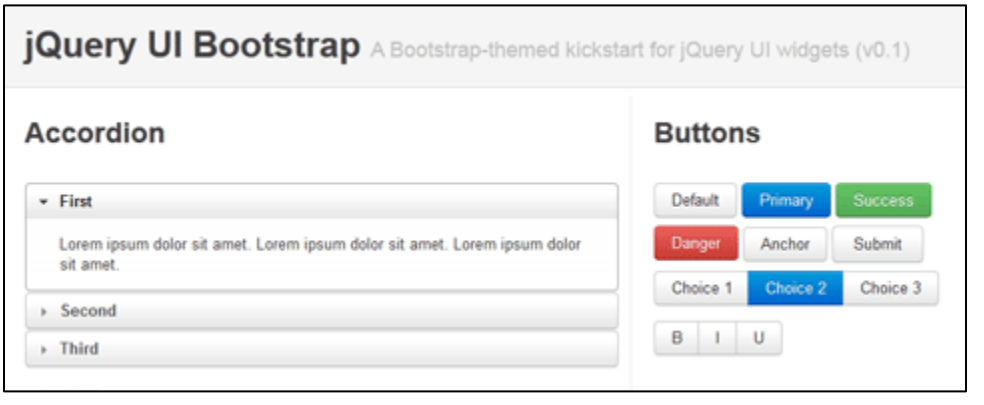



**Figure 13.1: jQuery UI Bootstrap Theme**

***Image Courtesy**: https://webresourcesdepot.com*

The theme allows using Bootstrap-themed widgets without breaking the appearance of components. Originally, Bootstrap and jQuery cannot exist simultaneously due to which using them together triggers conflicts with JS and CSS styles and classes. However, the new theme comes with the relevant JS and CSS for designing a Web application using jQuery and Bootstrap. It also offers the Bootstrap theme for several third-party jQuery widgets for designing user interfaces. Following are the steps to use the theme:

1. Download the archive from the Internet.
2. Unzip it.
3. Look for **custom-theme** folder within the **css** folder of the unzipped package.
4. Move the **custom-theme** folder to the **css** folder of the Web application for which this theme is to be used.
5. Add the relevant .css files to the Website's layout.

### *13.1.3 Using jQuery in Web Applications*

The jQuery library is a JS file that a Web developer uses for programming in jQuery for developing a Web application. There are two ways to use the file, which are as follows:

- Download the library file from jQuery.com and refer it in the HTML code just as a Web developer would do for any other JS file. Save the downloaded file in the directory of Webpages.
- Include the library into the code by referring to a CDN, such as Google and Microsoft. Code Snippet 1 shows how to refer to the Microsoft CDN.

  **Code Snippet 1**:

  ```
  <head>
  <script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
  3.2.0.min.js"></script>
  </head>
  ```

### *13.1.4 Understanding the jQuery Syntax*

With jQuery, a Web developer can quickly find and select an HTML element and trigger the desired action or actions on it. For this purpose, the makers of jQuery have offered a customized syntax, which is as follows:

```
$(selector).action()
```

where,

- *$*: Is the sign that indicates the use of jQuery and is the jQuery identifier.
- *(selector)*: Is used to find and select the HTML element or elements.
- *action()*: Is an action that jQuery should perform on the selected element or elements.

Following are a few samples that indicate how the syntax is used:

- *$(this).hide()* for hiding the element that is selected at present
- *$("div").fadeIn()* for fading in all * * elements
- `$("#info").hide()` for hiding an element whose `id` is set as `info`
- `$(".info").show()` for showing an element whose `class` attribute as the value, `info`

From these examples, it is clear that the selector can be an id, class, element, or `this`. Any jQuery code exists within the `document ready` event. This helps in executing the code only after the Document Object Model (DOM) loads completely or is ready. Following is the syntax of the event:

**Syntax**

```
$(document).ready(function()
{
   // Code jQuery functions here.
});
OR
$(function()//Alternate syntax
{
   // Code jQuery functions here.
});
```

Where,

- `$`: Is the jQuery identifier.
- `document`: Refers to the DOM of the HTML page.
- `ready`: Is an event that is raised when the fully loaded DOM is ready for manipulation through JavaScript. At this point, the script can safely find and access the HTML elements in the DOM. For example, it can find the `<a>` element for associating it with a click event handler.
- `function`: Refers to an anonymous function, as it has no name and it contains an action to be taken.

It is rational to wait for the DOM to load fully before the HTML page handles any requests and responses. This is because it is not sensible to hide an element or resize an image, which is not loaded completely. This enables a Web developer to include JavaScript code inside the `<head>` tag.

## *13.2 Functions in jQuery*

In jQuery, a Web developer can define a function in four different ways. It is also possible to define a custom function.

### *13.2.1 Basic Declaration*

This is the easiest way of declaring a function in jQuery. Code Snippet 2 shows the basic declaration to define a function in jQuery.

**Code Snippet 2**:

```
function Multiply(var1, var2, var3)
{
    return (var1 * var2 * var3);
}
document.write("Basic Way:"+Multiply(10, 10, 8));
```

In Code Snippet 2, the `Multiply` function is defined to return the product of three numbers. This way of declaring a function is useful for testing something quickly. It is not a good choice for supporting code reuse.

### *13.2.2 Declaration as a Variable Name*

A Web developer can define a function through an expression, which is assigned to a variable. After storing the expression in a variable, this variable is then used as a function. Code Snippet 3 uses a variable as a function.

**Code Snippet 3**:

```
var Multiply = function (var1, var2, var3)
{
    return (var1 * var2 * var3);
}
document.write("Function as Variable Name "+Multiply(10, 10, 8));
```

In Code Snippet 3, the code declares the variable `Multiply` and assigns an anonymous function to it. This function returns the product of three numbers. In the `write()` method, the variable itself is used as a function to return the product.

### *13.2.3 Self-invoking Function Declaration*

Apart from assigning an anonymous function to a variable, a Web developer can pass it directly to a method. This argument acts as a self-invoking function, which is typically an expression that the script automatically invokes without being called. It has no name and is impossible to call it explicitly.

To declare such a function expression, a Web developer should add parentheses around the function. Code Snippet 4 defines a self-invoking function.

**Code Snippet 4**:

```
(function () {document.write("Self Invoking Function, Hello! I
called myself");})();
```

In Code Snippet 4, the self-invoking function is within parentheses and calls itself on its own. The string is the output that is displayed in the browser.

### *13.2.4 User-defined Function Declaration*

Web developers can create their own jQuery functions, which are known as user-defined functions. The `jQuery.fn` or `$.fn` object allows creating a user-defined function. It is equal to `jQuery.prototype`. Code Snippet 5 creates a user-defined function.

**Code Snippet 5**:

```
$(document).ready(function(){
    $.fn.myFunction = function(){
        alert('You have successfully defined the function!');
```

```
    }
    $(".call-btn").click(function(){
        $.fn.myFunction();
    });
});
```

In Code Snippet 5, `$.fn` object allows creating a user-defined function, `myFunction`, which shows an alert message. It makes the function available for all objects of jQuery. This is evident when the click property of the button whose `class` attribute is set to `call-btn` which is used to invoke the newly created function.

Figure 13.2 shows a code that defines four functions, each with a different declaration method.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
5  <script>
6  //Basic Way
7  function Multiply(var1, var2, var3)
8  {
9      return (var1 * var2 * var3);
10 }
11 document.write("Basic Way:"+Multiply(10, 10, 8));
12 document.write("<br>");
13 //Function as Variable
14 var Multiply = function (var1, var2, var3)
15 {
16     return (var1 * var2 * var3);
17 }
18 document.write("Function as Variable Name "+Multiply(10, 10, 8));
19 document.write("<br>");
20 //Self Invoking Function
21 (function () {
22     document.write("Self Invoking Function, Hello! I called myself");
23 })();
24 //Creating own functions
25 $(document).ready(function(){
26     $.fn.myFunction = function(){
27         alert('You have successfully defined the function!');
28     }
29     $(".call-btn").click(function(){
30         $.fn.myFunction();
31     });
32 });
33 </script>
34 </head>
35 <body>
36 <div id="div1"><h2>Funtions in jQuery</h2></div>
37  <button type="button" class="call-btn">Click Me</button>
38 </body>
39 </html>
```

**Figure 13.2: Code for Declaring Functions in Different Ways**

In Figure 13.2, the code specifies the Google CDN reference. In the `<script>` tag, the code is executed to invoke the `Multiply` function, which is written in the basic as well as in the variable forms. The jQuery library invokes and executes the self-invoking function without writing any code for doing the same. Finally, when a user clicks the `Click Me` button, its `click` property invokes the user-defined function, `myFunction` and displays an alert box.

Figure 13.3 shows the output.

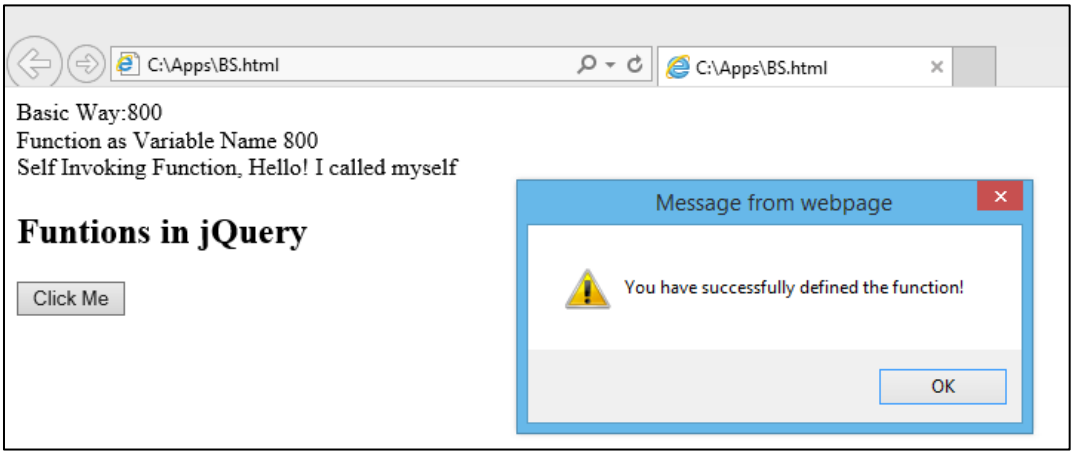



**Figure 13.3: Output of Code for Declaring Functions in Different Ways**

## *13.3 Selectors in jQuery*

The jQuery library allows using CSS and its own custom selectors for swiftly accessing and manipulating the HTML elements in the DOM. A jQuery Selector refers to a function in the form of an expression for searching the matching element in the DOM. In other words, selectors allow selecting single or multiple HTML elements for triggering different actions on the selected ones. All selectors begin with the jQuery identifier (`$`), which is followed by parentheses. The library allows accessing the elements by:

- ID
- Class name
- Tag name
- Attribute
- Attribute values
- Other selectors

### *13.3.1 ID Selector*

ID selector refers to the `id` attribute of a tag for searching the corresponding element. As the ID of each element is unique, the selector is useful only for finding a single tag at a time. It works by relying on the `document.getElementById()` function.

To find a tag or an element through its ID, a Web developer needs to specify ID selector, by beginning it with the hash (`#`) character and following it by the `id` value of the element.

**Syntax**

```
$('#<id>')
```

where,

`id`: Is the value of `id` attribute of the required HTML element.

Figure 13.4 shows how to use an ID selector in jQuery.

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <title></title>
6      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
7      <script type="text/javascript">
8          $(document).ready(function () {
9              $('#button1').click(function () {
10                 alert('jQuery Selector');
11             });
12         });
13     </script>
14 </head>
15 <body>
16 <input id="button1" type="button" value="Click Me" />
17 </body>
18 </html>
```

**Figure 13.4: Code for Using an ID Selector**

In Figure 13.4, the `click` event is associated with the `Click Me` button whose `id` is `button1`. When a user clicks this button, the event is raised to execute an anonymous function that displays the specified alert message.

Figure 13.5 shows the output.

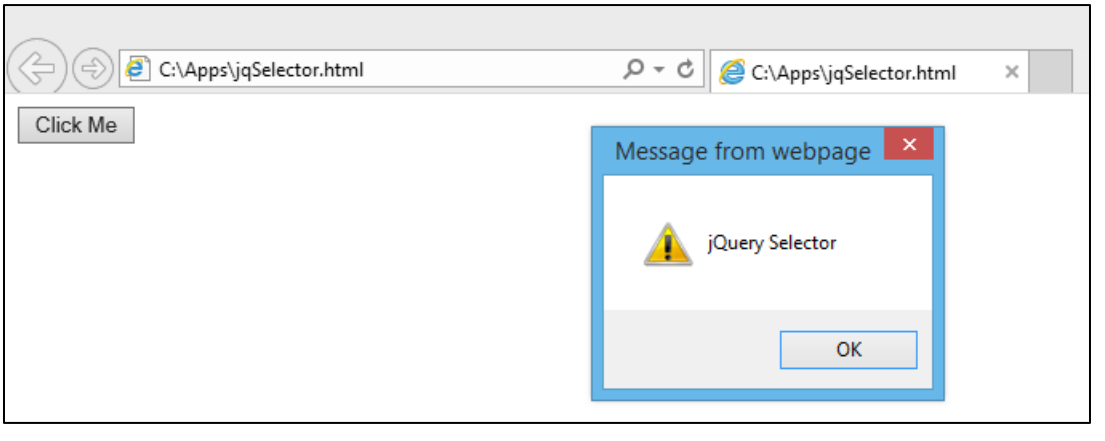



**Figure 13.5: Output of Code Using an ID Selector**

Following are some points to remember about ID selector:

- It is the most efficient of all selectors.
- It accesses only the first element, in case of two or more tags with the same ID.

- It is not the same as `document.getElementById()`, as the latter returns a DOM object and the former returns a jQuery object that includes the DOM object and provides different methods. Therefore, a Web developer can invoke jQuery methods such as `click()` on the returned object. To refer to the underlying DOM object using a jQuery object, a Web developer should use, `$('#btnidentifier')[0]`.
- It is slower in execution than `document.getElementById()`. Still, it is in use for harnessing the power of the extra functionality or methods of a jQuery object.
- It does not throw an error unlike `document.getElementById()`, which raises an error if the element is not found. The `length` property of ID selector allows checking whether an element was found and returned or not.

### *13.3.2 The Class Selector*

The class selector looks for HTML elements that have the specified class. It works by using the `getElementsByClassName()` native function. However, the user's browser should support this function. To find a tag or an element through its class, a Web developer needs to specify the class selector by beginning it with the dot (.) character and following it by the `class` name of the element.

**Syntax**

```
$('.<class>')
```

where,

`class`: Is the value of `class` attribute of the required HTML element.

Code Snippet 6 shows some examples of class selectors.

**Code Snippet 6**:

```
$('.small') // Selects all elements with class small
$('.small,.big') // Selects all elements with class small or big
```

Figure 13.6 shows how to use a class selector in jQuery.

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <title></title>
6      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
7      <script type="text/javascript">
8          $(document).ready(function () {
9              $('.small').css('border', '5px solid green');
10         });
11     </script>
12 </head>
13 <body>
14 <span class="small">
15         Span 1
16     </span>
17     <br /><br />
18     <div class="small">
19         Div 1
20     </div>
21     <br />
22     <span class="big">
23         Span 2
24     </span>
25     <div class="smaller">
26         Div 2
27     </div>
28 </html>
29
```

**Figure 13.6: Code Using a Class Selector in jQuery**

In Figure 13.6, jQuery fetches all the elements whose class name is `small`. Then, it changes their style to apply a green border, which is 5-pixel thick by invoking the jQuery object's `css()` method.

Figure 13.7 shows the output.

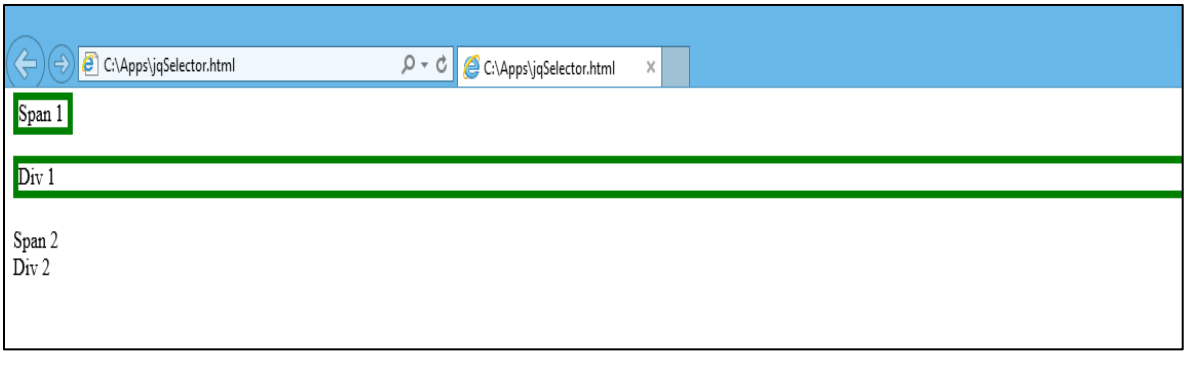



**Figure 13.7: Output of Code Using a Class Selector**

### *13.3.3 The Element Selector*

The element selector access elements as per their name. To find an element by its name, a Web developer needs to specify the element selector within the parentheses following the `$` character.

**Syntax**

```
$(<element>)
```

where,

`element`: Is the name of the element.

Code Snippet 7 shows some examples of element selectors.

**Code Snippet 7**:

```
$('td') // Selects all <td> elements
$('div a') // Selects all anchor elements that are inside the div element
```

Figure 13.8 shows the code for using an element selector.

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <title></title>
6      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
7      <script type="text/javascript">
8          $(document).ready(function () {
9              alert($('td').length);
10         });
11     </script>
12 </head>
13 <body>
14 <table style="border:1px solid black; border-collapse:collapse">
15         <tr>
16             <td>India</td>
17             <td>Germany</td>
18             <td>France</td>
19         </tr>
20         <tr>
21             <td>Europe</td>
22             <td>Asia</td>
23             <td>Australia</td>
24         </tr>
25     </table>
26     </body>
27 </html>
```

**Figure 13.8: Code for Using an Element Selector**

In Figure 13.8, the jQuery code displays the total number of `<td>` elements by triggering an alert box. To find the total number of `<td>` tags, it uses the `length` property of the element selector, `$('td')`.

Figure 13.9 shows the output, which returns the count as 6.

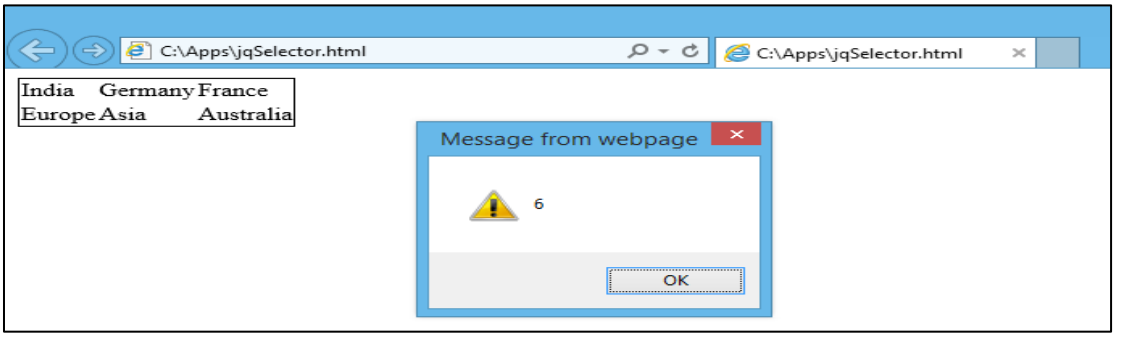



**Figure 13.9: Output of Code for Using an Element Selector**

### *13.3.4 The Checked Selector*

The checked selector accesses and selects all selected radio buttons and check boxes. To access these input controls, a Web developer needs to include: `checked` inside double quotes, which should be within the parentheses following the `$` character.

**Syntax**

```
$(":checked")
```

Code Snippet 8 shows how to access all radio buttons at once that a user has selected.

**Code Snippet 8**:

```
$('input[type="radio"]:checked')
```

Figure 13.10 shows how to use the checked selector.

```
1   <!DOCTYPE html>
2   <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3   <head>
4       <meta charset="utf-8" />
5       <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
6       <script type="text/javascript">
7           $(document).ready(function () {
8               $('#btnSubmit').click(function () {
9                   var result = $('input[type="checkbox"]:checked');
10                  if (result.length > 0) {
11                      var resultString = result.length + " checkboxe(s) checked<br/>";
12                      result.each(function () {
13                          resultString += $(this).val() + "<br/>";
14                      });
15                      $('#divResult').html(resultString);
16                  }
17                  else {
18                      $('#divResult').html("No checkbox checked");
19                  }
20              });
21          });
22      </script>
23  </head>
24  <body>
25  Countries :
26      <input type="checkbox" name="skills" value="India" />India
27      <input type="checkbox" name="skills" value="Germany" />Germany
28      <input type="checkbox" name="skills" value="Croatia"/>Croatia
29      <input type="checkbox" name="skills" value="Russia" />Russia
30      <br /><br />
31      <input id="btnSubmit" type="submit" value="submit" />
32      <br /><br />
33      <div id="divResult">
34      </div>
35      </body>
36  </html>
```

**Figure 13.10: Code for Using the Checked Selector**

In Figure 13.10, the jQuery code uses the checked selector to find out the number of selected check-boxes and stores it in `the result` variable. This variable then is recognized as the returned jQuery object. The object's `length` property helps in checking whether the user has selected any check boxes or not. If yes, the code under the `if` statement is executed, which stores the total count of selected check boxes in the `resultString` variable.

Next, the `each()` method of `result` object is used to refer to each selected check box for fetching the value of the corresponding `value` attribute.

After identifying the total count along with the check box names, ID selector is used to display the result by invoking its `html()` method. ID selector is referring to the `<div>` tag with the id, `divResult`. This whole process of finding the total count and check box names is triggered as soon as a user clicks the `submit` button.

Figure 13.11 shows the output.

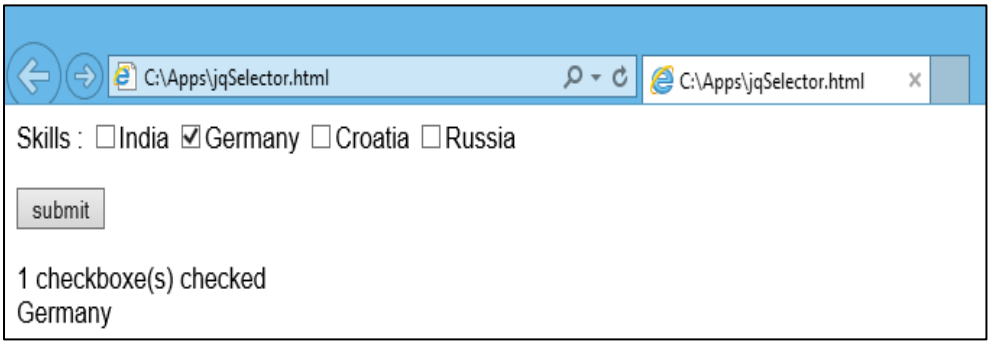



**Figure 13.11: Output of Code Using the Checked Selector**

## *13.4 JSON and Conversions to JSON*

JSON refers to a lightweight format for exchanging and storing data. It is an easier alternative to eExtensible Markup Language (XML) in terms of usage. This is because it is a self-describing format due to which it is simple to comprehend the data exchanged between a server and a client (browser).

Exchanging data between these two nodes happens only in text format. JSON is a simple text format to use in this scenario, although it uses JS syntax. It also works independent of any programming language.

A Web developer can convert an object into JSON and then send it to the server. It is also possible to convert data obtained from the server into JSON objects and arrays. Both the processes are free from complex parsing.

### *13.4.1 Creating a JSON Object*

The format of JSON is almost similar to JS objects. So, it is written in the form of key/value or name/value pairs. However, unlike in JS, the key or name in JSON is always a string due to which it must be within double quotes. Further, in JS, strings can appear in single or double quotes.

In JSON, a name in double quotes is separated by the value with a colon.

**Syntax**

```
{"name":value}
```

In a name/value pair, the value needs to a valid data type, which can be number, string, boolean, array, null, or object. JSON do not support date data type. So, the only way to represent a date in JSON is to make it a string.

Code Snippet 9 shows some examples of name/value pairs in JSON.

**Code Snippet 9**:

```
{"city":"Moscow"}

{"age":40}

{"marriage":"1980-12-23"}
```

In Code Snippet 9, `city` is the key or name in double quotes and `Moscow` is its string value. Similarly, `age` is the key, while `40` is its numeric value. For the `marriage` key, the value is a string showing the wedding date. In jQuery, Web developers usually work with JSON objects. Code Snippet 10 shows how to store employee data in a JSON object, `employeeJSON`.

**Code Snippet 10**:

```
var employeeJSON =
{
    "firstName": "Philips",
    "lastName": "LG",
    "gender": "Male",
    "salary": 5000,
    "bike": null
};
```

In Code Snippet 10, the employee data is included inside the curly braces. Further, comma separates each name/value pair. There is no comma at the end of the last name/value pair.

Each name/value pair is referred to as a property. A Web developer can specify any number of properties in a JSON object. Code Snippet 11 represents the same data using XML.

**Code Snippet 11**:

```
<Employee>
  <firstName>Philips</firstName>
  <lastName>LG</lastName>
  <gender>Male</gender>
  <salary>5000</salary>
  <bike>null</bike>
</Employee>
```

From Code Snippet 11, it is clear that the JSON format is simpler than XML.

### *13.4.2 Accessing Data from a JSON Object*

The dot (.) notation and the corresponding property name enable accessing data from a JSON object.

**Syntax**

```
<Object>.<propertyName>;
```

Code Snippet 12 shows how to fetch the first name of the employee from the `employeeJSON` object created in Code Snippet 10.

**Code Snippet 12**:

```
var firstName = employeeJSON.firstName;
```

Figure 13.12 shows the code for accessing all data from the `employeeJSON` object using jQuery.

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
6     <script type="text/javascript">
7         $(document).ready(function () {
8
9             // Creating a JSON object
10            var employeeJSON = {
11                "firstName": "Philips",
12                "lastName": "LG",
13                "gender": "Male",
14                "salary": 5000
15            };
16
17            // Accessing data from a JSON object
18            var result = '';
19            result += 'First Name = ' + employeeJSON.firstName + '<br/>';
20            result += 'Last Name = ' + employeeJSON.lastName + '<br/>';
21            result += 'Gender = ' + employeeJSON.gender + '<br/>';
22            result += 'Salary = ' + employeeJSON.salary + '<br/>';
23
24            $('#resultDiv').html(result);
25        });
26    </script>
27 </head>
28 <body style="font-family:Arial">
29     <div id="resultDiv"></div>
30 </body>
31 </html>
```

**Figure 13.12: Code for Fetching Data from a JSON Object**

In Figure 13.12, the code inside the anonymous function creates the required JSON object as soon as the document is loaded. It then declares a variable named `result` for accessing the data from the object. The variable is set to a concatenated string, which uses the individual property name of the JSON object for fetching all its data. The result is a JSON string that is displayed through the `<div>` tag.

Figure 13.13 shows the output.

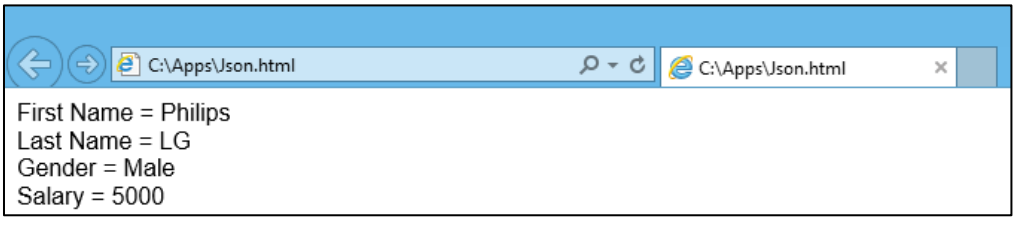



**Figure 13.13: Output of Code for Fetching Data from a JSON Object**

### *13.4.3 Using JSON Arrays*

Let's imagine a scenario wherein a Web developer needs to store the data of more than one employee in a JSON object. This is where a JSON array is useful. A JSON array can hold several objects. To create a JSON array, include the objects in square brackets and separate each by a comma. Code Snippet 13 shows how to create a JSON array indicating the colors in a rainbow.

**Code Snippet 13**:

```
[ "Red", "Blue", "Green", "Yellow", "Violet", "Indigo", "Orange" ]
```

A JSON array is almost similar to a JS array. However, its values or elements can be number, string, boolean, array, null, or object. It cannot be a valid expression, unlike in JS. Code Snippet 14 shows how to create a JSON object holding the data of two employees.

**Code Snippet 14**:

```
var employeesJSON = [
{
    "firstName": "Philips",
    "lastName": "LG",
    "gender": "Male",
    "salary": 5000
},
{
    "firstName": "Samsung",
    "lastName": "Sam",
    "gender": "Female",
    "salary": 4000
}];
```

In Code Snippet 14, both elements in the array are employee objects. To access these from the JSON array, `employeeJSON`; a Web developer should use the object's index position. Code Snippet 15 shows how to retrieve the last name from both the objects of `employeesJSON` array.

**Code Snippet 15**:

```
var result = employeesJSON[0].lastName;
var result = employeesJSON[1].lastName;
```

### *13.4.4 Converting an Object or Array into a JSON String*

When sending data to a server, it should be in the string format. So, for converting a JS object into a string, a Web developer should use `JSON.stringify()`. This method is used on both arrays and objects.

Figure 13.14 shows how to convert the `employeesJSON` array into a JSON string using `JSON.stringify()`.

```
1   <!DOCTYPE html>
2   <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3   <head>
4       <meta charset="utf-8" />
5       <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
6     <script type="text/javascript">
7         $(document).ready(function () {
8             var employeesJSON = [
9                 {
10                    "firstName": "Philips",
11                    "lastName": "LG",
12                    "gender": "Male",
13                    "salary": 5000
14                },
15                {
16                    "firstName": "Samsung",
17                    "lastName": "Sam",
18                    "gender": "Female",
19                    "salary": 4000
20                }
21            ];
22
23            var JSONString = JSON.stringify(employeesJSON);
24            $('#resultDiv').html(JSONString);
25        });
26      </script>
27  </head>
28  <body style="font-family:Arial">
29      <div id="resultDiv"></div>
30  </body>
31  </html>
```

**Figure 13.14: Code for Converting an Array into a JSON String**

In Figure 13.14, `JSON.stringify()` takes the array as an input and converts it into a string in JSON notation. This string is stored in `JSONString` variable and is displayed on the Webpage through the `<div>` tag. This string is now ready to be sent to the server.

Figure 3.15 shows the output.

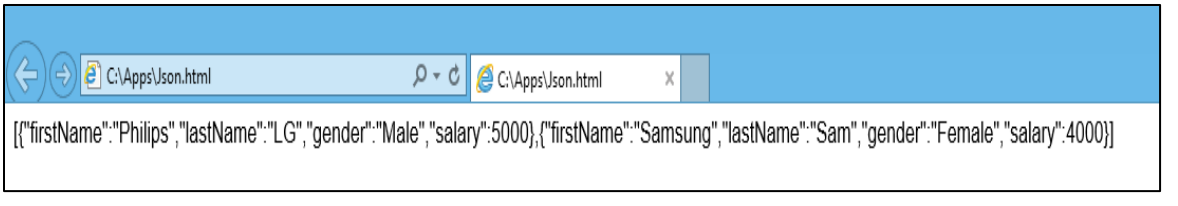



**Figure 13.15: Output of Code Converting an Array into a JSON String**

### *13.4.5 Converting a String into a JSON Object or Array*

A Web developer can parse a string using `JSON.parse()` to convert it into a JSON object or an array. To avoid a syntax error, the string should be in JSON format.

Figure 13.16 shows how to convert a string into a JSON array using `JSON.parse()`.

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
6     <script type="text/javascript">
7         $(document).ready(function () {
8             var JSONString = '[{ "firstName": "Philips", "lastName": "LG", "gender": "Male", "salary": 5000 },
9             { "firstName": "Samsung", "lastName": "Sam", "gender": "Female", "salary": 4000 }]';
10
11            var employeesJSON = JSON.parse(JSONString);
12
13            var result = '';
14
15            $.each(employeesJSON, function (i, item) {
16                result += 'First Name = ' + item.firstName + '<br/>';
17                result += 'Last Name = ' + item.lastName + '<br/>';
18                result += 'Gender = ' + item.gender + '<br/>';
19                result += 'Salary = ' + item.salary + '<br/><br/>';
20            });
21
22            $('#resultDiv').html(result);
23        });
24     </script>
25 </head>
26 <body style="font-family:Arial">
27     <div id="resultDiv"></div>
28 </body>
29 </html>
```

**Figure 13.16: Code for Converting a String into a JSON Array**

In Figure 13.16, a string variable, `JSONString` is set to a string in JSON syntax. The `JSON.parse()` function takes this string as a parameter and converts it into a JSON array, `employeesJSON`. Then, the `each()` method of jQuery allows looping through each employee element or object in the array to retrieve its corresponding property values.

Figure 13.17 shows the output.

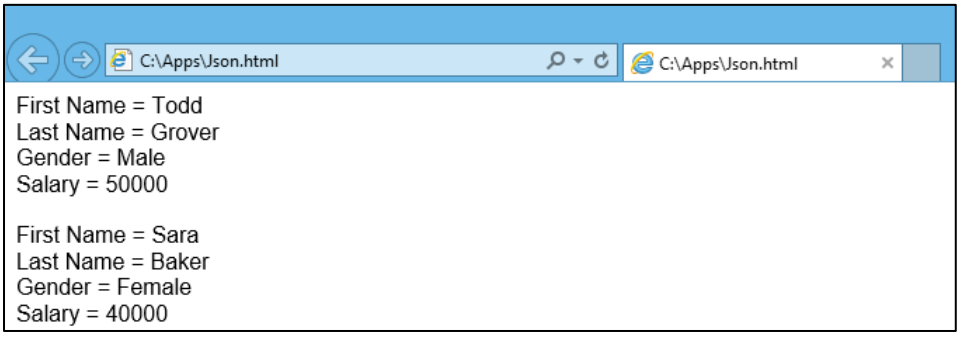



**Figure 13.17: Output of Code Converting a String into a JSON Array**

## *13.5 Event Delegation in jQuery*

jQuery is customized for reacting to events occuring on a Webpage. An event denotes the exact moment when an action from a user occurs. A few examples of events include loading a Webpage, clicking an element, selecting a check box, and hovering the mouse over an element.

With events, Web developers can design dynamic Webpages. When an event is raised, an event listener, which is associated with source of event, listens to it and notifies the handler. An event handler, which is associated with the event, is usually a custom function that conveys what to do for handling the event.

In jQuery, event delegation enables a Web developer to assign a single listener to a parent element, which will listen for all descendants corresponding to a selector.

This is irrespective of whether these descendants are present now or are included later. The descendants of an element encompasses all its children elements including the great grand-child ones.

The `on()` and `delegate()` methods facilitate delegating an event in jQuery. Delegating an event eliminates the need to add event listeners to particular tags. This is because a Web developer can now add a listener to a parent element, which can analyze the bubbled events to look for a match on child elements.

For example, let's assume that there is a parent `<ul>` element with 50 child elements. Something should happen when a user clicks any child element. For this, a Web developer may add a distinct event listener to each `<li>` element. However, a problem arises when the developer frequently tends to add and remove these child elements. In this case, adding and removing the listeners would be a tedious job and may lead to performance issues. So, it would be much better to add a listener to the parent element.

However, an important question here is that how would the developer find out which child element was clicked? The solution here is to use the `target` property of the event object to obtain a reference to the clicked node while the event bubbles up to the parent element.

When the click event triggers on any child element of the `<UL>`, it also triggers on all ancestors of that element. This is called event bubbling in which an event bubbles upwards from the source element in the DOM tree. As a result, it activates each of its parent click handlers, which includes `<ul>`, `<div>`, `<p>`, `<body>`, `<html>`, and root. In other words, clicking an `<li>` tag indicates clicking the whole body of the document.

Event bubbling and `target` property are the main features of event delegation in jQuery. They boost performance when there are hundreds of child elements on which a specific event should be listened and handled.

The `target` property of any event refers to the element that originated the event. Through event delegation, a Web developer can add a handler to an element, wait for event bubbling, and quickly identify the event's source, which is the element where the event occurred. Following are the benefits of event delegation:

- Less handlers to take up memory
- Better performance
- Reduced risk of crashing
- No re-assigning the handlers if the child elements keep changing

Figure 13.18 shows how to delegate and undelegate a click event for an unordered list using the `on()` method of jQuery.

```
1   <!DOCTYPE html>
2   <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3   <head>
4       <meta charset="utf-8" />
5       <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.3.min.js"></script>
6       <script type="text/javascript">
7           $(document).ready(function () {
8               $('ul').on('click', 'li', function () {
9                   $(this).fadeOut(500);
10              });
11
12              $('#btnAdd').on('click', function () {
13                  $('ul').append('<li>New List Item</li>');
14              });
15
16              $('#btnUndelegate').on('click', function () {
17                  $('ul').off('click', 'li');
18              });
19          });
20      </script>
21  </head>
22  <body style="font-family:Arial">
23      <input id="btnAdd" type="button" value="Add a New List Item" />
24      <input id="btnUndelegate" type="button" value="Undelegate" />
25      <ul>
26          <li>List Item1</li>
27          <li>List Item2</li>
28      </ul>
29  </body>
30  </html>
```

**Figure 13.18: Code for Delegating and Undelegating an Event**

In Figure 13.18:

- Clicking any `<li>` tag triggers event bubbling in which the click event bubbles up to its parent element, `<ul>`. This is because there is no dedicated event handler attached to the clicked list item. The parent element handles the bubbled event, as it has the click event handler. The `on()` method of the jQuery object directly binds the `click` event to the `<ul>` tag and the anonymous function acts as an event handler. It takes three parameters namely, the event, selector parameter, and the anonymous function.

- The selector parameter informs the handler to listen to the click event. It also tells the handler to check whether the element triggering the click event matches with the selector parameter at the time of hearing. In this case, the source of the event is `<li>`, which matches with the second parameter. Thus, the anonymous function executes to fade out the clicked `<li>` element in `3000` milliseconds.
- When a user clicks the button whose `id` is `btnAdd` for dynamically adding a new list item, the item is added to `<ul>` using the `append()` method. There is no need to specify the button's click event handler explicitly in the `<input>` tag. As the new item is added to the same parent element `<ul>`, the click event of the list item also gets bubbled to the same parent.
- Upon clicking the `Undelegate` button, the `undelegate()` method of the `<ul>` element selector is invoked to stop event delegation.

Figure 13.19 shows the output.

**Figure 13.19: Output of Code Delegating and Undelegating an Event**

**Quick Test 13.1**

1. The checked selector allows finding which radio buttons are selected.
   a. True
   b. False

2. It is impossible to call a self-invoking function explicitly.
   a. True
   b. False

## *13.6 Check Your Progress*

1. Which of the following statements about jQuery is true?

| | | | |
|---|---|---|---|
| (A) | It has a small but vibrant community | (C) | It supports AJAX |
| (B) | It uses obtrusive JS | (D) | None of these |

2. _____ is a valid jQuery selector.

| | | | |
|---|---|---|---|
| (A) | `$(this)` | (C) | `$(*)` |
| (B) | `$(p)` | (D) | `$(#btn)` |

3. The _____ object allows declaring a user-defined function.

| | | | |
|---|---|---|---|
| (A) | `.fn` | (C) | `Jquery.prototype` |
| (B) | `$.fn` | (D) | `$.fn.prototype` |

4. Which of the following methods converts an object into a JSON string?

| | | | |
|---|---|---|---|
| (A) | `JSON.string()` | (C) | `JSON.stringify()` |
| (B) | `JSON.convertostring()` | (D) | `JSON.tostring()` |

5. Which of the following statements about event delegation is true?

| | | | |
|---|---|---|---|
| (A) | It features event bubbling | (C) | It boosts performance |
| (B) | It assigns an event handler to a parent element | (D) | All of these |

## *13.6.1 Answers*

| | |
|---|---|
| 1 | It supports AJAX |
| 2 | `$(this)` |
| 3 | `$.fn` |
| 4 | `JSON.stringify()` |
| 5 | All of these |

*Answers to Quick Test*

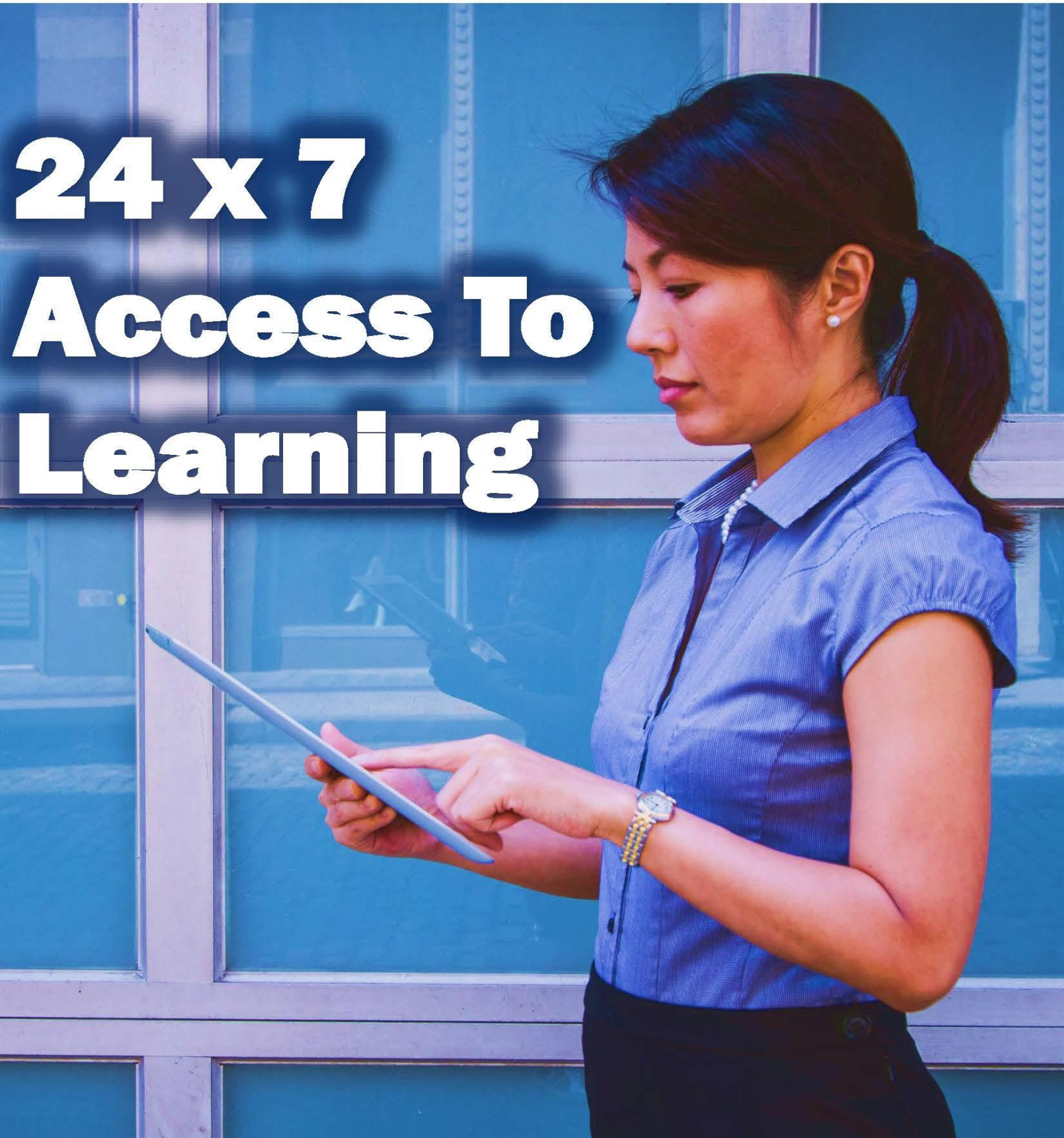
*Quick Test 13.1*

| | |
|---|---|
| 1 | True |
| 2 | True |

# Summary

- jQuery is a lightweight, open-source JS library that makes JS programming easier and more efficient.
- jQuery uses unobtrusive JS, offers a variety of reusable functions, is extensible, and is compatible with modern browsers.
- Web developers can either download the jQuery file or include it by referring to a CDN for using jQuery in their Web applications.
- The customized syntax of jQuery is `$(selector).action()`.
- There are four ways of declaring a function in jQuery, which are basic, as a variable, self-invoking, and user-defined.
- A jQuery Selector refers to a function in the form of an expression for searching the matching element in the DOM.
- The jQuery library allows selecting the HTML elements by ID, class name, tag name, attribute, and attributed values.
- JSON is a lightweight format for exchanging and storing data as well as an easier alternative to XML.
- Unlike in JS, the key or name in JSON is always a string.
- Each name/value pair in JSON format is referred to as a property.
- A JSON object is a collection of properties, each separated by a colon.
- The `JSON.stringify()` method converts a JSON object into a JSON string.
- The `JSON.parse()` method converts a JSON string into a JSON object or array.
- Event delegation allows assining a single listener to a parent element, which will listen for all descendants corresponding to a selector.
- The `on()` and `delegate()` methods facilitate delegating an event in jQuery.
- Event bubbling in which an event bubbles upwards from the source element in the DOM tree.
- Event bubbling and `target` property are the main features of event delegation.

Onlinevarsity



# Session - 14

## Functions, Widgets, and Plugins in jQuery

Session Objectives:

This session introduces the built-in methods in jQuery and elaborates briefly on the live() method for event delegation. It also describes the different Asynchronous JavaScript and XML (AJAX) methods that jQuery supports and the callback concept. The session also briefly introduces the jQuery UI JavaScript (JS) library, its different widgets including autocomplete, and the datatable dynamic control.

At the end of this session, you will be able to:

- → Describe various types of functions in jQuery
- → Explain use of AJAX in jQuery
- → Elucidate the callback concept in jQuery
- → Describe implementation and use of various controls such as widgets and datatables in jQuery

## *14.1 Built-in Methods in jQuery*

The jQuery framework features different capabilities of JavaScript (JS). As JS has a handy set of built-in functions, all of them are also available through this framework. This is one of the examples proving that functions are reusable blocks. The JS built-in functions are methods with which Web developers can quickly manipulate data values such as dates, numbers, and strings.

Table 14.1 shows some of the built-in jQuery methods.

| Method | Description |
|---|---|
| `charAt()` | Fetches the character at the mentioned index. |
| `concat()` | Returns a new string by combining two strings. |
| `indexOf()` | Fetches the index inside the requesting string object of the initial occurrence of the mentioned value. If nothing is found, it returns `-1`. |
| `length()` | Returns the number of characters in a string. |
| `forEach()` | Invokes a function for each array element. |
| `pop()` | Removes and returns the last array element. |
| `push()` | Appends single or multiple elements at the end of an array and fetches its modified length. |
| `substr()` | Extracts the string characters starting at the mentioned index location until the specified index length. |
| `reverse()` | Changes the order of array elements by making the first one last and the last one first. |
| `sort()` | Arranges the array elements in an ascending or descending order. |
| `toLowerCase()` | Converts a string value into lower case. |
| `toUpperCase()` | Converts a string value into upper case. |
| `tostring()` | Converts a given value into a string. |

**Table 14.1: Built-in Methods in jQuery**

Figure 14.1 shows the code using some of the built-in methods.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script>
function myFunction() {
    var str = "HELLO WORLD";
    var res = str.charAt(0)
    document.write("charAT Function:" +" " + res);
    document.write("<br>");
    var str1 = " FOLKS"
    document.write("concat Function:" + " " + str.concat(str1));
    document.write("<br>");
    document.write("Index Of Function:" +" "+ str.indexOf("WORLD"));
    document.write("<br>");
    document.write("Length of string:" +" "+ str.length);
    document.write("<br>");
    document.write("Converting to Lower case:" + " "+ str.toLowerCase());
}
</script>
</head>
<body>

<p>Click here to execute builtin functions in jQuery.</p>

<button onclick="myFunction()">Click me</button>

</body>
</html>
```

**Figure 14.1: Code Using Built-in Methods**

In figure 14.1, the code uses different built-in functions of jQuery for manipulating two strings. The `charAt()` built-in function returns the character at `0`th position in the string, which is precisely the first character. The `concat()` function adds two strings namely, `str` and `str1`. The `indexOf()` function retrieves the index position from where the string `WORLD` starts in the string, `str`. The `length()` function retrieves the total number of letters in the `str` string. Finally, the `toLowerCase()` function converts the upper case letters in the string to lower case.

Figure 14.2 shows the output.

**Figure 14.2: Output of Code Using Built-in Methods**

### *14.1.1 Live Function in JQuery*

The built-in `live()` method delegates single or multiple event handlers for all elements that match with the selector and defines a function to run once the event occur.

Event handlers attached in this manner work for both existing and upcoming elements that match the selector. The `live()` method is just another way to perform event delegation in jQuery.

The `on()` and `delegate()` methods facilitate bubbling up the event to the source's parent element. On the other hand, the `live()` method bubbles up to the document object. A Web developer can remove the associated event handler using the `die()` method.

**Syntax**

```
$(selector).live(event,data,function);
```

where,

- `event`: Refers to single or multiple events to delegate to the elements, with multiple event values parted by space.
- `data`: Is optional and represents extra data to be passed to the function.
- `function`: Indicates the function to execute once the event takes place.

**Note:** jQuery recommends using `on()` and `off()` methods instead of `live()` and `die()` methods to the users of latest version of jQuery.

In the latest version of jQuery, the `on()` method takes care of all event delegation tasks. It can handle all tasks of the `live()` method.

Figure 14.3 shows the code for using `live()`, `on()`, and `die()` methods.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js">
5   </script>
6   <script type="text/javascript">
7           $(document).ready(function () {
8               $('li').live('click', function () {
9                   alert("live event triggered")
10
11              });
12
13              $('#btnAdd').on('click', function () {
14                  $('ul').append('<li>New List Item</li>');
15              });
16
17              $('#btnUndelegate').on('click', function () {
18                  $('li').die('click');
19              });
20          });
21      </script>
22  </head>
23  <body style="font-family:Arial">
24      <input id="btnAdd" type="button" value="Add a New List Item" />
25      <input id="btnUndelegate" type="button" value="Undelegate" />
26      <ul>
27          <li>List Item</li>
28          <li>List Item</li>
29      </ul>
30  </body>
31
32  </html>
```

**Figure 14.3: Code Using `live()`, `on()`, and `die()` Built-in Methods**

In figure 14.3, the code refers to an older version of jQuery framework to demonstrate the use of `live()` method. The `live()` and `on()` methods attach the event handler to all `<li>` elements and `btnAdd` button, respectively. Both of them take the event and an anonymous function as parameters. When a user clicks any list item, the `live()` method executes the function and displays the alert box. When a user clicks the button, the `on()` method runs the function to add a new list item to the existing unordered list. Clicking the `btnUndelegate` button invokes the `on()` method, which then calls the `die()` method to stop event delegation.

Figure 14.4 shows the output.

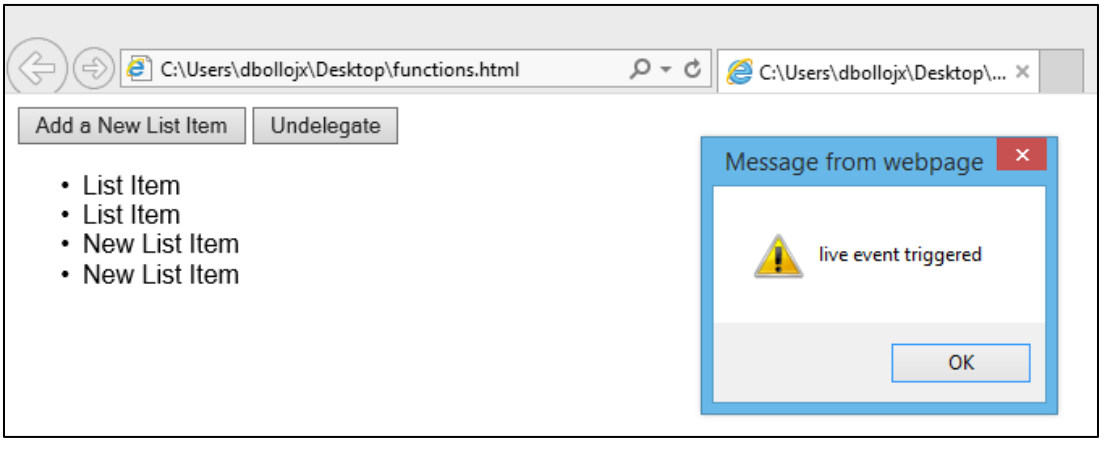



**Figure 14.4: Output of Code Using `live()`, `on()`, and `die()` Built-in Methods**

## *14.2 jQuery AJAX*

AJAX allows updating different parts of a Webpage by exchanging data with the Web server. However, it does so without reloading or refreshing the whole Webpage. Some examples of AJAX Websites are Gmail and YouTube.

Coding regular AJAX without jQuery can be a bit complicated, as its syntax for implementation tends to vary from one browser to another. This means a Web developer need to write additional code for testing their Webpages on each of the major browsers.

However, with jQuery, this extra effort is no longer required, as the makers have already coded for it. Web developers can now code implement an AJAX functionality in just one or two lines. So, in jQuery, it is easier to write AJAX code.

jQuery provides many AJAX methods for developing Web applications. Through the jQuery AJAX methods, Web developers can request data from a distant Web server via GET and POST. They can request data in any of the four different formats namely, text, JavaScript Object Notation (JSON), eXtensible Markup Language (XML), and HTML. Further, they can easily load the remote data quickly into desired HTML elements.

### *14.2.1 AJAX jQuery Load() Method*

The jQuery `load()` method is an easy and robust AJAX functionality. As the name suggests, it loads the requested data from a Web server and inserts it into the desired HTML element.

**Syntax**

```
$(selector).load(URL,data,callback);
```

where,

- `URL`: Indicates the Uniform Resource Locator (URL) of a resource on a server, a Web developer intends to load.
- `data`: Is optional and indicates a set of key/value pairs as a query string in the form of JSON object or string that goes along with the request. The default request method used here is GET.
- `callback`: Is optional and indicates the name of a callback function that will run after the `load()` method completes executing the request.

Figure 14.5 shows how to fetch data from an HTML file on a server and load it in the body of a `<div>` element.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
5   <script>
6   $(document).ready(function(){
7       $("button").click(function(){
8           $("#div1").load("test.txt");
9       });
10  });
11  </script>
12  </head>
13  <body>
14
15  <div id="div1"><h2>Ajax load demo</h2></div>
16
17  <button>Get test file Content</button>
18
19  </body>
20  </html>
```

**Figure 14.5: Code Using the `load()` Built-in Method**

In figure 14.5, once the document is ready for manipulation and when a user clicks the button, the click event is raised and the `load()` method triggers. The method triggers an AJAX request to the URL.

The `load()` method takes the URL as the parameter and fetches its contents. Here, the text file is in the same folder as this HTML file. The method finally loads the fetched content in the body of `<div>` by assigning the content to it through the id selector.

Figure 14.6 shows the output.

**Figure 14.6: Output of Code Using the `load()` Built-in Method**

## *14.3 Callbacks in jQuery*

A callback function runs once the current jQuery effect such as hide, fade, or slide, is over. Usually, in JS, execution occurs line by line. Nevertheless, when there is an effect to display, the next code line can execute even if the former is not over. This can lead to errors. To keep this risk away, a Web developer defines a callback function.

**Syntax for Hide Effect**

```
$(selector).hide(speed,callback);
```

where,

- `speed`: Is optional and indicates the speed of hiding an element in milliseconds or as `fast` or `slow`.
- `callback`: Is optional and indicates the name of a callback function to run after `hide()` executes completely.

Figure 14.7 shows the code for hiding a paragraph without using a callback function.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
5   <script>
6   $(document).ready(function(){
7       $("button").click(function(){
8           $("p").hide(1000);
9           alert("The sentence is now hidden");
10      });
11  });
12  </script>
13  </head>
14  <body>
15
16  <button>Hide</button>
17
18  <p>This is a sentence with little content.</p>
19
20  </body>
21  </html>
```

**Figure 14.7: Code Hiding an Element without a Callback Function**

In figure 14.7, the code invokes the anonymous function once a user clicks the button. The function executes to hide the sentence in the <p> element in 1000 milliseconds and show the alert dialog box.

Figure 14.8 shows the output.

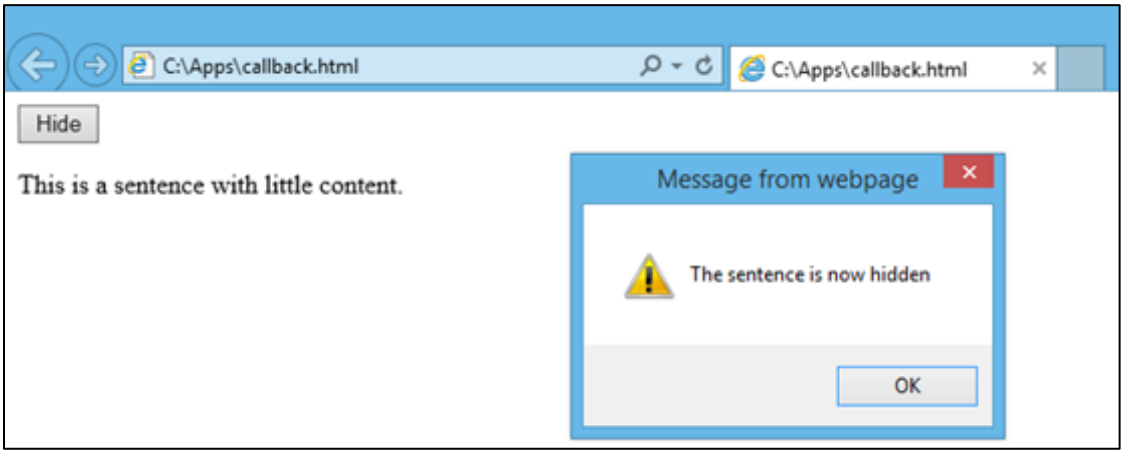



**Figure 14.8: Output of Code Hiding an Element without a Callback Function**

In figure 14.8, it is clear that the alert box is displayed without waiting for the hide effect to get over. Now let's see what happens when a callback fucntion is present.

Figure 14.9 shows the code for hiding a paragraph using a callback function.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
5  <script>
6  $(document).ready(function(){
7      $("button").click(function(){
8          $("p").hide("slow", function(){
9              alert("The sentence is now hidden");
10         });
11     });
12 });
13 </script>
14 </head>
15 <body>
16
17 <button>Hide</button>
18
19 <p>This is a sentence with little content.</p>
20
21 </body>
22 </html>
```

**Figure 14.9: Code Hiding an Element with a Callback Function**

In figure 14.9, the code invokes the anonymous function once a user clicks the button. The function executes to hide the sentence in the <p> element slowly and then, invoke a callback function to display the alert dialog box.

Figure 14.10 shows the output.

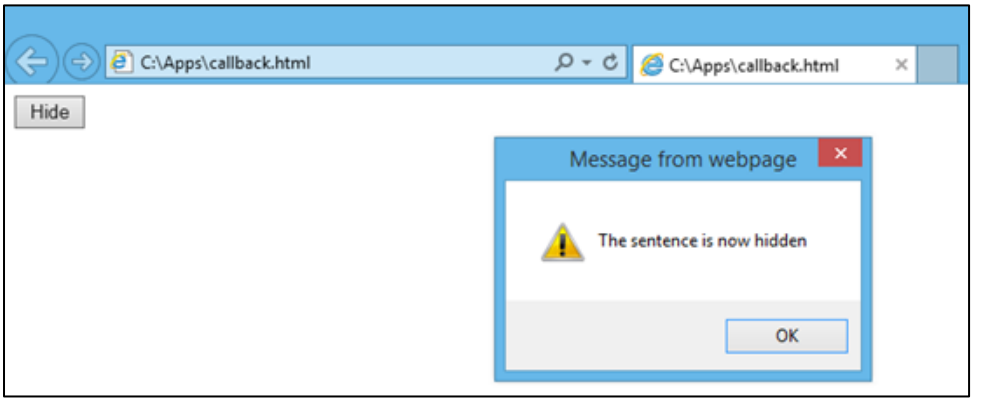



**Figure 14.10: Output of Code Hiding an Element with a Callback Function**

In figure 14.10, it is clear that the alert box is displayed only after the sentence in the paragraph hides. completely. This means that the callback function is called only after the effect is completed.

In the `load()` method, the optional callback parameter indicates a callback function. This executes only when the desired content loads into intended section of a Webpage. This callback function can take three parameters.

**Syntax**

```
$(selector).load(URL,data,function(response,status,XHR));
```

where,

- `response`: Holds the fetched content in case loading is successful.
- `status`: Has the status of the loading action.
- `XHR`: Possesses the `XMLHttpRequest` object with response text.

**Note:** XHR has all contents of the browser's native `XMLHttpRequest` object.

Code Snippet 1 shows how to pass a callback function as a parameter to the `load()` method.

**Code Snippet 1**:

```
$("btnLoad").click(function(){
    $("#divLoad").load("test.txt", function(response, status, xhr){
        if(status == "success")
            alert("Content loaded!");
        if(status == "error")
            alert("Error in loading: " + xhr.status);
    });
});
```

In code snippet 1, the code displays the alert box once the `load()` method's execution is over. If the method succeeds, it displays the corresponding alert message. Otherwise, it shows an error message using the status property of `XMLHttpRequest` object.

### *14.3.1 jQuery $.get() Method*

The `$.get()` method asks data of a resource from a Web server through a GET request.

**Syntax**

```
$.get(URL,callback);
```

where,

- `URL`: Indicates the URL a Web developer intends to request.
- `callback`: Is optional and is a function to be executed when the request is successful.

Figure 14.11 shows how to use `$.get()` method with a callback function to display the contents of an ASP file into an HTML page.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
5  <script>
6  $(document).ready(function(){
7      $("button").click(function(){
8          $.get("demo_test.asp", function(data, status){
9              alert("Data: " + data + "\nStatus: " + status);
10         });
11     });
12 });
13 </script>
14 </head>
15 <body>
16
17 <button>Get response from concerning asp page</button>
18 </body>
19 </html>
20
```

**Figure 14.11: Code for Using `$.get()` with a Callback Function**

In figure 14.11, the occurrence of click event invokes the `$.get()` method, which fetches the data from the ASP file at the specified URL. An ASP file is a server side script. The method takes two parameters, of which the second one is the callback function. This function contains the requested data and status of the request. Following is the content existing in the ASP file:

```
<%
response.write("Response from asp file")
%>
```

`Response.write()` works in the same way as `document.write()` in HTML. It shows the message on the HTML page.

Figure 14.12 shows the output.

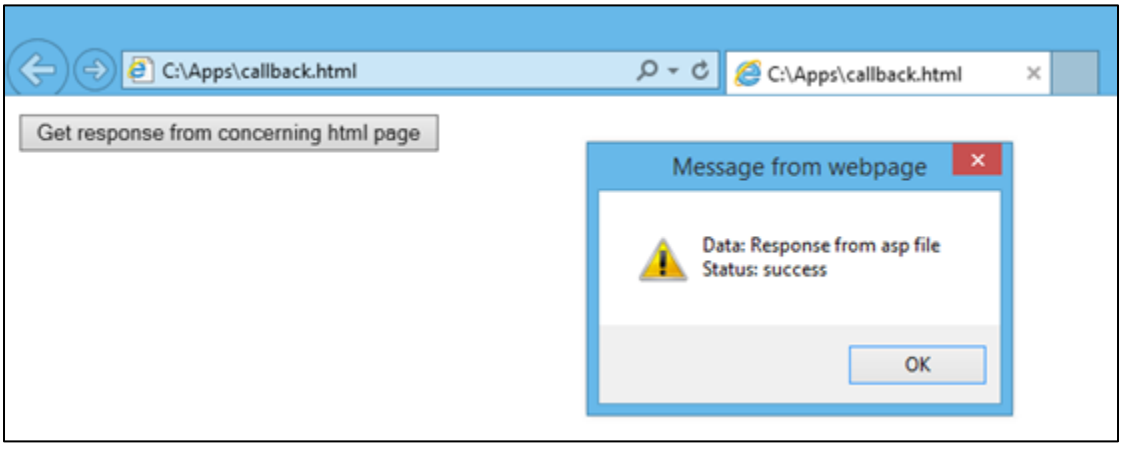



**Figure 14.12: Code for Using `$.get()` with a Callback Function**

Web developers might think that the `$.get()` and `load()` methods are same. However, there is a difference between these two methods of jQuery. Table 4.2 distinguishes them.

| Point of Distinction | `Load()` | `$.get()` |
|---|---|---|
| Functionality | Loads only HTML data after retrieving it from the server. | Loads any type of data, such as JSON and script. |
| Type of HTTP Request | Sends a GET or POST request, but it depends on whether the specified data parameter. If data parameter is specified, POST is used. Otherwise, GET is used. | Sends a GET request always. |
| Selector Usage | Follows a selector for inserting the retrieved content into the element that matches with the selector. For example: `$('selector').load ('url', 'data');` | Does not follow a selector while coding. Instead, it invokes a callback function to insert the retrieved data from the server into an HTML element. For example: `$.get('url', 'data', function (response) { $('selector').html(response); });` |

**Table 4.2: `Load()` versus `Get()`**

### *14.3.2 The jQuery $.post() Method*

The `$.post()` method asks data of a resource from a Web server through a POST request.

**Syntax**

```
$.post(URL,data,callback);
```

where,

- `URL`: Indicates the URL a Web developer intends to request.
- `data`: Is optional and represents extra data to be passed to the function.
- `callback`: Is optional and is a function to be executed when the request is successful.

Figure 14.13 shows how to use `$.post()` method with a callback function to send data and display the contents of an ASP file into an HTML page.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
5  <script>
6  $(document).ready(function(){
7      $("button").click(function(){
8          $.post("demo_test_post.asp",
9          {
10             name: "Bruce Wayne",
11             city: "Gotham"
12         },
13         function(data,status){
14             alert("Data: " + data + "\nStatus: " + status);
15         });
16     });
17 });
18 </script>
19 </head>
20 <body>
21
22 <button>Send an HTTP POST request and get the result back</button>
23
24 </body>
25 </html>
```

**Figure 14.13: Code for Using `$.post()` with a Callback Function**

In figure 14.13, the jQuery `$.post()` method takes three parameters: First being the ASP file URL, second being the data of name and city, and third being the callback function. The code passes this data along with the request. In the callback function, the first parameter possesses the requested page's content, while the second one indicates the request's status.

By sending data as its second parameter, we are writing it to the `demo_test_post.asp` file and fetching the same for displaying on the HTML page. The ASP file has the following script:

```
<%
dim fname,city
fname=Request.Form("name")
city=Request.Form("city")
```

```
Response.Write("Hello " & fname & ", where are you from" & city &
". ")
%>
```

The ASP script here reads the parameters, assigns them to variables, and returns a result. The script declares two variables namely, `fname` and `city`. Then, it assigns the values fetched from the HTML page to them using `request.form()`.

Figure 14.14 shows the output.

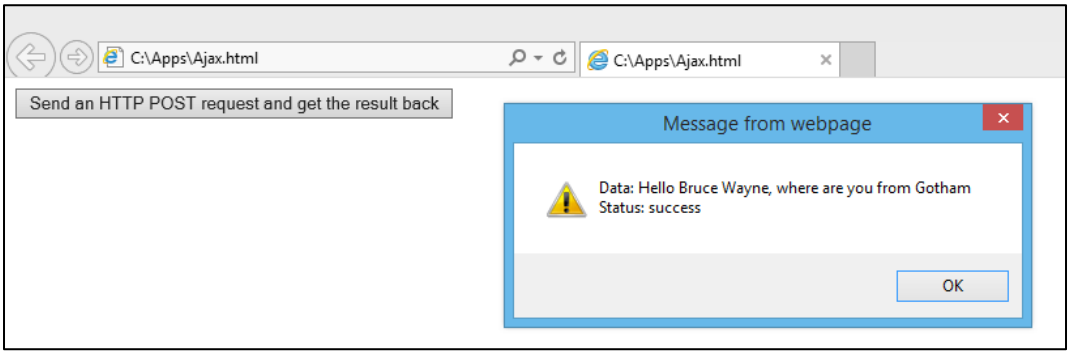



**Figure 14.14: Output of Code Using `$.post()` with a Callback Function**

Web developers may consider that the `$.get()` and `$.post()` methods are same. However, there is a difference between the two. Table 14.3 distinguishes them.

| Point of Distinction | `$.get()` | `$.post()` |
|---|---|---|
| Functionality | Is mainly for obtaining data from a Web server although it can send data. | Is mainly for changing data on a Web server although it can send data. |
| Caching | Is capable of caching the requested data and returning it. | Does not cache data. |
| Method of Sending Data | Appends data to the URL as a query string. | Includes the data in the message body. |
| Limitation on Data to Be Sent | Puts a limit on the amount of data to be sent because of the limit on the length of the data passed to an URL. | Has no such limitation. |

**Table 14.3: `$.get()` versus `$.post()`**

## *14.4 Autocomplete Function in jQuery*

While filling a form, it is common to see a drop-down menu giving a list of words that best matches with the typed characters. In short, the browser tries to predict the exact value a user wishes to enter in the text field. This is commonly evident when a user types just two or more characters of the word or phrase in the search box of Google after which a drop-down.

Auto completion is a comonly used technique in modern Websites to offer a list of suggestions for selecting the exact value instead of typing it. In jQuery, the autocomplete widget in JQuery UI provides this functionality. It is a control that functions similar to a drop-down list. However, it filters the options for showing the ones that match with what the user is entering into a control.

JQuery UI stands for jQuery User Interface and is a robust open-source JS library based on jQuery framework. It offers a collection of jQuery plugins for easily adding new and complex functionalities to the library of jQuery. It comprises the following:

- interactive plugins or interactions such as resize and drag
- jQuery plugins or widgets for inserting visual components such as menus, autocomplete, and progress bars
- Effects encompassing transitions and custom animations, including toggle, show, and hide
- Utilities as modular functions or tools such as position and toggle for internal use by jQuery UI

Just as jQuery, Web developers can either download the jQuery UI files from the its Official Website or use a CDN to refer to them. Code Snippet 2 shows the CDN links to include in the `<head>` tag.

**Code Snippet 2**:

```
<link href = https://code.jquery.com/ui/1.10.4/themes/ui-
lightness/jquery-ui.css rel = "stylesheet">
<script src = "https://code.jquery.com/jquery-1.10.2.js"></script>
<script src = "https://code.jquery.com/ui/1.10.4/jquery-
ui.js"></script>
```

In code snippet 2, the first link includes the theme of jQuery UI, `ui-lightness`, through CSS to give a stylish look to the user interface. The second link includes the jQuery library. It is essential to add this link, as the new library is built based on the jQuery library. Finally, the third link includes the jQuery UI library to enable it for the Webpage in which these three links exist.

To use the autocomplete plugin, the library offers the `autocomplete()` method. Using this method, a Web developer can specify a list of predicted values as a drop-down for any input field.

**Syntax**

```
$ (selector, context).autocomplete (opt);
```

This syntax states that an `<input>` element should be handled as an input field and displayed just atop the drop-down list. The `opt` parameter refers to an object that conveys the behavior of the list. A Web developer can include more than one option, which they can separate using a comma.

**Syntax**

```
$(selector, context).autocomplete({opt1: val1, opt2: val2..... });
```

Table 14.4 lists the different options.

| Option | Description |
|---|---|
| `appendTo` | Appends an element to the list. The value is `null` by default. |
| `source` | Indicates how the data will be obtained, which will then match with the input data. It is essential to specify a value for using the autocomplete widget. The default value is `none`. |
| `delay` | Refers to the waiting time in milliseconds before fetching the matching values, as per the value of the `source` option. The default value is `300`. This option alleviates thrashing, especially while fetching non-local data by giving time for typing more characters prior to auto search. |
| `miniLength` | Refers to the number of characters to be typed before fetching the matching values. This option averts generating a too big list. The default value is `1`. |
| `Autofocus` | Focuses the first menu item when its value is `true`. The default value is `false`. |
| `Disabled` | Inactivates the widget if the value is `true`. The default value is `false`. |
| `Position` | Specifies the position of the list with regards to input element. The default value is `{ my: "left top", at: "left bottom", collision: "none" }`. |

**Table 14.4: Autocomplete Options**

Figure 14.15 shows how to use the `autocomplete()` method in jQuery.

```html
<!doctype html>
<html lang = "en">
   <head>
      <meta charset = "utf-8">
      <title>jQuery UI Autocomplete function</title>
      <link href = "https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
       rel = "stylesheet">
      <script src = "https://code.jquery.com/jquery-1.10.2.js"></script>
      <script src = "https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
      <script>
       $(function() {
          var countries  =  [
             "Germany",
             "Ghana",
             "Craotia",
             "Cambodia",
          ];
          $( "#automplete-1" ).autocomplete({
             source: countries
          });
       });
     </script>
   </head>
   <body>
      <!-- HTML -->
      <div class = "ui-widget">
         <p>Type "g" or "c"</p>
         <label for = "automplete-1">Tags: </label>
         <input id = "automplete-1">
      </div>
   </body>
</html>
```

**Figure 14.15: Code for Using the `autocomplete()` Method in jQuery**

In figure 14.15, the CDN links for jQuery UI and jQuery exists under the `<head>` tag. Next, the code defines an array named `countries` holding the names of four countries. Then, the `autocomplete()` method is associated with the input field whose id is `automplete-1`. It takes the `source` option whose value is set to `countries`, which is an array. This populates the autocomplete drop-down list with the elements of the array. This list appears when the user enters '`g`' or '`c`' in the input field. The type of source mentioned here is local, as it is defined in the code. A local source is preferable for having a fewer values in the autocomplete drop-down list.

In the HTML body, the `div` tag contains the `.ui-widget` class. This class is applicable to the autocomplete widget's outer container for setting the font family and its size.

Figure 14.16 shows the output.

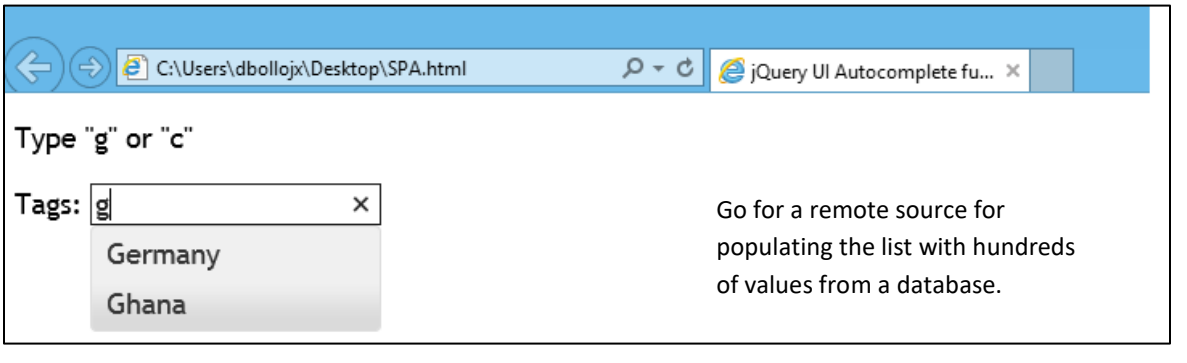


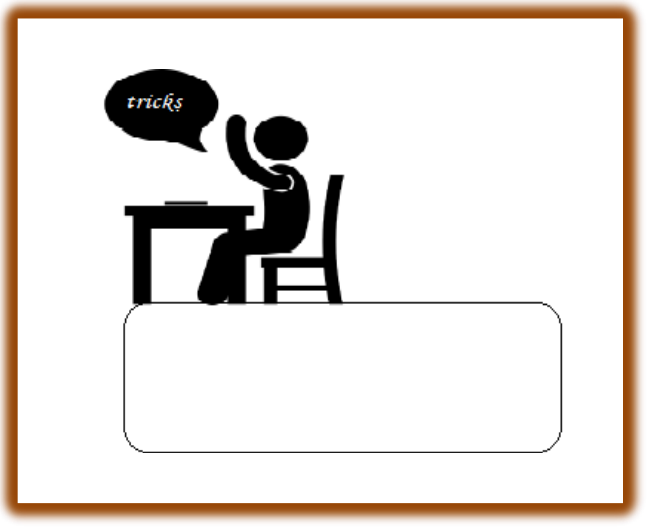

**Figure 14.16: Output of Code Using the `autocomplete()` Method in jQuery**

## *14.5 Widgets in jQuery*

A few visual controls or elements on a Webpage are likely to accept the intended state, roles, and responsibilities naturally at the time of development. Adding more such elements to a Webpage usually increases the level of complexity. This triggers a need for a maintainable design solution. Such a solution features modularity, purposeful design, and unit tests that strengthen the major parts of the environment, coding language, and the platform.

In case of jQuery, the platform is the browser, JS is the language, and jQuery and jQuery UI libraries denote the environment's major parts. The libraries tend to make the code simple by:

- Reducing the number of lines.
- Resolving browser compatibility issues.
- Ensuring consistency in AJAX interactions.
- Implementing modularity.




At the core of jQuery UI's visual section is a widget. A widget refers to an object associated with an element on a Webpage for handling many aspects with other JS objects or widgets. These aspects are:

- Lifespan
- Interaction
- State
- Inheritance

Widgets in jQuery come from jQuery UI and are handy for designing Webpages that are highly interactive. The jQuery UI widgets are special plugins that simplify the task of applying a functionality to their associated elements. The standard jQuery plugins do not possess a few built-in capabilities, such as regulating the plugin's lifespan, revealing methods, and associating data with its elements. However, widgets come with these capabilities. Following are the jQuery UI widgets:

- Accordions
- Buttons
- Tabs
- Sliders
- Autocomplete
- Dialogs
- Datepickers
- Progress bars
- Menus
- Spinners
- Tooltips

### *14.5.1 Using the Tooltip Widget*

Replacing the native tooltip, the Tooltip widget comes with fresh themes and facilitates customization. It allows display other content instead of just the `title` attribute, such as AJAX-retrieved data and footnotes. The widget also customizes the alignment of tooltips, such as aligning it in the middle. It also applies additional styling to personlize the appearance. This is useful for indicating a warning or error.

Web developers can apply a tooltip to any HTML element. In HTML without jQuery, a Web developer simply needs to specify the `title` attribute to the desired element for showing a tooltip. The browser displays the value of this attribute as a tooltip when a user hovers over the element. Code Snippet 3 illustrates the use of both the *`title`* attribute and Tooltip widget of jQuery UI.

**Code Snippet 3**:

```
HTML
<label id="lName" for="tName" title="Full Name">Name</label>
<input id="tName" type="text" title="Your full name as it appears
in memo"/>
jQuery
$('#tName').tooltip();
```

In code snippet 3, the *label* element shows the native tooltip, while the text field uses the widget's *tooltip()* method to display the tooltip. Code Snippet 4 shows how to apply the widget to all the elements in a document, which have the title attribute.

**Code Snippet 4**:

```
$(document).tooltip();
```

jQuery UI offers the *tooltip()* method for applying a tooltip to any HTML element. It applies a fading effect by default to hide or show the tooltip instead of simply toggling the tooltip.

**Syntax**

```
$(selector, context).tooltip(opt);
$(selector, context).tooltip({opt1: val1, opt2: val2..... });
```

Table 14.5 lists the different options that a Web developer can pass as parameters to the `tooltip()` method.

| Option | Description |
|---|---|
| `content` | Indicates the tooltip content. The default value is a function that returns the value of the title attribute. |
| `items` | Denotes the items that can display the tooltip. The default value is `title`. |
| `disabled` | Inactivates the widget if the value is `true`. The default value is `false`. |
| `hide` | Denotes the effect of animation while hiding the widget. The default value is `true`. |
| `show` | Conveys the manner in which the display of the tooltip shall be animated. The default value is `true`. |
| `track` | Allows the tooltip to follow the mouse if the value is `true`. The default value is `false`. |
| `position` | Sets the tooltip's position with regards to its target element. The values can be `at`, `within`, `of`, `using`, `collision`, and `my`. The default value is a function returning the value of title attribute. |
| `tooltipClass` | Is useful for showing an error or warning via a tooltip. The default value is `null`. |

**Table 14.5: Tooltip Options**

Figure 14.17 shows how to use the `tooltip()` method in jQuery.

```
1  <!doctype html>
2  <html lang = "en">
3     <head>
4        <meta charset = "utf-8">
5        <title>jQuery UI Tooltip</title>
6        <link href = "https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
7         rel = "stylesheet">
8        <script src = "https://code.jquery.com/jquery-1.10.2.js"></script>
9        <script src = "https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
10       <script>
11        $(function() {
12           $("#myTooltip").tooltip();
13        });
14      </script>
15    </head>
16
17    <body>
18       <!-- HTML -->
19       <label for = "name">FirstName:</label>
20       <input id = "myTooltip" title = "Enter Your first name">
21    </body>
22 </html>
```

**Figure 14.17: Code for Using the `tooltip()` Method in jQuery**

In figure 14.17, the `tooltip()` method is invoked to display the value of `title` attribute near the text field. Figure 14.18 shows the output.

**Figure 14.18: Output of Code Using the `tooltip()` Method in jQuery**

### *14.5.2 Using the Dialog Widget*

The Dialog widget provides a floating window that has a title and a body with content. The window is moveable via dragging and resizable through stretching. A user can even close the dialog by clicking the **x** icon present in its top-right corner. The window is dynamic enough to display a scrollbar if the content exceeds its maximum height. For customizing it, a Web developer can add a button to its bottom and modal in its body area.

jQuery UI offers the *`dialog()`* method for showing a dialog box.

**Syntax**

```
$(selector, context).dialog(opt);
$(selector, context).dialog({opt1: val1, opt2: val2..... });
```

Table 14.6 lists a few options that a Web developer can pass as parameters to the `dialog()` method.

| Option | Description |
|---|---|
| `title` | Indicates the title text. The default value is null. |
| `autoOpen` | Opens the dialog box if the value is `true`, which is the default value. If false, it opens the dialog box only when a Web developer invokes dialog('open'). |
| `closeOnEscape` | Closes the dialog box once a user presses the Esc key when the focus in on the dialog box if the value is `true`. This is the default value. |
| `buttons` | Includes buttons in the dialog box. |
| `draggable` | Allows dragging the dialog box by dragging its title bar if the value is `true`. This is the default value. |
| `resizable` | Allows resizing the dialog box in any direction if the value is `true`. This is the default value. |
| `modal` | Applies the behavior of a modal to the dialog box if the value is `true`. Other items on the page are disabled. The default value is `false`. |
| `position` | Sets the dialog box's initial position, which can be top, center, left, right, or bottom. |

**Table 14.6: Dialog Box Options**

Figure 14.19 shows how to use the `dialog()` method in jQuery.

```
1  <!doctype html>
2  <html lang = "en">
3     <head>
4        <meta charset = "utf-8">
5        <link href = "https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css"
6         rel = "stylesheet">
7        <script src = "https://code.jquery.com/jquery-1.10.2.js"></script>
8        <script src = "https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
9        <script>
10        $(function() {
11           $( "#myDialog" ).dialog({
12              autoOpen: false,
13           });
14           $( "#open" ).click(function() {
15              $( "#myDialog" ).dialog( "open" );
16           });
17        });
18     </script>
19  </head>
20  <body>
21     <div id = "myDialog"
22      title = "Dialog">jQuery Dialog!</div>
23     <button id = "open">Open Dialog</button>
24  </body>
25 </html>
```

**Figure 14.19: Code for Using the `dialog()` Method in jQuery**

In figure 14.19, the `<div>` and `<button>` elements have their respective dialogs. For `<div>`, the `dialog()` method has the `autoOpen` option set to `false`. This means that the dialog box will not open when the page is loading. When a user clicks the button, the `dialog()` method with the `open` parameter invokes the dialog box for the `<div>` element. This means that clicking the button will create and display the dialog box with the content, `jQuery Dialog!`.

Figure 14.20 shows the output upon clicking the button.

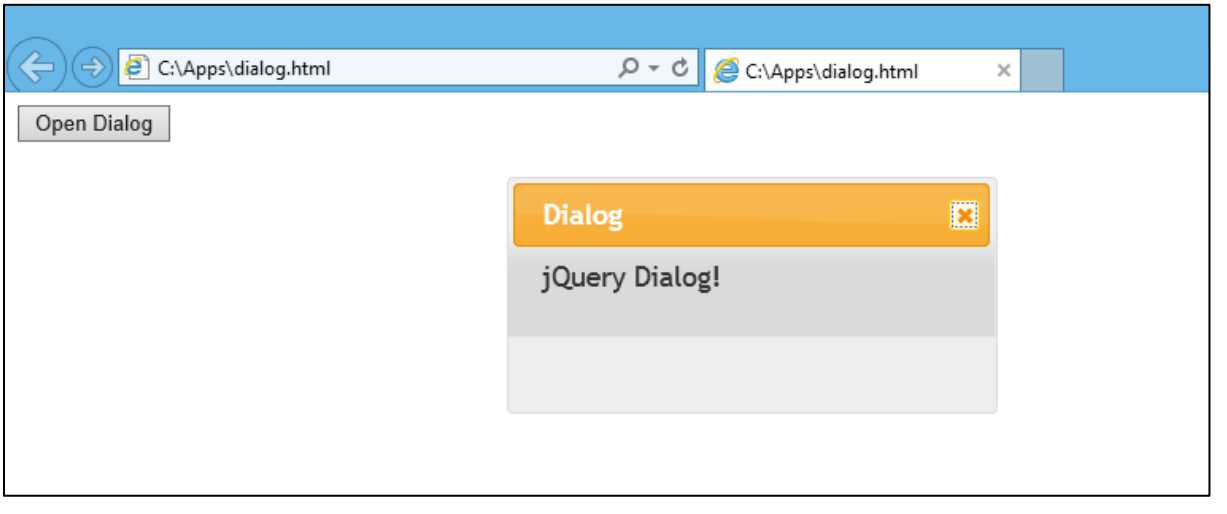



**Figure 14.20: Output of Code Using the `dialog()` Method in jQuery**

## *14.6 Datatable Plugin in jQuery*

Datatable is an intuitive jQuery plugin that adds more functionality to the `<table>` element. It makes a plain HTML table dynamic by easily including the pagination, sorting, and searching utilities in it. This enables a developer to manage the display of huge quantity of data. Similarly, the plugin also enables a user to access the desired data quickly. The plugin is quite flexible to facilitate user-friendly interaction for an HTML table.

Following are the salient feature of datatables:

- Free and open-source
- Support for any data source including AJAX
- Built-in utilities such as instant search, pagination, and column-based ordering
- Built-in extensions such as fixed columns and buttons
- Built-in options such as `scrollY` and `order` along with callback functions
- Completely internationalizable
- Comprehensive documentation
- Hundreds of examples

For using the plugin, a Web developer needs to include only two additional files in the intended Webpage. They are the DataTables JS file and the DataTables CSS file. Web developers can either download the files from the Official Website or refer to a CDN. Code Snippet 5 shows the CDN links to these files, which a Web developer needs to add along with jQuery CDN link.

**Code Snippet 5**:

```
<link rel="stylesheet" type="text/css" href="//cdn.datatables.net/1
.10.13/css/jquery.dataTables.min.css">
<script type="text/javascript" charset="utf8" src="//cdn.datatables
.net/1.10.13/js/jquery.dataTables.min.js"></script>
```

Web developers can use datatables with a variety of data sources. A datatable can accept on the data of an array, an HTML table, or AJAX-retrieved data.

**Syntax**

```
$(selector).dataTable();
```

In this syntax, the jQuery selector is usually a reference to an HTML table, which a Web developer intends to enhance. The `datatable()` method can accept options as optional parameters, which form a configuration object of the dataTable constructor. Code Snippet 6 shows the usage of some of these options.

**Code Snippet 6**:

```
$('table').dataTable( {
  paginate: true,
  scrollY: 400
} );
```

In code snippet 6, pagination and scrolling both are enabled for the datatable.

Figure 14.21 shows how to use the `datatable()` method in jQuery.

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml">
3  <head runat="server">
4      <title></title>
5      <script
6    src="https://code.jquery.com/jquery-2.2.4.min.js"
7    integrity="sha256-Bbhdlvqf/xTY9gja0Dq3HiwQF8LaCRTXxZKRutelT44="
8    crossorigin="anonymous"></script>
9     <script
10   src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"
11   integrity="sha256-VazP97ZCwtekAsvgPBSUwPFKdrwD3unUfSGVYrahUqU="
12   crossorigin="anonymous"></script>
13     <link rel="stylesheet" type="text/css"
14       href="https://cdn.datatables.net/1.10.13/css/jquery.dataTables.min.css" />
15     <script src="https://cdn.datatables.net/1.10.13/js/jquery.dataTables.min.js">
16    </script>
17     <script type="text/javascript">
18       $(document).ready(function () {
19           $('#datatable').dataTable();
20       });
21    </script>
22 </head>
23 <body style="font-family: Arial">
24     <form id="form1" runat="server">
25         <div style="width: 500px; border: 1px solid black; padding: 5px">
26             <table id="datatable">
27                 <thead>
28                     <tr>
29                         <th>ID
30                         </th>
31                         <th>City
32                         </th>
33                          <th>Country
34                          </th>
35                          <th>Continent
36                          </th>
37                     </tr>
38                 </thead>
39                 <tbody>
40                     <tr>
41                         <td>1</td>
42                         <td>Monteria</td>
43                         <td>Colombia</td>
44                         <td>South America</td>
45                     </tr>
46                     <tr>
47                         <td>2</td>
48                         <td>Birmingham</td>
49                         <td>United Kingdom</td>
50                         <td>Europe</td>
51                     </tr>
52                     <tr>
53                         <td>3</td>
54                         <td>Bangalore</td>
55                         <td>India</td>
56                         <td>Asia</td>
57                     </tr>
58                 </tbody>
59             </table>
60         </div>
61     </form>
62 </body>
63 </html>
```

**Figure 14.21: Code for Using the `datatable()` Method in jQuery**

In figure 14.21, the `<head>` tag includes CDN references to jQuery, jQuery UI, and dataTables files. Within the `<script>` tag, the `datatable()` method is associated with the HTML table whose id is `datatable`. This table has three rows.

Figure 14.22 shows the output.

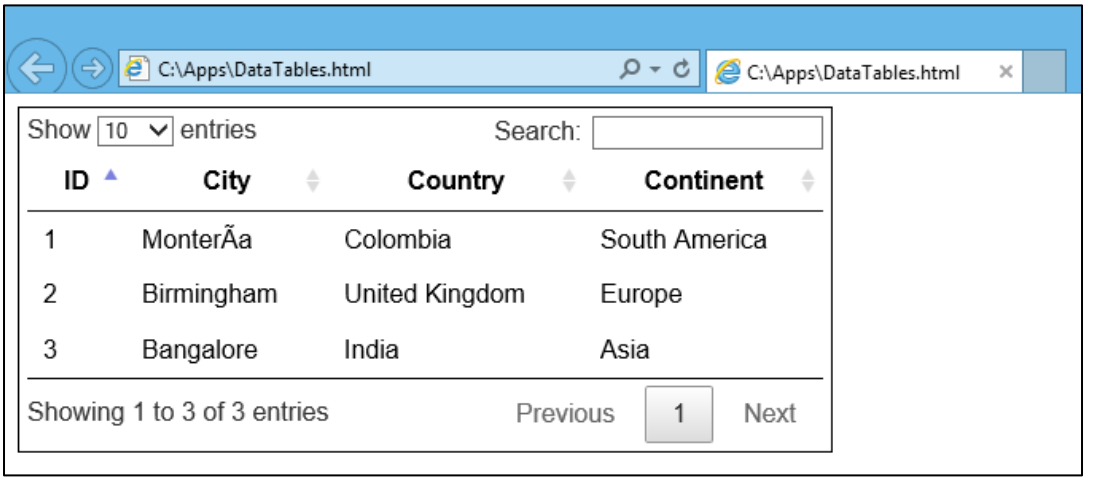



**Figure 14.22: Output of Code Using the `datatable()` Method in jQuery**

In figure 14.22, the HTML table features the search, pagination, and scrolling utilities. It is no longer a plain HTML table.

**Quick Test 14.1**

1. The autocomplete functionality is added directly in jQuery.
   True
   False

2. A callback function waits for an effect to occur completely.
   True
   False

## *14.7 Check Your Progress*

1. _____ is a built-in jQuery function.

| | | | |
|---|---|---|---|
| (A) | `sortstr()` | (C) | `pushstr()` |
| (B) | `pop()` | (D) | `tostr()` |

2. Which of the following parameters are of the live() method?

| | | | |
|---|---|---|---|
| (A) | `Event listener` | (C) | `Callback function` |
| (B) | `data` | (D) | `element` |

3. The _____ object is a parameter of the load() function.

| | | | |
|---|---|---|---|
| (A) | `$.fn` | (C) | `XMLHttpRequest` |
| (B) | `HttpRequest` | (D) | `html()` |

4. Which of the following statements is true about $.get() method?

| | | | |
|---|---|---|---|
| (A) | It can load any type of data. | (C) | It can cache the requested data. |
| (B) | It always uses GET. | (D) | All of these. |

5. Which of the following statements about jQuery UI is true?

| | | | |
|---|---|---|---|
| (A) | It is built upon AJAX. | (C) | It provides widgets.<br>None of these. |
| (B) | It offers the datatable functionality. | (D) | Inline |

## *14.7.1 Answers*

| | |
|---|---|
| 1 | `pop()`. |
| 2 | `data`. |
| 3 | `XMLHttpRequest`. |
| 4 | All of these. |
| 5 | It provides widgets. |

*Answers to Quick Test*

*Quick Test 14.1*

| | |
|---|---|
| 1 | False |
| 2 | True |

# Summary

- Built-in functions are methods that allow manipulating data values such as dates, numbers, and strings.
- The live() method delegates event handlers to existing and future elements that match with the selector.
- The jQuery AJAX methods allow requesting data from a distant Web server via LOAD, GET, and POST in different formats and load the same into the desired HTML element.
- A callback function runs once the current jQuery effect is over.
- The $.get() method asks data of a resource from a Web server through a GET request.
- The $.post() method asks data of a resource from a Web server through a POST request.
- The jQuery UI library offers the autocomplete() method to implment the autocomplete plugin.
- A widget refers to an object associated with an element on a Webpage for handling the lifespan, interaction, state, and inheritance and other aspects with other JS objects or widgets.
- The jQuery UI widgets are special plugins that simplify the task of applying functionality to their associated elements and offering some capabilities lacking in standard jQuery plugins.
- The Datatable jQuery plugin makes a plain HTML table dynamic by easily including the pagination, sorting, and searching utilities in it.

# Onlinevarsity

# Session - 15

# Operators and Statements

Welcome to the Session, **Operators and Statements**.

This session explains about the different type of operators supported by JavaScript. It also explains how to use pattern matching functions on text using regular expressions. Finally, it explains the use of decision-making statements in JavaScript.

In this Session, you will learn to:

- ➔ Explain operators and their types in JavaScript
- ➔ Explain regular expressions in JavaScript
- ➔ Explain decision-making statements in JavaScript

## *15.1 Introduction*

An operator specifies the type of operation to be performed on the values of variables and expressions. JavaScript provides different types of operators to perform simple to complex calculations and evaluations.

Certain operators are also used to construct relational and logical statements. These statements allow implementing decision and looping constructs.

## *15.1.2 Basics of Operators*

An operation is an action performed on one or more values stored in variables. The specified action either changes the value of the variable or generates a new value. This means that an operation requires minimum one symbol and some value. Here, the symbol is called an operator and it specifies the type of action to be performed on the value. The value or variable on which the operation is performed is called an operand. For example, `x*2` is an expression, where `x` and `2` are operands and `*` is an operator. This operator specifies that the multiplication operation is to be performed on the operands.

There are three main types of operators, which are as follows:

- **Unary operators** - Operates on a single operand. For example, the expression `y = -x`.
- **Binary operators** - Operates on two operands. For example, the expression `sum = y + x`.
- **Ternary operators** - Operates on three operands. For example, the expression `age >= 18 ? "Eligible" : "Not Eligible"`.

## *15.2 Operators and their Types*

Operators help in simplifying expressions. JavaScript provides a predefined set of operators that allow performing different operations. JavaScript operators are classified into six categories based on the type of action they perform on operands.

These six categories of operators are as follows:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Bitwise operators

➔ Special operators

## *15.2.1 Arithmetic Operators*

Arithmetic operators are binary operators, as they perform basic arithmetic operations on two operands. The operator appears in between the two operands, which allow you to perform computations on numeric and string values. These computations include addition, subtraction, multiplication, and division.

Table 15.1 lists the arithmetic operators with their descriptions and an example of each type.

| Arithmetic Operator | Description | Example |
|---|---|---|
| + (Addition) | Performs addition. In case of string values, it behaves as a string concatenation operator and appends a string at the end of the other | 45 + 56 |
| – (Subtraction) | Performs subtraction. If a larger value is subtracted from a smaller value, it returns a negative numeric value | 76-78 |
| / (Division) | Divides the first operand by the second operand and returns the quotient | 24 / 8 |
| % (Modulo) | Divides the first operand by the second operand and returns the remainder | 90 % 20 |
| * (Multiplication) | Multiplies the two operands | 98 * 10 |

**Table 15.1: Arithmetic Operators**

Code Snippet 1 calculates the loan interest and the total loan amount to be repaid by using the arithmetic operators.

**Code Snippet 1**:

```
<script>
  var loanAmount = 34500;
  var interest = 8;
  var interestAmount, totalAmount;
  interestAmount = loanAmount * (interest / 100);
  totalAmount = loanAmount + interestAmount;
  document.write("<B>Total amount to be paid ($):</B>" +
  totalAmount + "<BR />");
</script>
```

The code uses the arithmetic operators `*` and `/` to calculate the interest amount. This interest amount is then added to the total amount by using the `+` operator. The `write()` method displays the total amount to be repaid on the loan.

## *15.2.2 Increment and Decrement Operators*

The increment and decrement operators are unary operators, as they operate only on a single operand. The increment operator (`++`) increases the value by 1, while the decrement operator (`--`) decreases the value by 1. These operators can be placed either before or after the operand. If the operator is placed before the operand, the expression is called pre-increment or pre-decrement. If the operator is placed after the operand, the expression is called post-increment or post-decrement.

Table 15.2 demonstrates the use of increment and decrement operators by assuming that the **`numOne`** variable's value is 2.

| Expression | Type | Result |
|---|---|---|
| `numTwo = ++numOne;` | Pre-increment | `numTwo = 3` |
| `numTwo = numOne++;` | Post-increment | `numTwo = 2` |
| `numTwo = --numOne;` | Pre-decrement | `numTwo = 1` |
| `numTwo = numOne--;` | Post-decrement | `numTwo = 2` |

**Table 15.2: Increment and Decrement Operators**

Code Snippet 2 demonstrates the use of unary operators in JavaScript.

**Code Snippet 2**:

```
<script>
  var number = 3;
  alert('Number after increment = ' + ++number);
  alert('Number after decrement = ' + number--);
</script>
```

The first `alert()` function will display the incremented value of the **`number`** variable. This is because in the first statement, `++` operator is evaluated first, and then the incremented value is substituted in the variable **`number`**. The second `alert()` function will not display the decremented value of the **`number`** variable. This is because the current value is first assigned to the variable, and then the `--` operator is evaluated.

## *15.2.3 Relational Operator*

Relational operators are binary operators that make a comparison between two operands. After making a comparison, they return a boolean value namely, `true` or `false`.

The expression consisting of a relational operator is called as the relational expression or conditional expression.

Table 15.3 lists the relational operators along with their descriptions and an example of each type.

| Relational Operator | Description | Example |
|---|---|---|
| == (Equal) | Verifies whether the two operands are equal | 90 == 91 |
| ! = (Not Equal) | Verifies whether the two operands are unequal | 99 != 98 |
| === (Strict Equal) | Verifies whether the two operands are equal and whether are of the same type | 3 === 4 |
| ! == (Strict Not Equal) | Verifies whether the two operands are unequal and whether are not of the same type | 3 !== “3” |
| > (Greater Than) | Verifies whether the left operand is greater than the right operand | 97 > 95 |
| < (Less Than) | Verifies whether the left operand is less than the right operand | 94 < 96 |
| >= (Greater Than or Equal) | Verifies whether the left operand is greater than or equal to the right operand | 92 >= 93 |
| <= (Less Than or Equal) | Verifies whether the left operand is less than or equal to the right operand | 99 <= 100 |

Table 15.3: Relational Operators

Code Snippet 3 compares the value of the **`firstNumber`** variable with the value of the **`secondNumber`** variable using relational operators.

**Code Snippet 3**:

```
<script>
  var firstNumber = 3;
  var secondNumber = 4;
      document.write('First number is greater than the second
    number: ' + (firstNumber > secondNumber));
     document.write('<br/>First number is less than the
   second number: ' + (firstNumber < secondNumber));
   document.write('<br/>First number is equal to the second
   number: ' + (firstNumber == secondNumber));
</script>
```

In the code, each condition is evaluated to return a boolean value. The `alert()` function displays the boolean value as `true` or `false`.

## *15.2.4 Logical Operator*

Logical operators are binary operators that perform logical operations on two operands. They belong to the category of relational operators, as they return a boolean value.

Table 15.4 lists the various logical operators and an example of each type, assuming that **x** is 2 and **y** is 2.

| Logical Operator | Description | Example |
|---|---|---|
| `&&` (AND) | Returns true, if either of the operands are evaluated to true. If first operand evaluates to true, it will ignore the second operand | (x == 2) && (y == 5)<br>Returns `false` |
| `!` (NOT) | Returns false, if the expression is true and vice-versa | !(x == 3)<br>Returns `true` |
| `\|\|` (OR) | Returns true, if either of the operands are evaluated to true. If first operand evaluates to true, it will ignore the second operand | (x == 2) \|\| (y == 5)<br>Returns `true` |

**Table 15.4: Logical Operators**

Code Snippet 4 shows the script that uses logical `AND` operator to check whether the value of **name** and **age** variables are John and 23.

**Code Snippet 4**:

```
<script>
  var name = "John";
  var age = 23;
  alert('John\'s age is greater than or equal to 23 years : ' +
  ((name=="John") && (age >= 23)));
</script>
```

The code declares and initializes two variables namely, **name** and **age** to John and 23 respectively. In the `alert()` function, the logical `AND` operator checks whether the value of the **name** variable is John and the value of the **age** variable is 23. Here, both the expressions are true and therefore, the operator will return `true`, which will be displayed in the message box to the user.

## *15.2.5 Assignment Operator*

Assignment operator assign the value of the right side operand to the operand on the left side by using the equal to operator (=).

The assignment operator is divided into two categories in JavaScript that is as follows:

- **Simple assignment operator** - Is the '=' operator which is used to assign a value or result of an expression to a variable. For example, `result = numOne + numTwo;`

- **Compound assignment operator** - Is formed by combining the simple assignment operator with the arithmetic operators. For example, `salary -= eval(salary * tax / 100);`

Table 15.5 demonstrates the use of assignment operator by assuming the value of the variable **`numOne`** as 6.

| Expression | Description | Result |
|---|---|---|
| `numOne += 6;` | numOne = numOne + 6 | `numOne = 12` |
| `numOne -= 6;` | numOne = numOne – 6 | `numOne = 0` |
| `numOne *= 6;` | numOne = numOne * 6 | `numOne = 36` |
| `numOne %= 6;` | numOne = numOne % 6 | `numOne = 0` |
| `numOne /= 6;` | numOne = numOne / 6 | `numOne = 1` |

**Table 15.5: Assignment Operator**

## *15.2.6 Bitwise Operator*

Bitwise operators represent their operands in bits (zeros and ones) and perform operations on them. However, they return standard decimal values.

Table 15.6 lists various bitwise operators along with their descriptions and an example of each type, assuming **`a`** is 9 (`00001001`) and `b` is 14 (`00001110`).

| Bitwise operators | Description | Example |
|---|---|---|
| `&` (Bitwise AND) | Compares two bits and returns 1 if both of them are 1 or else returns 0 | a & b<br>Returns 8 (`00001000`) |
| `~` (Bitwise NOT) | Inverts every bits of the operand, changes its sign and subtracts by 1. It is a unary operator | ~a<br>Returns -10 |
| `\|` (Bitwise OR) | Compares two bits and returns 1 if the corresponding bits of either or both the operands is 1 | a \| b<br>Returns 15 (`00001111`) |

| Bitwise operators | Description | Example |
|---|---|---|
| ^ (Bitwise XOR) | Compares two bits and returns 1 if the corresponding bit of either, but not both the operands is 1 | a ^ b<br>Returns 7 (00000111) |

Table 15.6: Bitwise Operators

Code Snippet 5 demonstrates the working of the bitwise AND and OR operator in JavaScript.

**Code Snippet 5**:

```
//(56 = 00111000 and 28 = 00011100)
  alert ("56" + ' & ' + "28" + ' = ' + (56 & 28));
//(56 = 00111000 and 28 = 00011100)
  alert ("56" + ' | ' + "28" + ' = ' + (56 | 28));
```

In the code, the bitwise AND operator performs the comparison operation on the corresponding bits of the two operands. It returns 1, if both the bits in that position are 1. The result of these comparisons is an 8-bit binary number, which is automatically converted into integer. This integer is displayed as the output which is 24.

Similarly, the bitwise OR operator performs the comparison operation on the corresponding bits of the two operands. It returns 1, if either of the bits or both the bits in that position is 1. The result of these comparisons is an 8-bit binary number, which is automatically converted into integer. This integer is displayed as the output which is 60.

Code Snippet 6 demonstrates the working of the bitwise XOR and NOT operators in JavaScript.

**Code Snippet 6**:

```
//(56 = 00111000 and 28 = 00011100)
  alert ("56" + ' ^ ' + "28" + ' = ' + (56 ^ 28));
//(28 = 00011100)
  alert ('~' + "28" + ' = ' + (~28));
```

In the code, the bitwise XOR operator performs the comparison operation on the corresponding bits of the two operands. It returns 1, if only 1 of the bits in that position is 1. The result of these comparisons is an 8-bit binary number, which is automatically converted into integer. This integer is displayed as the output that is 36.

Similarly, the bitwise NOT operator inverts each bit, which results in an 8-bit binary number. This binary number automatically converts into integer. This integer is displayed as the output which is -29.

## *15.2.7 Special Operator*

There are some operators in JavaScript which do not belong to any of the categories of JavaScript operators. Such operators are referred to as the special operators.

Table 15.7 lists the most commonly used special operators in JavaScript.

| Special Operator | Description |
|---|---|
| , (comma) | Combines multiple expressions into a single expression, operates on them in the left to right order and returns the value of the expression on the right |
| `?:` (conditional) | Operates on three operands where the result depends on a condition. It is also called as ternary operator and has the form condition, `? value1:value2`. If the condition is true, the operator obtains value1 or else obtains value2 |
| `typeof` | Returns a string that indicates the type of the operand. The operand can be a string, variable, keyword, or an object |

**Table 15.7: Special Operators**

Code Snippet 7 demonstrates the use of special operators to validate the age of a person for voting.

**Code Snippet 7**:

```
<script>
  var age = parseInt(prompt("Enter age", "Age"))
  status = ((typeof(age) == "number" && (age >= 18)) ?"eligible"
  : "not eligible";
    document.write('You are ' + age + ' years old, so you are '
  +status + ' to vote.');
</script>
```

In the code, the `prompt()` function accepts the value from the user. The value is converted into an integer and is stored in the **`age`** variable. The conditional operator specifies a condition before `?` symbol. The condition checks whether the value of the variable **`age`** is a number and whether it is greater than equal to 18. If both these expressions return `true`, the value `eligible` is assigned to the **`status`** variable. Otherwise, the value `not eligible` is assigned to the **`status`** variable. The `alert()` function displays the final output as whether the user is eligible for voting or not eligible.

Figure 15.1 shows the prompt box in which a user enters the value 19 for the **age** variable.

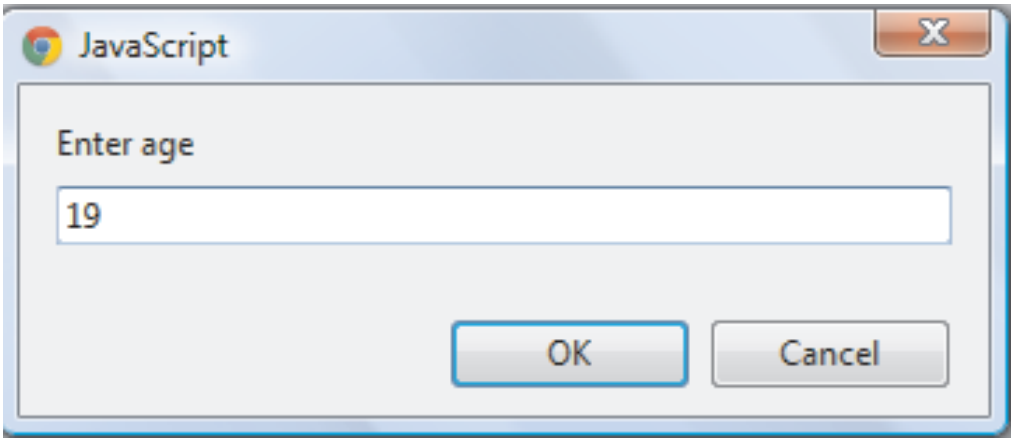



**Figure 15.1: Output – Prompt Box Accepting Value**

Figure 15.2 displays the output indicating that the user is eligible for voting.

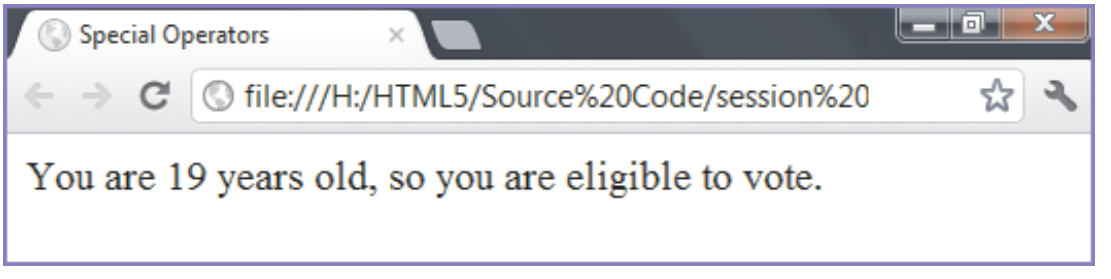



**Figure 15.2: Output – User Eligible for Voting**

## *15.2.8 Operator Precedence*

Operators in JavaScript have certain priority levels based on which their execution sequence is determined. For example, the division operator (/) has a higher priority than the subtraction (–) operator. Therefore, the division operator will be carried out first, if an expression involves both these operators.

Further, an execution order is also defined for the operators within expression. This order is referred to as the associativity, which is either from left to right or vice-versa depending upon the operators.

Table 15.8 lists the precedence of the operators from the highest to the lowest and their associativity.

| Precedence Order | Operator | Description | Associativity |
|---|---|---|---|
| **1** | `()` | Parentheses | Left to Right |
| **2** | `++, --` | Post-increment and Post-decrement operators | Not Applicable |
| **3** | `typeof, ++, --, -, ~, !` | Pre-increment and Pre-decrement operators, Logical NOT, Bitwise NOT, and Unary negation | Right to Left |
| **4** | `*, /, -` | Multiplication, Division, and Modulo | Left to Right |
| **5** | `+, -` | Addition and Subtraction | Left to Right |
| **6** | `<, <=, >, >=` | Less than, Less than or equal, Greater than, and Greater than or equal | Left to Right |
| **7** | `==, ===, !=, !==` | Equal to, Strict equal to, Not equal to, and Strict not equal to | Left to Right |
| **8** | `&, \|, ^, &&, \|\|` | Bitwise AND, Bitwise OR, Bitwise XOR, Logical AND, and Logical OR | Left to Right |
| **9** | `?:` | Conditional operator | Right to Left |
| **10** | `=, +=, -=, *=, /=, %=` | Assignment operators | Right to Left |
| **11** | `,` | Comma | Left to Right |

**Table 15.8: Precedence of Operators**

Code Snippet 8 demonstrates the evaluation of an expression based on its operator's precedence.

**Code Snippet 8**:

```
<SCRIPT>
 var numOne = 96, numTwo = 45, numThree = 200;
 var result;
 result =  2 * (numThree - numOne + numTwo);
 alert('The result is: ' + result);
</SCRIPT>
```

In the code, operators such as `*`, `-`, and `+` are being used to form the expression. According to the rules of operator precedence, parentheses alter the order of evaluation.

Thus, the expression within parentheses is evaluated first and its result is used to evaluate the remaining expression. Since, the + and – operators have same precedence, therefore, the expression within parentheses is evaluated from left to right. Then, the outcome of the expression is multiplied with the value 2 and assigned to the variable **`result`**.

Figure 15.3 shows the result of the evaluated expression.

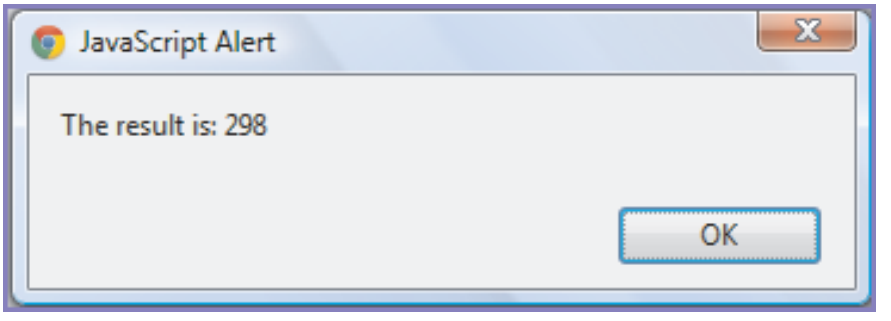



**Figure 15.3: Output of an Expression**

## *15.3 Regular Expression*

A regular expression is a pattern that is composed of set of strings, which is to be matched to a particular textual content. For example, you can specify a pattern for US postal code that the code will not contain more than five digits. When the user enters the postal code, the digits entered by the user will be verified against the pattern to ensure that the postal code is valid.

Regular expressions allow handling textual data effectively, as it allows searching and replacing strings. They allow handling complex manipulation and validation that could otherwise be implemented through lengthy scripts.

In JavaScript, there are two ways to create regular expressions which are as follows:

➔ **Literal Syntax**

A literal refers to a static value. Therefore, a literal syntax allows specifying a fixed pattern, which is stored in a variable. This method of specifying patterns is useful when the Web page designer knows the pattern at the time of scripting.

The syntax to specify the literal syntax is as follows:

**Syntax**:

```
var variable_name = /regular_expression_pattern/;
```

where,

`regular_expression_pattern`: Is a string pattern.

`variable_name`: Is the name of the variable which stores the pattern.

Code Snippet 9 shows the use of literal syntax to create a regular expression pattern.

**Code Snippet 9**:

```
<SCRIPT>
  var ageFormat = /^\d{2}$/;
</SCRIPT>
```

In the code, to match a 2-digit number, the **`ageFormat`** variable is declared. The variable stores the literal pattern. The pattern starts and ends with the forward slashes. The `^` character denotes the starting position of a string, `\d{2}` represents two digits, and `$` character denotes the end of a string. This means that the string must contain only two digits to represent a valid age value.

### ➔ `RegExp()` Constructor

The `RegExp()` constructor is useful when the Web page designer does not know the pattern at the time of scripting. This means that the method dynamically constructs a regular expression when the script is executed. The `RegExp()` constructor is a function that returns a reference to the built-in `RegExp` object.

The syntax to use the `RegExp()` constructor is as follows:

**Syntax**:

```
var variable_name = new RegExp("regular_expression_pattern","flag");
```

where,

`new`: Is a keyword and a special operator that creates the `RegExp` object.

`variable_name`: Is the name of the variable which refers to the `RegExp` object that holds the pattern.

`flag`: Is a letter that specifies whether to search for patterns in the complete string and to consider the casing of characters in the string.

Code Snippet 10 shows the use to `RegExp()` constructor to create a regular expression pattern.

**Code Snippet 10**:

```
<script>
  var nameCheck = new RegExp("^Jo");
</script>
```

In the code, the `RegExp()` constructor creates a search pattern at runtime. The pattern, "`^Jo`" is matched with the specified string to check, if it starts with "`Jo`" at the first position.

## *15.3.1 RegExp Methods and Properties*

The `RegExp` object supports methods that are used for searching the pattern in a string. These methods are as follows:

- **test(string)** – Tests a string for matching a pattern and returns a boolean value of `true` or `false`. The boolean value indicates whether the pattern exists in the string. This method is commonly used for validation.

- **exec(string)** – Executes a string to search the matching pattern within it. The method returns a null value, if pattern is not found. In case of multiple matches, it returns the matched result set.

Apart from these, `RegExp` object also supports properties that are used to get information regarding the string.

Table 15.9 lists the properties of the `RegExp` object.

| Property | Description |
|---|---|
| `$n` | Represents the number from 1 to 9. It stores the recently handled parts of a parenthesized pattern of a regular expression<br>For example, if the pattern for the recent search was `/(w+)(\s+)(w+)/` and the string found was "`Hi Mary`", the value of `RegExp. $1` would contain the value `Hi` |
| `aif` | Indicates whether the given regular expression contain a `g` flag. The `g` flag specifies that all the occurrences of a pattern will be searched globally, instead of just searching for the first occurrence. For example, `/no/g` matches both "`no`"s in "`No health no life`" |
| `aifc` | Indicates whether the given regular expression contains an `i` flag |
| `aiff` | Stores the location of the starting character of the last match found in the string. In case of no match, the value of the property is `-1` |
| `asc` | Stores the copy of the pattern |

**Table 15.9: Properties of RegExp Object**

Code Snippet 11 demonstrates the script to verify whether the entered zip code is valid or not using a regular expression.

**Code Snippet 11**:

```
<script>
var zipcodepattern = /^\d{5}$/;
 var zipcode = zipcodepattern.exec(prompt('Enter ZIP Code:'));
 if(zipcode != null)
 {
  alert('Valid ZIP Code.');
  alert('Regular Expression Pattern: ' + zipcodepattern.source);
 }
 else
 {
    alert('Invalid ZIP Code – Format xxxxx.');
 }
</script>
```

In the code, **`zipcodepattern`** variable stores a pattern that states that the search string must contain only 5 digits. The zip code is accepted from the user and the pattern is matched with the code using the `exec()` method. The `if` condition checks whether the entered zip code is not `null`. If it is not `null`, the `Valid ZIP Code` message and the pattern is displayed or else the `Invalid ZIP Code – Format xxxxx` message is displayed.

## *15.3.2 Categories of Pattern Matching*

There are different categories of pattern matching character that are required to create a regular expression pattern. These categories are as follows:

- Position Matching
- Character Classes
- Repetition
- Alternation and Grouping
- Back Reference

The brief description of these categories is as follows:

➔ **Position Matching**

Characters or symbols in this category allow matching a substring that exists at a specific position within a string.

Table 15.10 lists the various position matching symbols.

| Symbol | Description | Example |
|---|---|---|
| ^ | Denotes the start of a string | /^Good/ matches "Good" in "Good night", but not in "A Good Eyesight" |
| $ | Denotes the end of a string | /art$/ matches "art" in "Cart", but not in "artist" |
| \b | Matches a word boundary. A word boundary includes the position between a word and the space | /ry\b/ matches "ry" in "She is very good" |
| \B | Matches a non-word boundary | /\Ban/ matches "an" in "operand", but not in "anomaly" |

Table 15.10: Position Matching Symbols

➔ **Character Classes**

Characters or symbols in this category are combined to form character classes for specifying patterns. These classes are formed by placing a set of characters within the square brackets. For example, the /[a-zA-Z0-9]/ pattern matches all alphabets and digits.

Table 15.11 lists the various character classes symbols.

| Symbol | Description | Example |
|---|---|---|
| [xyz] | Matches one of the characters specified within the character set | /[BC]RT/<br>Matches "BRT" and "CRT" but, not "RRT", since the leading letter "R" is not specified in the set |
| [^xyz] | Matches one of the characters not specified within the character set | /[^BC]RT/<br>Matches "RRT", but not "BRT" or "CRT" |
| . | Denotes a character except for the new line and line terminator | /s.t/<br>Matches "sat", "sit", "set", and so on |

| Symbol | Description | Example |
|---|---|---|
| \w | Matches alphabets and digits along with the underscore | /\w/<br>Matches "600" in "600%" |
| \W | Matches a non-word character | /\W/<br>Matches "%" in "800%" |
| \d | Matches a digit between 0 to 9 | /\d/<br>Matches "4" in "A4" |
| \D | Searches for a non-digit | /\D/<br>Matches "ID" in "ID 2246" |
| \s | Searches any single space character including space, tab, form feed, and line feed | /\s\w*/<br>Matches " bar" in "scroll bar" |
| \S | Searches a non-space character | /\S\w*/<br>Matches "scroll" in "scroll bar" |

Table 15.11: Character Classes Symbols

Code Snippet 12 uses the position matching symbols, escape sequence characters, and character classes to form a regular expression.

**Code Snippet 12**:

```
<script>
 // To match a string ending with "ing"
    var stringEnd = /ing$/;
 // To match single digit
    var num = /\d/;
</script>
```

The code declares and initializes the **stringEnd** and **num** variables. The pattern for the **stringEnd** variable specifies that the search string must end with letters ing. The $ character denotes the end position of the string. The pattern for the **num** variable specifies that the search string must have at least one digit within it.

### ➔ Repetition

Characters or symbols in this category allow matching characters that reappear frequently in a string.

Table 15.12 lists the various repetition matching symbols.

| Symbol | Description | Example |
|---|---|---|
| `{x}` | Matches x number of occurrences of a regular expression | `/\d{6}/`<br>Matches exactly 6 digits |
| `{x, }` | Matches either x or additional number of occurrences of a regular expression | `/\s{4,}/`<br>Matches minimum 4 whitespace characters |
| `{x,y}` | Matches minimum x to maximum y occurrences of a regular expression | `/\d{6,8}/`<br>Matches minimum 6 to maximum 8 digits |
| `?` | Matches minimum zero to maximum one occurrences of a regular expression | `/l\s?m/`<br>Matches "`lm`" or "`l m`" |
| `*` | Matches minimum zero to multiple occurrences of a regular expression | `/im*/`<br>Matches "`i`" in "`Ice`" and "`imm`" in "`immaculate`", but nothing in "`good`" |
| `+` | Matches one or multiple number of occurrences of a regular expression | `/le+d/`<br>Matches both "`led`" and "`lead`" |

**Table 15.12: Repetition Matching Symbols**

### ➔ Alternation and Grouping

Characters or symbols in this category allow grouping characters as an individual entity or adding the '`OR`' logic for pattern matching.

Table 15.13 lists the various alternation and grouping character symbols.

| Symbol | Description | Example |
|---|---|---|
| `()` | Organizes characters together in a group to specify a set of characters in a string | `/(xyz)+(uvw)/`<br>Matches one or more number of occurrences of "`xyz`" followed by one occurrence of "`uvw`" |
| `\|` | Combines sets of characters into a single regular expression and then matches any of the character set | `/(xy)\|(uv)\|(st)/`<br>Matches "`xy`" or "`uv`" or "`st`" |

**Table 15.13: Alternating and Grouping Matching**

➔ **Back References**

Characters or symbols in this category allow referring back to a sub-expression in the same regular expression. This is useful when matching the remaining sub-expression of a regular expression is based upon the result of matching the previous sub-expression.

Table 15.14 lists the back reference matching symbol.

| Symbol | Description | Example |
|---|---|---|
| `()\n` | Matches a parenthesized set within the pattern, where n is the number of the parenthesized set to the left | `/(\w+)\s+\1/`<br>Matches any word occurring twice in a line, such as "`hello hello`." The `\1` specifies that the word following the space should match the string, which already matched the pattern in the parentheses to the left of the pattern. To refer to more than one set of parentheses in the pattern, you would use `\2` or `\3` to match the appropriate parenthesized clauses to the left. You can have maximum 9 back references in the pattern |

**Table 15.14: Back References**

## *15.4 Decision-making Statements*

Statements are referred to as a logical collection of variables, operators, and keywords that perform a specific action to fulfill a required task. For example, the line of code that declares a variable is a statement. Statements help you build a logical flow of the script. In JavaScript, a statement ends with a semicolon. JavaScript is written with multiple statements, wherein the related statements are grouped together. Such a group of statements is referred to as a block of code and the statements within it are enclosed in curly braces.

Decision-making statements allow implementing logical decisions for executing different blocks to obtain the desired output. They execute a block of statements depending upon a Boolean condition. This condition is an expression that returns either `true` or `false`.

JavaScript supports four decision-making statements which are as follows:

- `if`
- `if-else`
- `if-else if`
- `switch`

## *15.4.1 if Statement*

The `if` statement executes a block of statements based on a logical Boolean condition. If this condition is `true`, the block following the `if` statement is executed. If the condition is `false`, the block after the `if` statement is not executed and the immediate statement after the block is executed.

Figure 15.4 shows the flow of execution for the '`if`' statement.

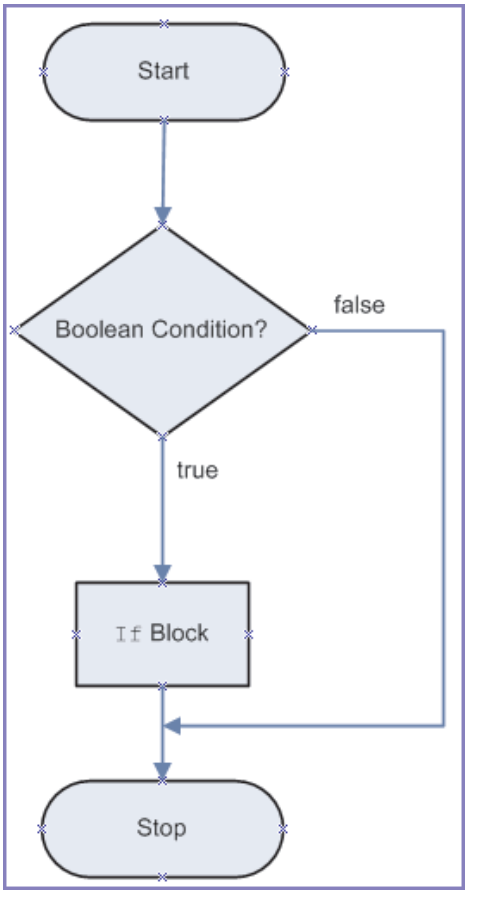



**Figure 15.4: Flow of Execution – if Statement**

The syntax to use the `if` statement is as follows:

**Syntax**:

```
if (condition)
{
  // one or more statements;
}
```

where,

`condition`: Is the boolean expression.

`statements`: Consists of instructions to be executed when the boolean expression is `true`.

Code Snippet 13 checks whether the quantity value is a number and whether it is greater than 0 using the `if` statement.

**Code Snippet 13**:

```
<script>
  var quantity = prompt('Enter quantity of product:',0);
    if(quantity < 0 || isNaN(quantity))
    {
      alert('Please enter a positive number.');
    }
</script>
```

The code accepts a quantity value from the user using the `prompt()` function and stores it in the `quantity` variable. The `if` statement is executed and the value of variable **`quantity`** is checked whether it is less than `0` and whether it is not a number. If value provided by the user is less than `0` or is a value other than a number, the condition evaluates to `true` and the output `Please enter a positive number` is displayed to the user.

## *15.4.2 if-else Statement*

The `if` statement specifies a block of statement to be executed when the condition in the `if` statement is `true`. However, sometimes it is required to define a block of statements to be executed when a condition is evaluated to `false`. This is done using the `if-else` statement.

The `if-else` statement begins with the `if` block, which is followed by the `else` block. The `else` block begins with the `else` keyword followed by a block of statements to be executed upon the `false` condition.

The syntax to use the `if-else` statement is as follows:

**Syntax**:

```
if (condition)
{
  // one or more statements;
}
else
{
  // one or more statements;
}
```

Code Snippet 14 performs the division operation and validates that the divisor is not equal to 0 using the `if-else` statement.

**Code Snippet 14**:

```
<script>
 var firstNumber = prompt('Enter first number:',0);
 var secondNumber = prompt('Enter second number',0);
 var result = 0;
 if (secondNumber == 0)
 {
  alert('ERROR Message: Cannot divide by zero.');
 }
 else
 {
  result = firstNumber/secondNumber;
  alert("Result: " + result);
 }
</script>
```

The code accepts two numbers for the division operation and stores them in the variables **`firstNumber`** and **`secondNumber`** respectively. The `if` statement checks whether the value of the variable **`secondNumber`** is 0. If it is 0, the `alert()` function displays the error message to the user. If the value is not 0, the `else` block is executed, which performs the division operation. The quotient is stored in the **`result`** variable and is displayed to the user.

## *15.4.3 if-else if statement*

The `if-else if` statements allow you to check multiple conditions and specify a different block to be executed for each condition. The flow of these statements begins with the `if` statement followed by multiple `else if` statements and finally by an optional `else` block. The entry point of execution in these statements begins with the `if` statement. If the condition in the `if` statement is `false`, the condition in the immediate `else if` statement is evaluated.

The `if-else if` statements are also referred to as the `if-else if` ladder.

The syntax to use the `if-else if` statements are as follows:

**Syntax**:

```
if (condition)
{
  // one or more statements;
}
else if (condition)
{
  // one or more statements;
}
else
{
  // one or more statements;
}
```

Code Snippet 15 displays the grades according to the percentage value entered by the user using the `if-else if` statements.

**Code Snippet 15**:

```
<script>
  var percentage = prompt('Enter percentage:',0);
  if (percentage >= 60)
  {
      alert ('You have obtained the A grade.');
  }
  else if (percentage >= 35 && percentage < 60)
```

```
{
 alert ('You have obtained the B class.');
}
else
{
 alert ('You have failed');
}
</script>
```

The code accepts the percentage value from the user and stores it in the **`percentage`** variable. The `if` statement checks whether the value of the **`percentage`** variable is greater than or equal to 60. If this is `true`, the user has obtained the `A` grade. If the condition is `false`, the execution control is passed to the `else if` block. Here, the value of the **`percentage`** variable is checked as to whether it is greater than or equal to 35 and less than 60. If this is `true`, the user has obtained the `B` grade. If the condition is `false`, the `else` block is executed.

## *15.4.4 Nested-if Statement*

The nested-if statements comprises multiple `if` statements within an `if` statement. The flow of the nested-if statements starts with the `if` statement, which is referred to as the outer if statement. This outer if statement consists of multiple `if` statements, which are referred to as the inner if statements.

The inner if statements are executed only if the condition in the outer if statement is `true`. Further, each of the inner if statements is executed, but only if the condition in its previous inner if statement is `true`.

The syntax to use the nested if statements.

**Syntax**:

```
if (condition)
{
  // one or more statements;
    if (condition)
   {
     // one or more statements;
       if (condition)
       {
```

```
        // one or more statements;
       }
    }
}
```

Code Snippet 16 validates the username and password using the nested-if statements.

**Code Snippet 16**:

```
<SCRIPT>
 var username = prompt('Enter Username:');
 var password = prompt('Enter Password:');
 if (username != "" && password != "")
 {
  if (username == "admin" && password == "admin123")
  {
    alert('Login Successful');
  }
  else
  {
   alert ('Login Failed');
  }
 }
</SCRIPT>
```

The code accepts the username and password and stores them in the **`username`** and **`password`** variables. The `if` statement checks whether the values of both the variables are not empty. If they are not empty, the inner if statement is executed. The inner if statement checks whether the value of the **`username`** variable is admin and the value of the **`password`** variable is admin123. If this condition is `true`, the `Login Successful` message is displayed to the user. If the condition is `false`, the `else` block is executed.

## *15.4.5 switch-case Statement*

A program becomes quite difficult to understand when there are multiple `if` statements. To simplify coding and to avoid using multiple `if` statements, `switch-case` statement can be used as a different approach to code the same logic. The `switch-case` statement allows comparing a variable or expression with multiple values.

The syntax to use the `switch-case` statement is as follows:

**Syntax**:

```
switch(expression/variable)
{
 case value1:
       // statements;
          break;
 case value2:
       // statements;
          break;
     . . .
 case valueN:
       // statements;
          break;
 default:
       // default statement
}
```

where,

`switch`: Executes a specific case statement that holds the value of the expression or the variable.

`case`: A value and a colon follow the case keyword. The block of a specific case statement is executed when the value of switch expression and the case value are the same. Each case block must end with the `break` keyword.

`break`: Passes the execution control to the statement existing immediately out of the `switch`-case statement. If there is no `break` statement, the next case statement is executed.

`default`: The execution control passes to the default block when none of the case values matches with the switch expression. The default block is the same as the `else` block of the `if-else if` statements.

Code Snippet 17 displays the salary of an employee according to the designation by using the `switch-case` statement.

**Code Snippet 17**:

```
<SCRIPT>
 var designation = prompt('Enter designation:');
 switch (designation)
 {
 case 'Manager':
  alert ('Salary: $21000');
  break;
 case 'Developer':
  alert ('Salary: $16000');
  break;
 default:
  alert ('Enter proper designation.');
  break;
 }
</SCRIPT>
```

The code uses the variable **`designation`** to store the designation of an employee, which is accepted from the user. The `switch` statement takes the value of the **`designation`** variable and this value is matched with the different `case` statements. If the value matches, the particular `case` block is executed, which displays the respective salary. If none of the case values matches with the switch variable, the `default` block is executed.

## *15.5 Check Your Progress*

1. Match the relational operators with their corresponding descriptions.

| Operator | | Description | |
|---|---|---|---|
| a. | Logical AND | 1. | Checks whether the operands are equal and are of the same type |
| b. | Logical OR | 2. | Returns true if either of the operands are true |
| c. | Equal | 3. | Returns true if both the expressions are true |
| d. | Strict Equal | 4. | Checks whether the two operands are identical |

| | | | |
|---|---|---|---|
| (A) | a-3, b-2, c-4, d-1 | (C) | a-4, b-3, c-2, d-1 |
| (B) | a-2, b-3, c-4, d-1 | (D) | a-1, b-2, c-3, d-4 |

2. Which of the following are the correct ways to create a regular expression?

| | | | |
|---|---|---|---|
| (A) | Literal Syntax | (C) | `RegExp` Constructor |
| (B) | `exec()` | (D) | `test()` |

3. Which of the following operators return a decimal value when they are evaluated?

| | | | |
|---|---|---|---|
| (A) | Arithmetic | (C) | Relational |
| (B) | Logical | (D) | Bitwise |

4. Match the character symbols with their corresponding descriptions.

| Symbol | | Description | |
|---|---|---|---|
| a. | `\D` | 1. | Matches alphabets and digits along with the underscore |
| b. | `\d` | 2. | Matches a non-word character |
| c. | `\W` | 3. | Matches a digit between 0 to 9 |
| d. | `\w` | 4. | Matches a non-digit |

| | | | |
|---|---|---|---|
| (A) | a-4, b-1, c-2, d-3 | (C) | a-3, b-2, c-1, d-4 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-4, b-3, c-2, d-1 |

5. The _____ statement allows comparing a variable or expression with multiple values.

| | | | |
|---|---|---|---|
| (A) | `if` | (C) | `if-else` |
| (B) | `if-else if` | (D) | `switch-case` |

## *15.5.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | A, C |
| 3. | D |
| 4. | D |
| 5. | D |

# Summary

- An operator specifies the type of operation to be performed on the values of variables and expressions.
- JavaScript operators are classified into six categories based on the type of action they perform on operands.
- There are six category of operators namely, Arithmetic, Relational, Logical, Assignment, Bitwise, and Special operators.
- Operators in JavaScript have certain priority levels based on which their execution sequence is determined.
- A regular expression is a pattern that is composed of set of strings, which is to be matched to a particular textual content.
- In JavaScript, there are two ways to create regular expressions namely, literal syntax and RegExp() constructor.
- Decision-making statements allow implementing logical decisions for executing different blocks to obtain the desired output.
- JavaScript supports four decision-making statements namely, if, if-else, if-else if, and switch-case statement.

# *Try it Yourself*

1. Write a JavaScript code to accept three numbers from the user and print the largest and the smallest number.

2. Write a JavaScript code that accepts name and password from the user and validates the data entered by the user.

   The validation criteria's are as follows:

   a. The value accepted for the name field should contain only characters with a maximum limit of 10 characters.

   b. Similarly, value for the password field should be a combination of characters, digits, and underscore. Also, the password should start with one or more characters followed by the digits.

3. Write a JavaScript code that accepts the name from the user and greets them depending upon the hour of time. For example, if it is morning, the message displayed to the user will be '`Good Morning <name>`'.

4. Write a JavaScript code that contains an expression for e-mail matching pattern. This pattern is checked with the e-mail address provided by the user and displays whether it is valid or invalid.

5. A school conducts a Mathematics competition for its students. The grade system for the competition is as follows:

   Grade A: Marks greater than 75

   Grade B: Marks in the range 60 to 75

   Grade C: Marks in the range 45 to 60

   Grade D: Marks less than 45

   Write a JavaScript code to accept the marks from the user and display the grade achieved by the student.

# Session - 16

## Loops and Arrays

Welcome to the Session, **Loops and Arrays**.

This session explains about the different type of looping constructs supported by JavaScript. It also explains how to store collection of values using arrays. Finally, it explains the for...in loop which is an extension of for loop.

In this Session, you will learn to:

- Explain while loop
- Explain for loop
- Explain do..while loop
- Explain break and continue statement
- Explain single-dimensional arrays
- Explain multi-dimensional arrays
- Explain for..in loop

## *16.1 Introduction*

Consider a scenario where you want to accept and display ten numbers to the user. Instead of writing the same lines of code again and again for 10 times, you can use loops.

Loops allow you to execute a single statement or a block of statements multiple times. They are widely used when you want to display a series of numbers and accept repetitive input. A loop construct consists of a condition that instructs the compiler the number of times a specific block of code will be executed.

JavaScript supports three types of loops that are as follows:

- `while` Loop
- `for` Loop
- `do-while` Loop

If the condition is not specified within the construct, the loop continues infinitely. Such, loop constructs are referred to as infinite loops.

## *16.2 while Loop*

The `while` loop executes a block of code as long as the given condition remains `true`. The `while` loop begins with the `while` keyword, which is followed by parentheses containing a boolean condition. If this condition returns `true`, the block of statements within the `while` loop are executed. After every iteration, the program control is transferred back to the `while` statement, where the condition is again checked for another round of execution. This process is continued till the specified condition becomes `false`. Once the condition becomes `false`, the `while` statement stops the execution of loop and transfers the control to next statement appearing after the block.

Figure 16.1 shows the flow of execution - `while` loop.

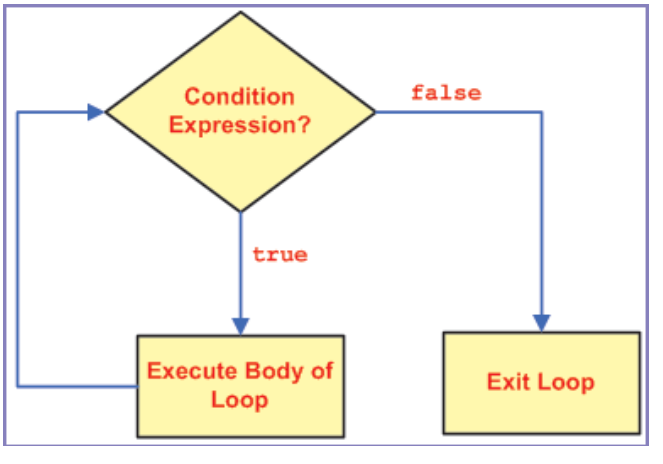



**Figure 16.1: Flow of Execution - while Loop**

The syntax for the `while` loop is as follows:

**Syntax**:

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

where,

`condition`: Is a boolean expression.

Code Snippet 1 displays the sum of numbers from 1 to 10 by using the `while` loop.

**Code Snippet 1**:

```
<script>
var i = 0;
var sum = 0;
 while (i<=10)
 {
   sum = sum + i;
   i = i + 1;
 }
alert ('Sum of first 10 numbers: ' + sum);
</script>
```

The code declares two variables, **`i`** and **`sum`**, which are initialized to value 0. The variable, **`i`**, is a counter variable, whose value increases for every execution of loop. The condition in the `while` loop checks that the value of the counter variable, **`i`**, is less than or equal to 10. If this condition is `true`, the value of the **`sum`** variable is added to the value of **`i`** variable. The value of the variable **`i`** is incremented by 1. Then, the program control is passed to the `while` statement to check the condition again. When the value of **`i`** becomes 11, the `while` loop terminates as the loop condition becomes `false`.

Figure 16.2 shows the output.

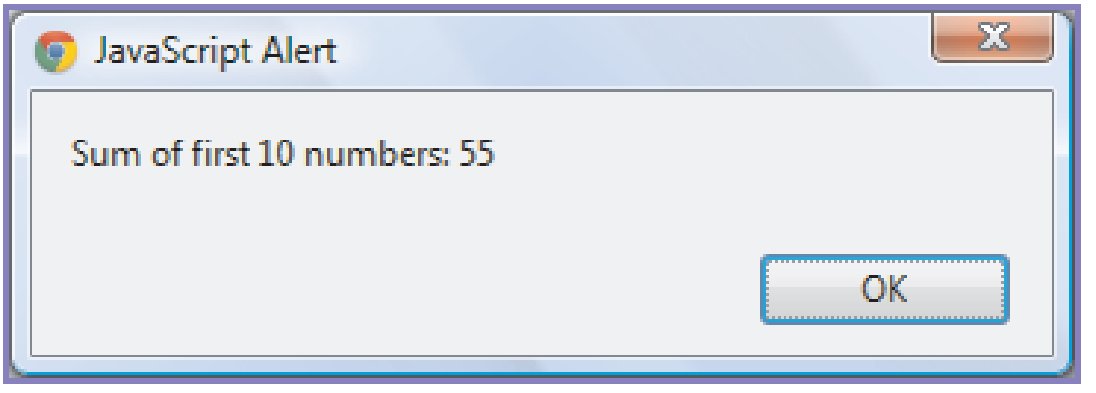



**Figure 16.2: Output**

## *16.3 for Loop*

The `for` loop is similar to the `while` loop in functionality. It executes the statements within the loop as long as the given condition is `true`. Unlike the `while` loop, the `for` loop specifies the loop control statements at the top instead in the body of the loop.

The `for` loop begins with the `for` keyword, which is followed by parentheses containing three expressions, each of which are separated by a semicolon. The three expressions are referred to as **initialization expression**, **condition expression**, and **increment/decrement expression** respectively. These three expressions are optional.

The syntax for the `for` loop is as follows:

**Syntax**:

```
for (initialization; condition; increment/decrement)
{
// statements;
}
```

where,

`initialization`: Initializes the variable(s) that will be used in the condition.

`condition`: Comprises the condition that is checked, before the statements in the loop are executed.

`increment/decrement`: Comprises the statement that changes the value of the variable(s) on every successful execution of the loop to ensure that the condition specified in the condition section is reached. The increment and decrement operators, such as `++`, `--`, and shortcut operators: `+=` or `-=` are used in this section.

Figure 16.3 shows the `for` loop.

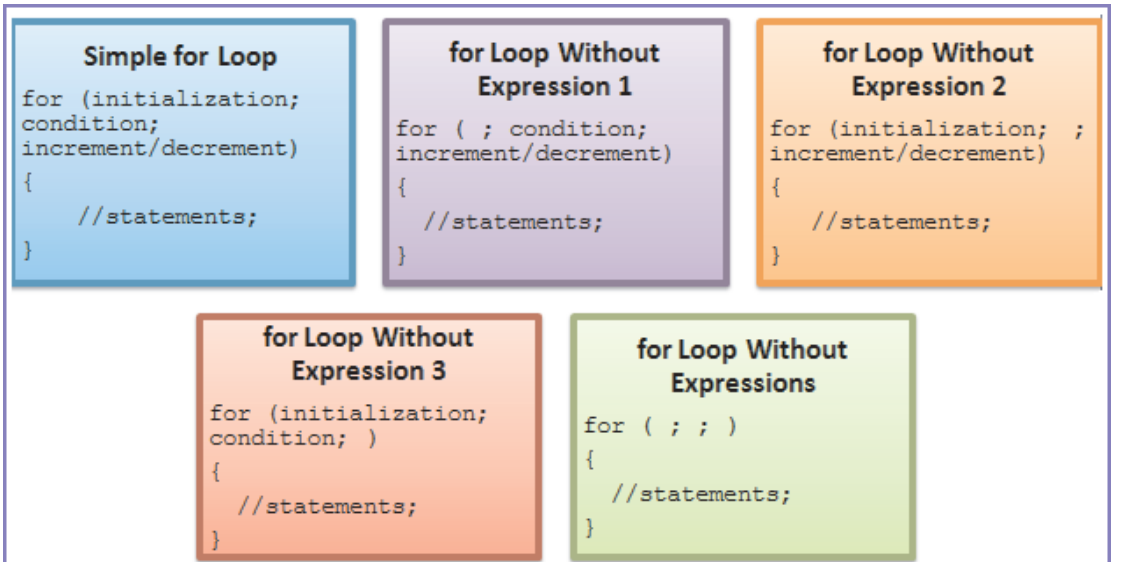



Figure 16.3: for Loop

Code Snippet 2 demonstrates the script that accepts a number from the user and displays the first ten multiples of that number.

**Code Snippet 2**:

```
<script>
var inputNum=prompt('Enter any number:');
var result=0;
document.write ('Multiples of: '+inputNum+'<br />');
 for (var i=1; i<=10; i++)
 {
  result=inputNum*i;
  document.write (inputNum+' * '+i+' = '+result+'<br />');
 }
</script>
```

In the code, a variable, **`inputNum`**, is created and initialized to the value specified by the user in the prompt box. The `for` loop declares a variable, **`i`**, and initializes it to the value 1. If the condition is `true`, the number specified by the user is multiplied to the value of **`i`** variable and the result is appended to the result variable. The program control is again passed to `for` statement, where the value of **`i`** is incremented. The incremented value is again checked with the specified condition and it is multiplied to the number specified by the user. This process continues till the value of **`i`** becomes 11.

Figure 16.4 shows the multiples of a number.

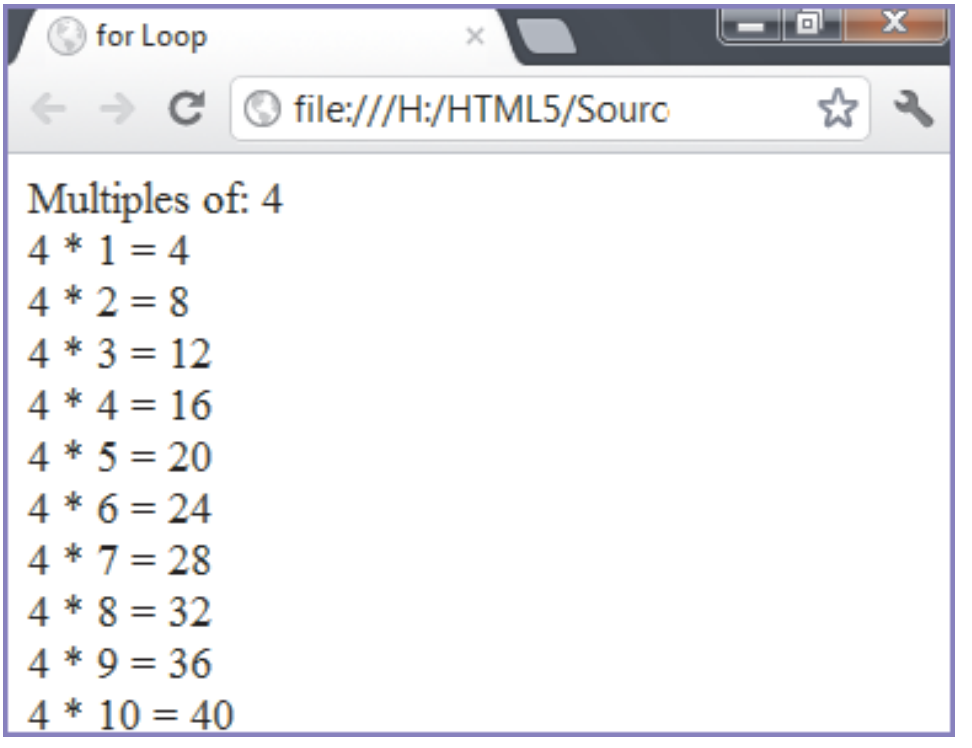



**Figure 16.4: Multiples of a Number**

## *16.4 do-while Loop*

The `do-while` loop is similar to the while loop. This is because both the `do-while` and `while` loops execute until the condition becomes `false`. However, the `do-while` loop differs by executing the body of the loop at least once before evaluating the condition. Thus, even if the condition is `false`, the `do-while` loop executes at least once.

The `do-while` loop starts with the `do` keyword and is followed by a block of statements. At the end of the block, the `while` keyword is specified that is followed parentheses containing the condition. When the specified condition returns `false`, the block of statements after the `do` keyword are ignored and the next statement following the `while` statement is executed.

Figure 16.5 shows the `do-while` loop.

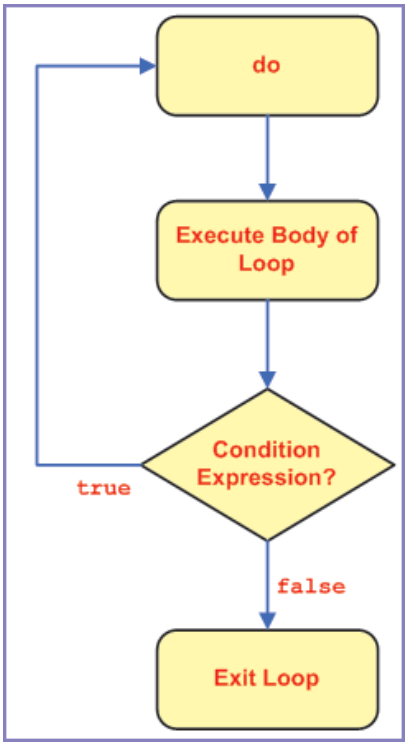



Figure 16.5: do-while Loop

The syntax for the `do-while` loop is as follows:

**Syntax**:

```
do
{
...
statements;
...
}while(condition);
```

where,

`condition`: Is a boolean expression.

Code Snippet 3 demonstrates the script to accept the capital of United States from the user using the `do-while` loop.

**Code Snippet 3**:

```
<script>
 var answer = '';
 do
 {
     answer = prompt('Capital of United States:', '');
 }while(answer!='Washington');


 alert('Capital of United States: ' + answer);
</script>
```

The code declares a variable, **`answer`**, which stores the string entered by the user. The `do` block displays a prompt box without checking any condition. The prompt box accepts the capital of United States and stores this string in the variable, **`answer`**. The condition is specified in the `while` block that checks, if the user has entered the string **`Washington`**. If this condition is `true`, prompt box is closed; else the prompt box is again displayed to accept the user input.

Figure 16.6 shows the output of capital of United States.

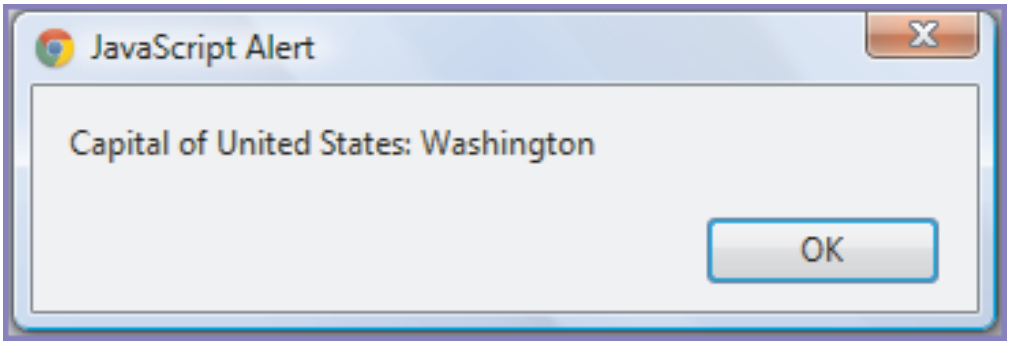



**Figure 16.6: Output of Capital of United States**

## *16.5 break Statement*

The `break` statement can be used with decision-making statements, such as `switch-case` and loop constructs, such as `for` and `while` loops. The `break` statement is denoted by using the `break` keyword. It is used to exit the loop without evaluating the specified condition. The control is then passed to the next statement immediately after the loop.

Figure 16.7 shows the flow of execution of the `break` statement.

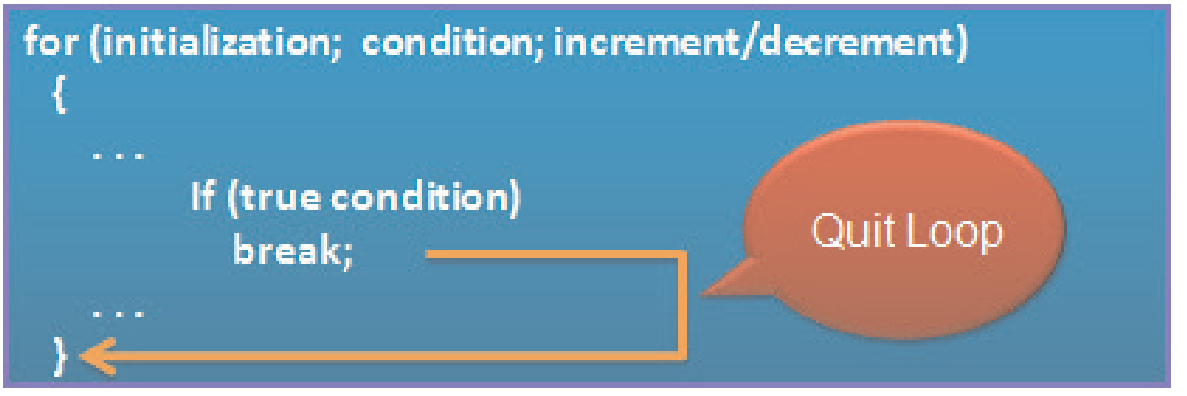



Figure 16.7: Flow of Execution - break Statement

Code Snippet 4 demonstrates the script that accepts a number from the user and determines if it is a prime number or not.

**Code Snippet 4**:

```
<script>
var inputNum=parseInt(prompt('Enter number: ',''));
var num=2;
 while(num<=inputNum-1)
 {
   if(inputNum%num==0)
   {
       alert(inputNum+'is not a Prime Number');
        break;
   }
  num++;
 }
 if(num==inputNum)
 {
  alert (inputNum+'is a Prime Number');
 }
</script>
```

The code creates a variable, `inputNum`, which is initialized to the number entered by the user.

The variable **num** is declared and initialized to 2. If the `while` condition returns `true`, the inner `if` statement is checked. If this condition returns `true`, an alert box is displayed stating that the number is not a prime number. The `break` statement is used to exit the entire `while` loop. If the condition evaluates to `false`, the program control is passed to `if` statement outside the `while` loop.

Figure 16.8 shows the output of the prime number on accepting number, 6 from the user in the prompt box.

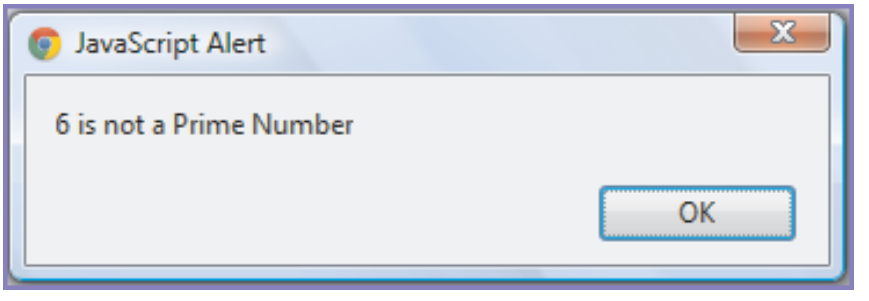



**Figure 16.8: Output - Prime Number**

## *16.6 continue Statement*

The `continue` statement is mostly used in the loop `constructs`. The `continue` statement is denoted by the `continue` keyword. It is used to terminate the current execution of the loop and `continue` with the next repetition by returning the control to the beginning of the loop. This means, the `continue` statement will not terminate the loop entirely, but terminates the current execution.

Figure 16.9 shows the flow of execution of the `continue` statement.

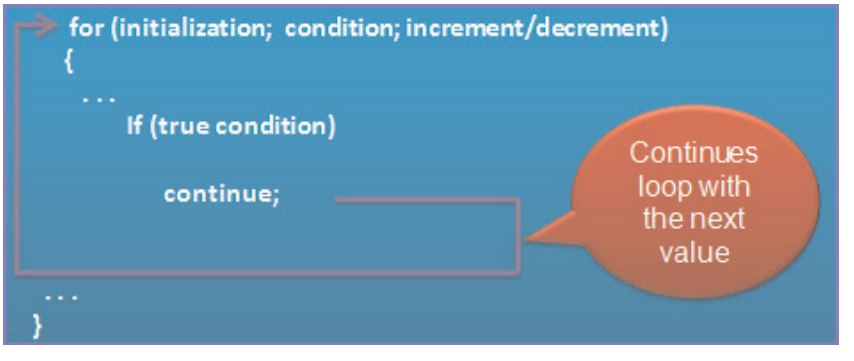



**Figure 16.9: continue Statement**

Code Snippet 5 displays even numbers from 0 to 15.

**Code Snippet 5**:

```
<script>
 var result = '';
 for (var i = 0; i <= 15; i++)
 {
   if((i%2) != 0)
   {
     continue;
   }
  result = result + i + '\n';
 }
 alert ('Even Numbers:\n' + result);
</script>
```

The code declares a variable, **`i`**, in the `for` loop definition and initializes it to value 1. When the value of **`i`** is divided by zero, the `if` statement checks whether the remainder is equal to zero. If the remainder is zero, the value of **`i`** is displayed as the value is an even number. If the remainder is not equal to 0, the `continue` statement is executed. It transfers the program control to the beginning of the `for` loop.

Figure 16.10 shows the output of the `continue` statement.

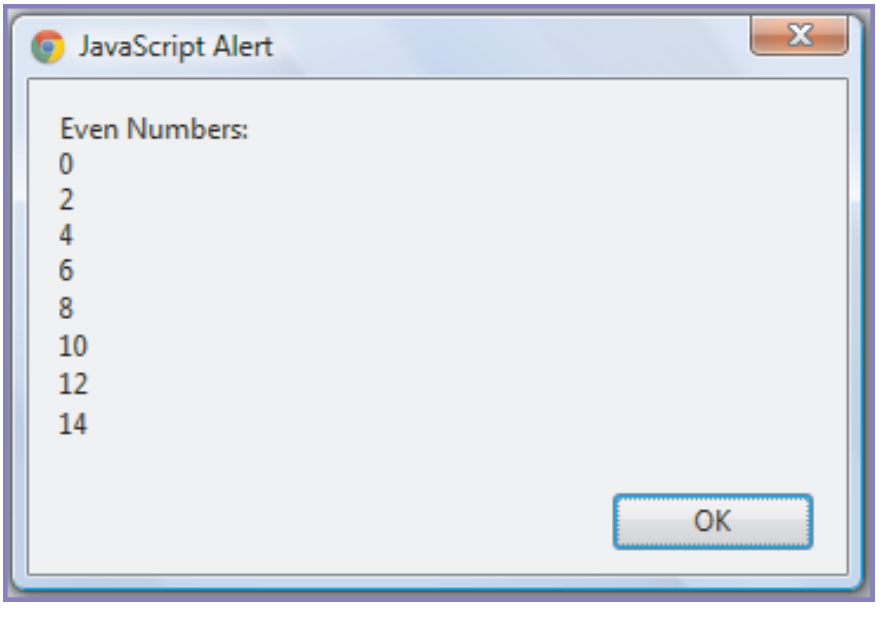



**Figure 16.10: Output – continue Statement**

## *16.7 Arrays*

Consider a scenario where you want to store the names of 100 employees within an IT department. This can be done by creating 100 variables and storing the names. However, keeping track of 100 variables is a tedious task and it results in inefficient memory utilization. The solution to this problem is to create an array variable to store the names of 100 employees.

Figure 16.11 shows the effective usage of memory achieved using an array.

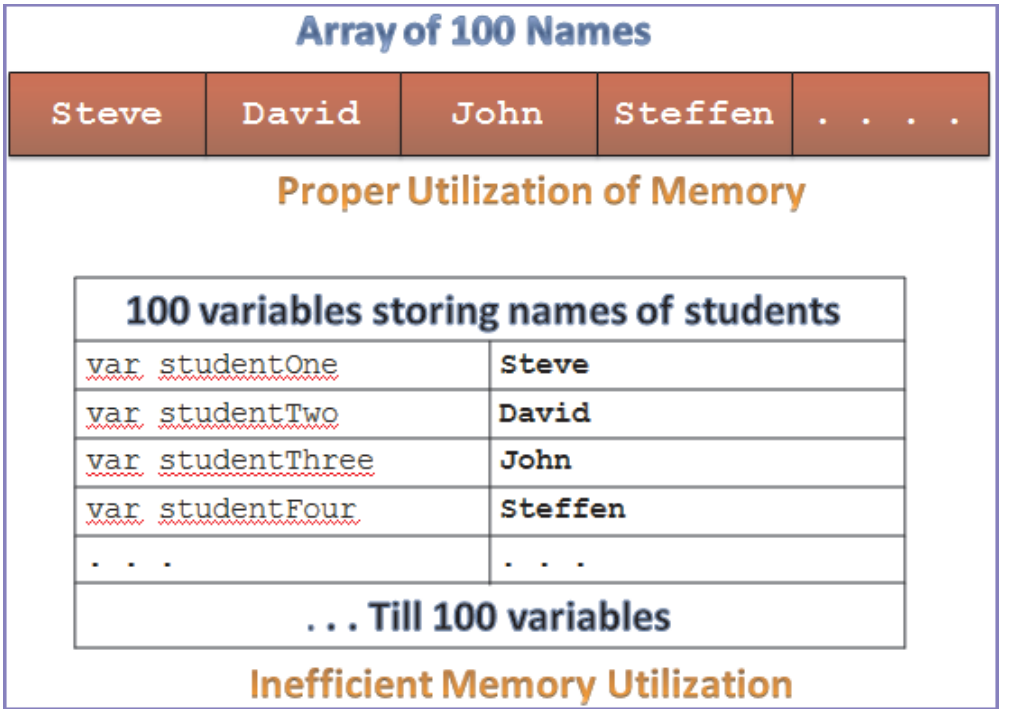



Figure 16.11: Memory Utilization Using an Array

An array is a collection of values stored in adjacent memory locations. These array values are referenced using a common array name. The values of an array variable must be of the same data type. These values that are also referred to as elements and can be accessed by using subscript or index numbers. The subscript number determines the position of an element in an array list.

JavaScript supports two types of arrays that are as follows:

- Single-dimensional array
- Multi-dimensional array

## *16.7.1 Single-dimensional Array*

In a single-dimensional array, the elements are stored in a single row in the allocated memory.

Figure 16.12 shows the allocation of single-dimensional array.

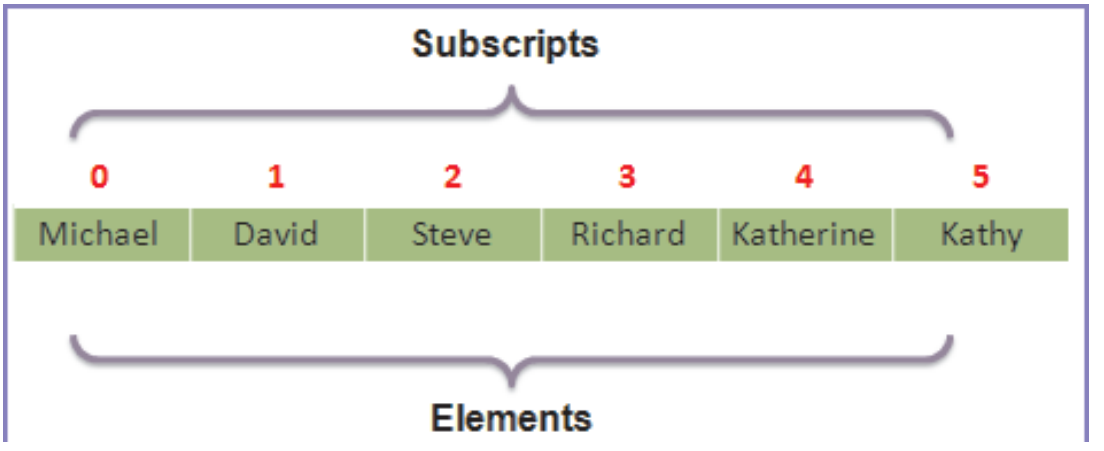



Figure 16.12: Single-dimensional Array

As shown in figure 16.12, the first array element has the index number zero and the last array element has an index number one less than the total number of elements. This arrangement helps in efficient storage of data. In addition, it also helps to sort data easily and track the data length.

The array variable can be created using the `Array` object and `new` keyword along with the size of the array element.

The syntax to declare and initialize a single-dimensional array is as follows:

**Syntax**:

```
var variable_name = new Array(size); //Declaration
variable_name[index] = 'value';
```

where,

`variable_name`: Is the name of the variable.

`size`: Is the number of elements to be declared in the array.

`index`: Is the index position.

Code Snippet 6 shows the different ways to declare and initialize a single-dimensional array.

**Code Snippet 6**:

```
<script>
 //Declaration using Array Object and then Initialization
 var marital_status = new Array(3);
 marital_status[0] = 'Single';
 marital_status[1] = 'Married';
 marital_status[2] = 'Divorced';
 //Declaration and Initialization
 var marital_status = new Array('Single','Married','Divorced');
 //Declaration and Initialization Without Array
 var marital_status = ['Single','Married','Divorced'];
     </script>
```

## *16.7.2 Accessing Single-dimensional Arrays*

Array elements can be accessed by using the array name followed by the index number specified in square brackets.

➔ **Access Array Elements Without Loops**

An array element can be accessed without using loops by specifying the array name followed by the square brackets containing the index number.

Code Snippet 7 demonstrates a script that stores and displays names of the students using a single-dimensional array.

**Code Snippet 7**:

```
<script>
var names = new Array("John", "David", "Kevin");
alert('List of Student Names:\n' + names[0] + ',' + ' ' + names[1] + ',' + ' ' +
names[2]);
</script>
```

In the code, `var names = new Array("John","David","Kevin");` declares and initializes an array. The `names[0]` accesses the first array element which is `John`. The `names[1]` accesses the second array element which is `David`.

The `names[2]` accesses the third array element, which is `Kevin`.

Figure 16.13 displays the names of the students.

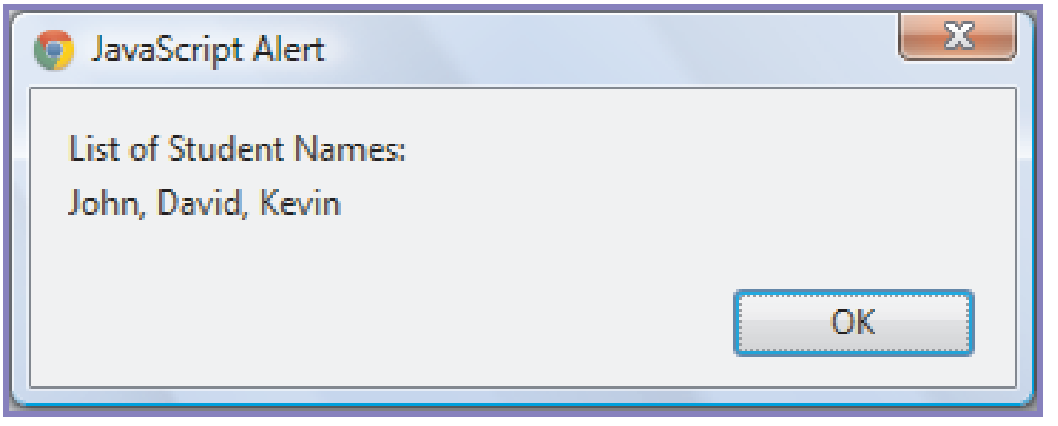



Figure 16.13: Output – Single-dimensional Array

### ➔ Access Array Elements With Loops

JavaScript allows you to access array elements by using different loops. Thus, you can access each array element by putting a counter variable of the loop as the index of an element. However, this requires the count of the elements in an array. So, the `length` property can be used to determine the number of elements in an array.

Code Snippet 8 demonstrates the script that creates an array to accept the marks of five subjects and display the average.

**Code Snippet 8:**

```
<script>
 var sum = 0;
 var marks = new Array(5);
 for(var i=0; i<marks.length; i++)
 {
      marks[i] = parseInt(prompt('Enter Marks:', ''));
       sum = sum + marks[i];
 }
 alert ('Average of Marks: ' + (sum/marks.length));
</script>
```

In the code, `var marks = new Array(5);` declares an array of size 5. It displays a prompt box that accepts the marks for a subject in each iteration. Then, the code calculates and displays the average marks.

Figure 16.14 displays the average of the marks, 90, 75, 85, 95, and 82 accepted from the user in the prompt box.

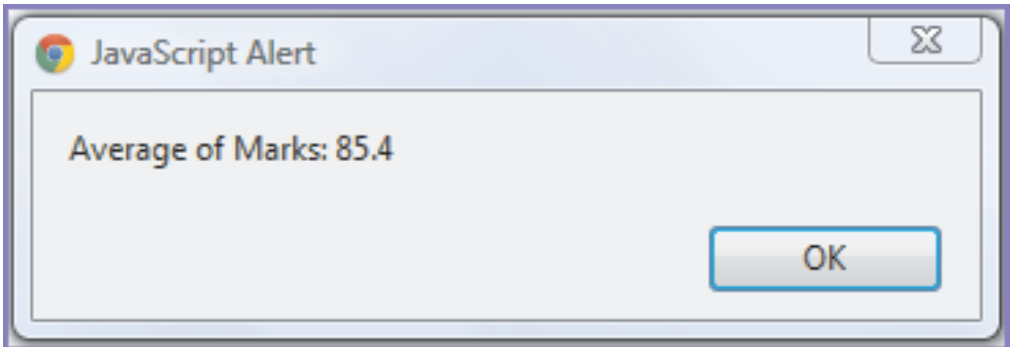



Figure 16.14: Average Marks of Five Subjects

## *16.7.3 Multi-dimensional Array*

Consider a scenario to store the employee IDs of 100 employees and their salary structure. The salary structure will include the basic salary, allowances, HRA, and the total gross salary. Now, if a single-dimensional array is used, then two separate arrays need to be created for storing employee IDs and salaries. However, using a multi-dimensional array, both IDs and salaries are stored in just one array.

A multi-dimensional array stores a combination of values of a single type in two or more dimensions. These dimensions are represented as rows and columns similar to those of a Microsoft Excel sheet. A two-dimensional array is an example of the multi-dimensional array.

Figure 16.15 shows the representation of a multi-dimensional array.

| Employee Salaries → ↓ | 0 BASIC | 1 HRA | 2 ALLOWANCE | 3 TOTAL |
|---|---|---|---|---|
| 0 | 14350 | 10500 | 1500 | 26350 |
| 1 | 34350 | 4050 | 1000 | 39400 |
| 2 | 6150 | 4500 | 3250 | 13900 |
| 3 | 4920 | 4500 | 2250 | 11670 |
| 4 | 12300 | 9000 | 2000 | 23300 |

Figure 16.15: Multi-dimensional Array

A two-dimensional array is an array of arrays. This means, for a two-dimensional array, first a main array is declared and then, an array is created for each element of the main array.

The syntax to declare a two-dimensional array is as follows:

**Syntax**:

```
var variable_name = new Array(size); //Declaration
variable_name[index] = new Array('value1','value2'..);
```

where,

`variable_name`: Is the name of the array.

`index`: Is the index position.

`value1`: Is the value at the first column.

`value2`: Is the value at the second column.

Figure 16.16 shows the declaration of a two-dimensional array.

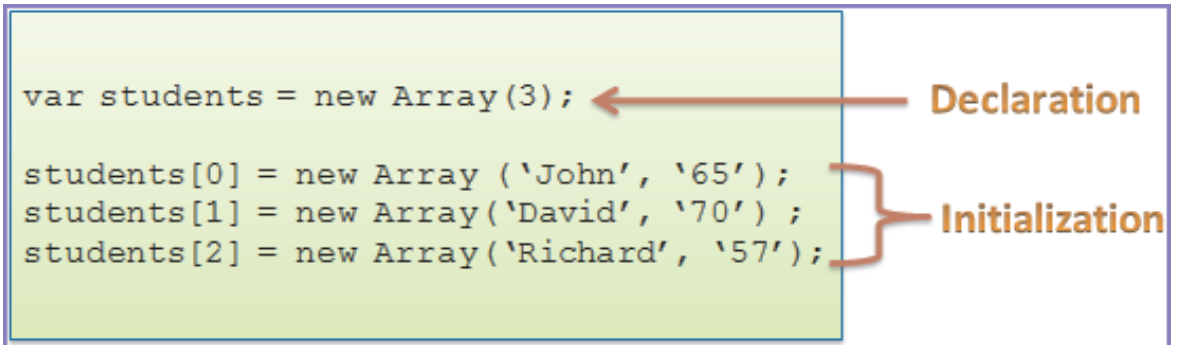



Figure 16.16: Declaration of Two-dimensional Array

## *16.7.4 Accessing Two-dimensional Arrays*

Multi-dimensional arrays can be accessed by using the index of main array variable along with index of sub-array.

### ➔ Access Array Elements Without Loops

Code Snippet 9 creates a two-dimensional array that displays the employee details.

**Code Snippet 9**:

```
<script>
 var employees = new Array(3);
 employees[0] = new Array('John', '25', 'New Jersey');
 employees[1] = new Array('David', '21', 'California');
 document.write('<H3> Employee Details </H3>');
 document.write('<P><B>Name: </B>' + employees[0][0] + '</P>');
 document.write('<P><B>Location: </B>' + employees[0][2] + '</P>');
 document.write('<P><B>Name: </B>' + employees[1][0] + '</P>');
 document.write('<P><B>Location: </B>' + employees[1][2] + '</P>');
</script>
```

In the code, `var employees = new Array(3)` creates an array of size 3. The declaration `employees[0] = new Array('John','23','New Jersey')` creates an array at the 0th row of the **`employees`** array. Similarly, `employees[1]=new Array('David','21','California')` creates an array at the first row of the **`employees`** array.

Figure 16.17 displays the employee details.

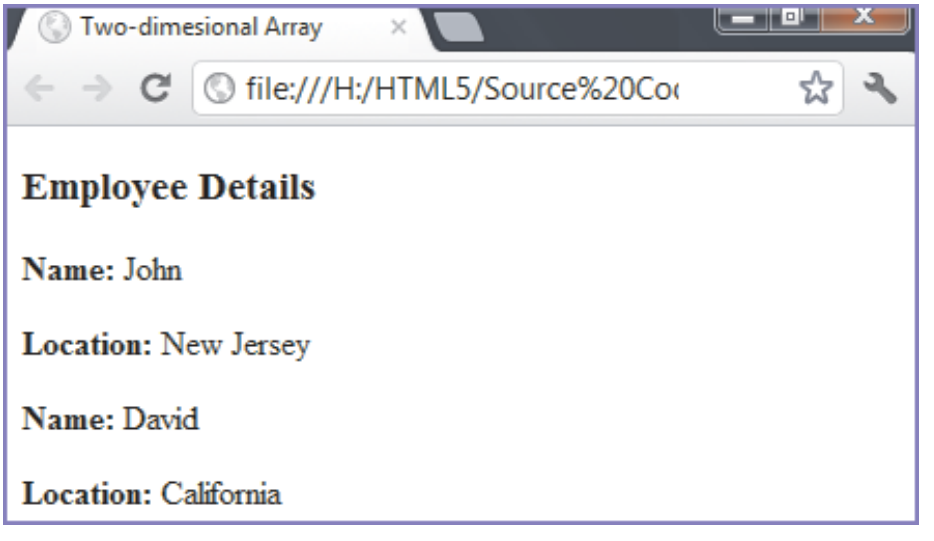



Figure 16.17: Output - Employee Details

### ➔ Access Array Elements With Loops

Code Snippet 10 creates a two-dimensional array to display the product details.

**Code Snippet 10**:

```
<script>
 var products = new Array(2);
 products[0] = new Array('Monitor', '236.75');
 products[1] = new Array('Keyboard', '45.50');
 document.write('<TABLE border=1><TR><TH>Name</TH><TH>
                                    Price</TH></TR>');
 for(var i=0; i<products.length; i++)
 {
    document.write('<TR>');
     for(var j=0; j<products[i].length; j++)
    {
      document.write('<TD>' + products[i][j] + '</TD>');
     }
    document.write('</TR>');
 }
 document.write('</TABLE>');
</script>
```

In the code, `products[0] = new Array('Monitor','236.75')` creates an array at the 0th row of the **`products`** array. Similarly, `products[1] = new Array('Keyboard','45.50')` creates an array at the first row of the `products` array. The condition, **`i < products.length`**, specifies that the counter variable **`i`** should be less than the number of rows in the array variable, **`products`**. For each row in the array, the condition, **`j < products[i].length`** specifies that the counter variable **`j`**, should be less than the number of columns specified the **`i`**th row of the array variable, **`products`**. Finally, `document.write("<TD>" + products[i][j] + "</TD>")` displays the values at the **`i`**th row and **`j`**th column of array variable, **`products`**.

Figure 16.18 displays the product details in a table.

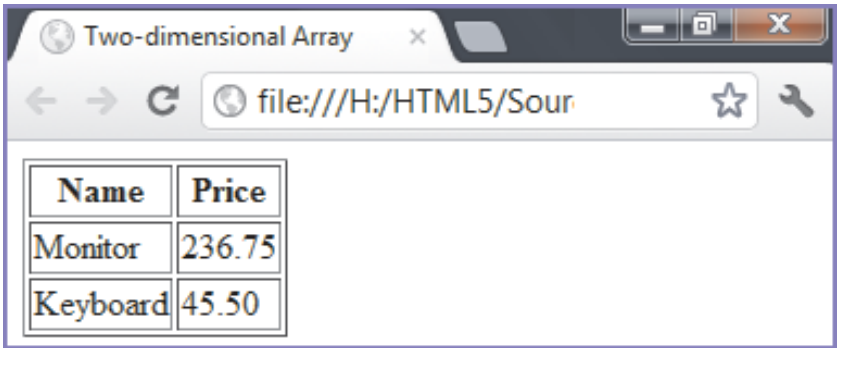



Figure 16.18: Output – Product Details in a Table

## *16.8 Array Methods*

An array is a set of values grouped together and identified by a single name. In JavaScript, the `Array` object allows you to create arrays. It provides the `length` property that allows you to determine the number of elements in an array. The various methods of the `Array` object allow to access and manipulate the array elements.

Table 16.1 lists the most commonly used methods of the `Array` object.

| Method | Description |
|---|---|
| `concat` | Combines one or more array variables |
| `join` | Joins all the array elements into a string |
| `pop` | Retrieves the last element of an array |
| `push` | Appends one or more elements to the end of an array |
| `sort` | Sorts the array elements in an alphabetical order |

**Table 16.1: Methods of Array Object**

Code Snippet 11 demonstrates how to access and manipulate the array elements.

**Code Snippet 11**:

```
<script>
 var flowers = new Array('Rose', 'Sunflower', 'Daisy');
 document.write('Number of flowers: '+flowers.length+'<br/>');
 document.write('Flowers: '+flowers.join(', ')+'<br/>');
 document.write('Orchid and Lily are Added: '+flowers.push("Orchid", "Lily")+
'<br/>');
 document.write('Flowers (In Ascending Order): '+flowers.sort()+'<br/>');
 document.write('Flowers Removed: '+flowers.pop()+'<br/>');
</script>
```

Figure 16.19 displays the corresponding output of the script.

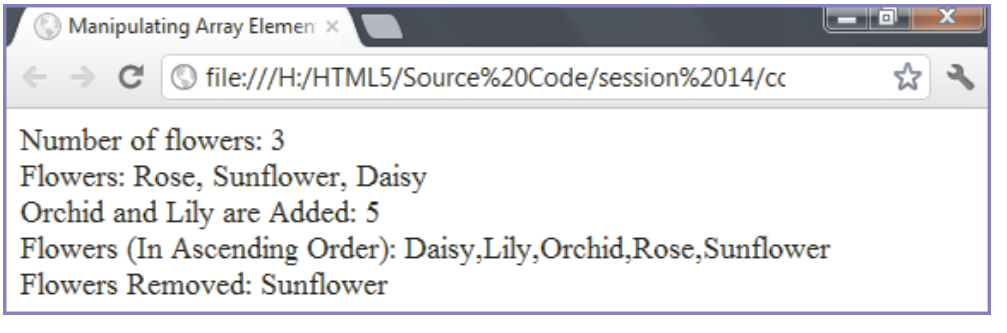



**Figure 16.19: Output – Manipulating Array Elements**

In the code, an array variable **flowers** is created, that stores the names of three flowers. The `length` property is used to display the number of flowers in the array variable. The `join()` method joins the flower names and separates them with a comma. The `push()` method adds orchid and lily at the end of the array and the total number of flowers in the array list is displayed as 5. The `sort()` method sorts the flowers in alphabetical order. The `pop()` method retrieves the last element that is `Sunflower`, from the array list.

## *16.9 for...in Loop*

The `for...in` loop is an extension of the `for` loop. It enables to perform specific actions on the arrays of objects. The loop reads every element in the specified array and executes a block of code only once for each element in the array.

The syntax of the `for...in` loop is as follows:

**Syntax**:

```
for (variable_name in array_name)
{
     //statements;
}
```

where,

`variable_name`: Is the name of the variable.

`array_name`: Is the array name.

Code Snippet 12 demonstrates how to display elements from an array using the `for...in` loop.

**Code Snippet 12**:

```
<script>
 var books = new Array('Beginning CSS 3.0', 'Introduction to HTML5', 'HTML5 in
Mobile Development');


 document.write('<H3> List of Books </H3>');
 for(var i in books)
 {
  document.write(books[i] + '<br/>');
 }
</script>
```

Figure 16.20 displays the corresponding output of the script.

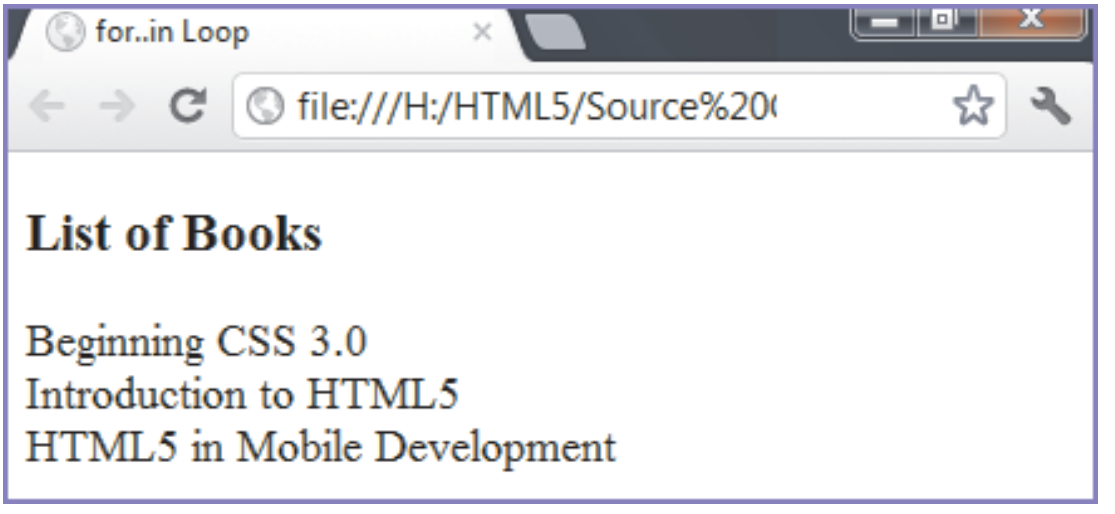



**Figure 16.20: Output - for...in Loop**

## *16.10 Check Your Progress*

1. The _____ loop executes a block of code until the condition becomes false.

| | | | |
|---|---|---|---|
| (A) | for | (C) | do-while |
| (B) | while | (D) | do-until |

2. Match the array methods with their corresponding descriptions.

| Method | | Description | |
|---|---|---|---|
| (A) | push | (1) | Joins all the array elements into a string |
| (B) | pop | (2) | Appends one or more elements to the end of an array |
| (C) | concat | (3) | Retrieves the last element of an array |
| (D) | join | (4) | Combines one or more array variables |

| | | | |
|---|---|---|---|
| (A) | a-4, b-1, c-2, d-3 | (C) | a-2, b-3, c-4, d-1 |
| (B) | a-1, b-2, c-3, d-4 | (D) | a-3, b-2, c-4, d-1 |

3. Which of the following statement terminates the current execution, but does not terminate the execution of the loop entirely?

| | | | |
|---|---|---|---|
| (A) | halt | (C) | break |
| (B) | continue | (D) | exit |

4. Which of the following option is not a valid declaration of the for loop?

| | | | |
|---|---|---|---|
| (A) | `for (initialization; condition; increment/decrement)` | (C) | `for (;condition; increment/decrement)` |
| (B) | `for (initialization; condition; )` | (D) | `for (condition; increment/decrement)` |

5. Identify the correct ways to create an array variable in the JavaScript.

| | | | |
|---|---|---|---|
| (A) | Object | (C) | Array |
| (B) | new | (D) | String |

## *16.10.1 Answers*

| | |
|---|---|
| 1. | B |
| 2. | C |
| 3. | B |
| 4. | D |
| 5. | B, C |

# Summary

- A loop construct consists of a condition that instructs the compiler the number of times a specific block of code will be executed.
- JavaScript supports three types of loops that include: while loop, for loop, and do-while loop.
- The break statement is used to exit the loop without evaluating the specified condition.
- The continue statement terminates the current execution of the loop and continue with the next repetition by returning the control to the beginning of the loop.
- JavaScript supports two types of arrays namely, Single-dimensional array and Multi-dimensional array.
- The for..in loop is an extension of the for loop that enables to perform specific actions on the arrays of objects.

# *Try it Yourself*

1. Write a JavaScript code to accept a number from the user and check whether the number is armstrong or not.

   An armstrong number is a number whose sum of cubes of individual digit is same as the number itself. For example, 153 is an armstrong number, as the sum of $1^3 + 5^3 + 3^3 = 153$.

2. Write a JavaScript code to create an array that store names of 10 students in it. Then, display the names of the students in an alphabetic order from the array. Also, use the for..in loop to display the name of the students from the array.

3. Write a JavaScript code to print the transpose of a matrix. Transpose of a matrix is where the rows are interchanged with columns and vice versa.

4. Write a JavaScript code to display the following pattern on the Web page:

```
*
***
*****
*******
*********
***********
```

Onlinevarsity

# Session - 17

## Functions and Objects

Welcome to the Session, **Functions and Objects**.

This session explains functions, which are independent reusable blocks of code, executed on the occurrence of an event. The session also explains the concept of objects used for storing and manipulating entities in JavaScript. Finally, it explains different types of built-in and browser objects supported by JavaScript.

In this Session, you will learn to:

- Explain functions
- Explain parameterized functions
- Explain return statement
- Describe objects
- Explain different browser objects
- Describe DOM and its objects

## *17.1 Introduction*

Consider a scenario where a Web page has been designed to greet the user with his/her name on the click of a button. A code can be used here to accomplish this task, but may result in the same output on repetitive execution. However, writing these statements each time for the same action is tedious, time consuming, and error prone.

To make the code more task-oriented and manageable, JavaScript allows to group statements before they are actually invoked. This can be achieved by using the concept of functions. A function is a reusable block of code that is executed on the occurrence of an event. The event can be a user action on the page or a call within the script.

## *17.2 Functions*

A function is an independent reusable block of code that performs certain operations on variables and expressions to fulfill a task. A function might take parameters, which are variables or values on which it performs operations. After performing operations, a function might return the resultant value to display it in the browser. For example, a function named `Add()` might take two numbers on which the addition operation will be performed and will return the result of addition.

A JavaScript function is always created under the `script` element. JavaScript supports both user-defined and built-in functions.

Figure 17.1 shows the functions in an HTML page.

**Figure 17.1: Functions in HTML Page**

## *17.2.1 Declaring and Defining Functions*

JavaScript allows declaring a function using the `function` keyword. The keyword is followed by the name of the function and parameters enclosed within the parenthesis. If the function does not take any parameters, then it must be specified with the empty parenthesis.

Once the function is declared, you need to define the function by specifying the operations or instructions within the curly braces `{` and `}`. These curly braces indicate the start and end of the function block, which is collectively referred to as the body of the function.

There are certain conventions that must be followed for naming functions. They are as follows:

- ➔ Can consist of letter, digits, and underscore
- ➔ Can begin only with a letter or an underscore
- ➔ Cannot be a JavaScript keyword
- ➔ Cannot begin with a digit
- ➔ Cannot contain spaces

The syntax to create a function in JavaScript is as follows:

**Syntax**:

```
function function_name(list of parameters)
{
   // Body of the function
}
```

where,

`function`: Is a keyword used to declare a function.

`function_name`: Indicates the name of the function.

`list of parameters`: Is optional and specifies the parameters to be passed to the function separated by commas.

A function must be defined, before it can be invoked in the script. Also, there can be multiple functions defined within the `script` element.

Figure 17.2 shows the declaration and definition of a function.

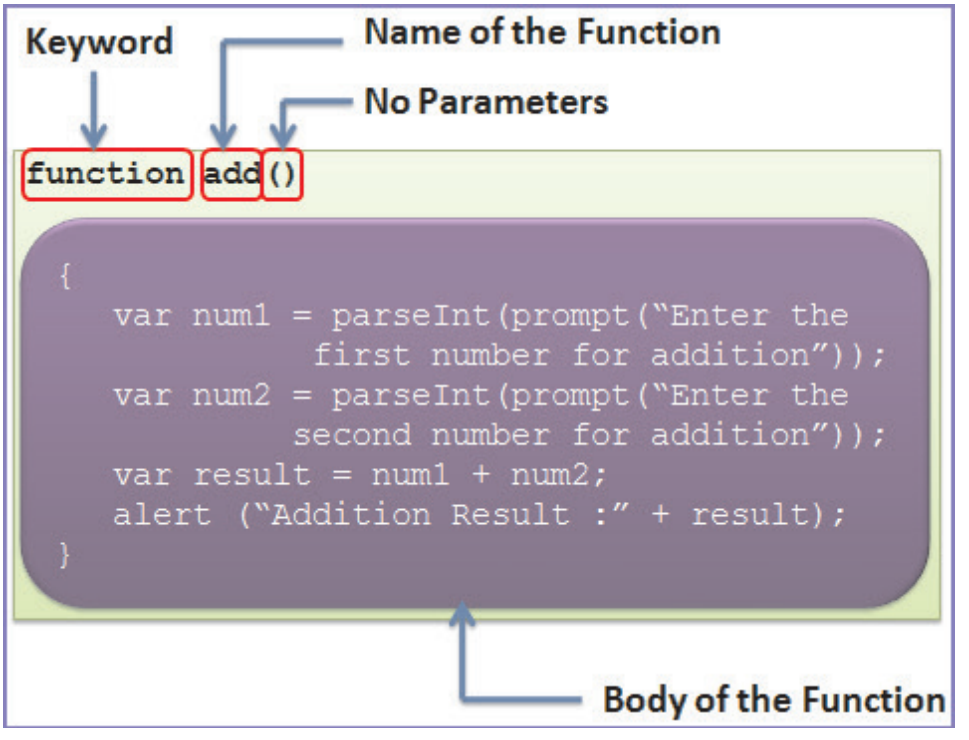



Figure 17.2: Declaration and Definition of a Function

## *17.2.2 Invoking Functions*

A function need to be invoked or called to execute it in the browser. To invoke a function, specify the function name followed by parenthesis outside the function block.

A function can be defined and invoked even in an external JavaScript file. Also, a function can be called from another function in JavaScript. The function that invokes another function is called the **calling** function; whereas the function that is called is referred to as the **called** function.

Functions provide the benefit of code reusability by allowing the user to call a function multiple times.

Figure 17.3 shows invoking of function.

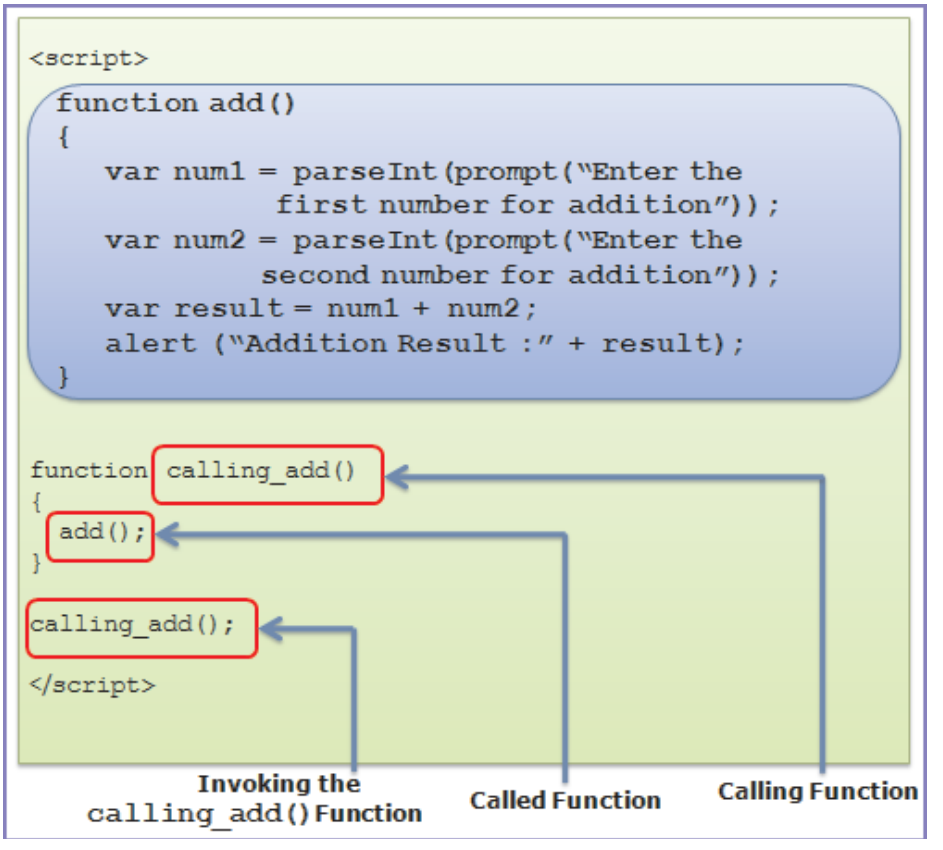



Figure 17.3: Invoking of Function

## *17.2.3 Parameterized Functions*

Parameterized functions refer to JavaScript functions that take parameters. These parameters hold values on which the function needs to perform operations. Parameterized functions can be created to accept values for performing operations.

Figure 17.4 shows the parameterized functions.

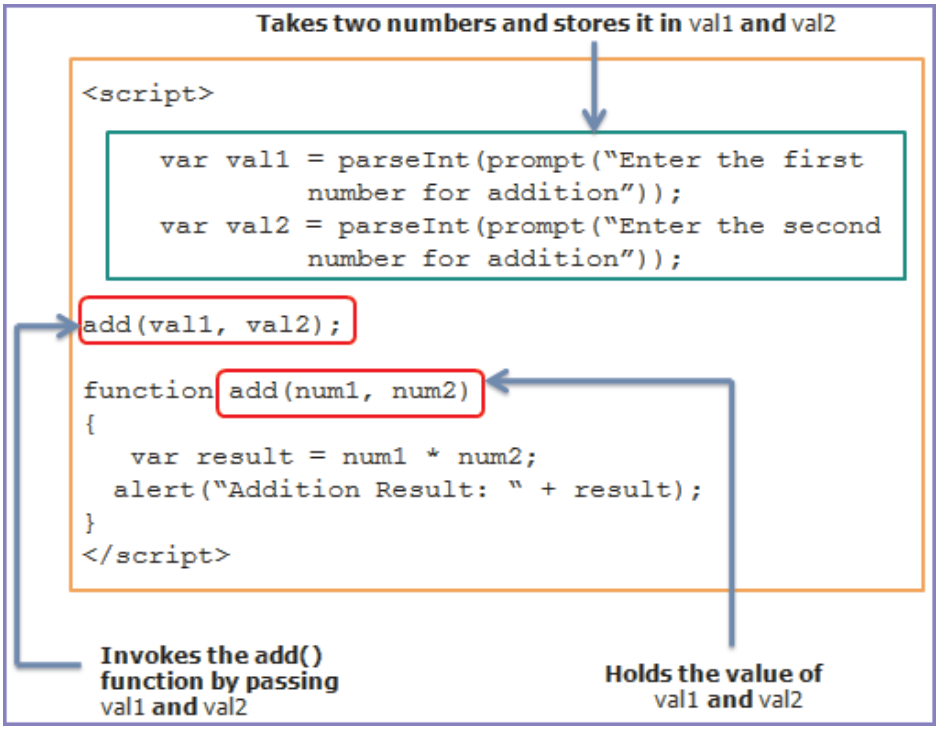



Figure 17.4: Parameterized Functions

As shown in figure 17.4, the **num1** and **num2** parameters will hold the values of **val1** and **val2** arguments to perform the operations. The **num1** and **num2** parameters are only accessible within the function, whereas the **val1** and **val2** variables are accessible anywhere within the `script` element.

The parameters of a function are variables that are declared in the function declaration. Here, **num1** and **num2** are the parameters of the function. Similarly, arguments are the values passed to the function. Here, **val1** and **val2** are the arguments whose values are passed to the parameters, **num1** and **num2**, while invoking the function.

Alternatively, one can use same variable names for arguments and parameters while creating and invoking functions. In either of the case, the variables will occupy different memory space.

## *17.2.4 Ways of Passing Arguments*

There are two ways of passing arguments to a function namely, pass by value and pass by reference. The description about these is as follows:

➔ **Passing by value** - Refers to passing variables as arguments to a function. In the pass by value method, the called function do not change the values of the parameters passed to it from the calling function.

This is because each parameter occupies different memory locations.

Figure 17.5 shows the pass by value method.

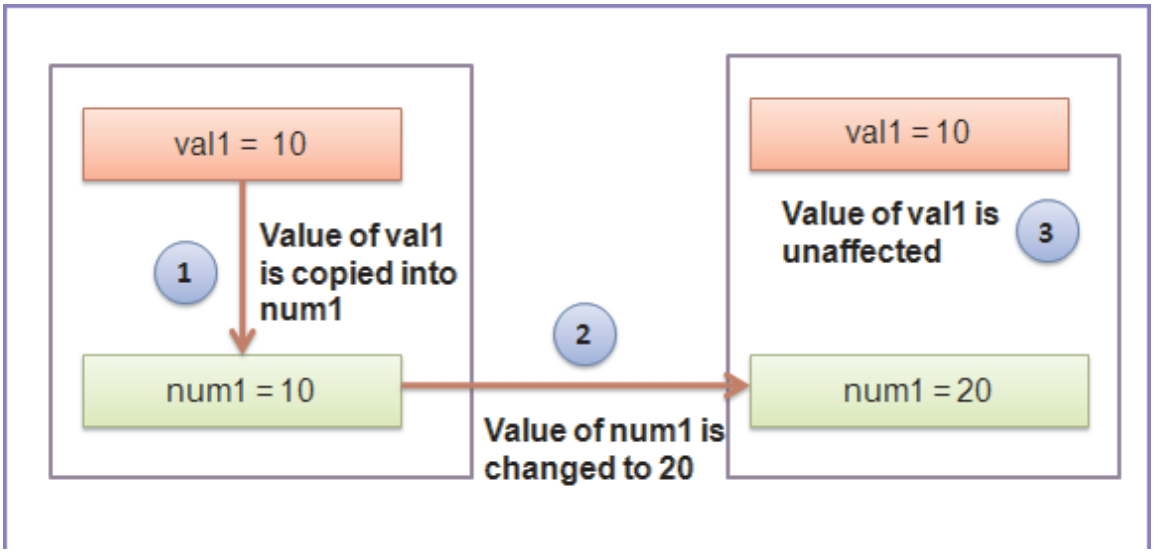



Figure 17.5: Pass By Value Method

➔ **Passing by reference** - Refers to passing objects as arguments to a function. In the pass by reference method, the called function modifies the value of parameters passed to it from the calling function. This change is reflected when the control passes back to the calling function.

Figure 17.6 shows the pass by reference method.

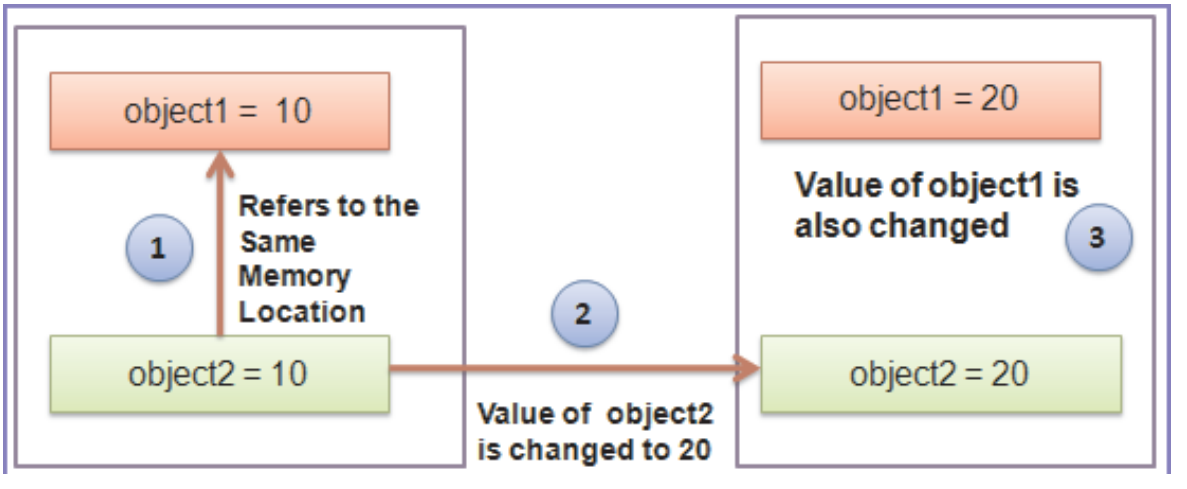



Figure 17.6: Pass By Reference Method

Code Snippet 1 demonstrates the code to pass `Array` object as a parameter to a function.

**Code Snippet 1**:

```
<script>
var names =new Array('James', 'Kevin', 'Brad');
function change_names(names)
{
  names[0]= 'Stuart';
}
function display_names()
{
 document.writeln('<H3> List of Student Names:</H3>');
 document.write('<UL>');
  for(vari=0; i<names.length; i++)
  {
     document.write('<LI>' + names[i]+ '</LI>');
  }
 document.write('</UL>');
 change_names(names);
 document.write('<H3> List of Changed Students Names:</H3>');
 document.write('<UL>');
 for(vari=0; i<names.length; i++)
 {
    document.write('<LI>' + names[i]+ '</LI>');
 }
document.write('</UL>');
}
display_names(names);
</script>
```

In the code, the function **`change_names(names)`** takes the **`names`** array as parameter. It changes the value at the 0th position in the array. The function is further invoked in the **`display_names()`** function. The **`display_names()`** function displays the values from the array before and after the value is changed at the 0th position in the array.

Figure 17.7 shows the passing of an array to a function.

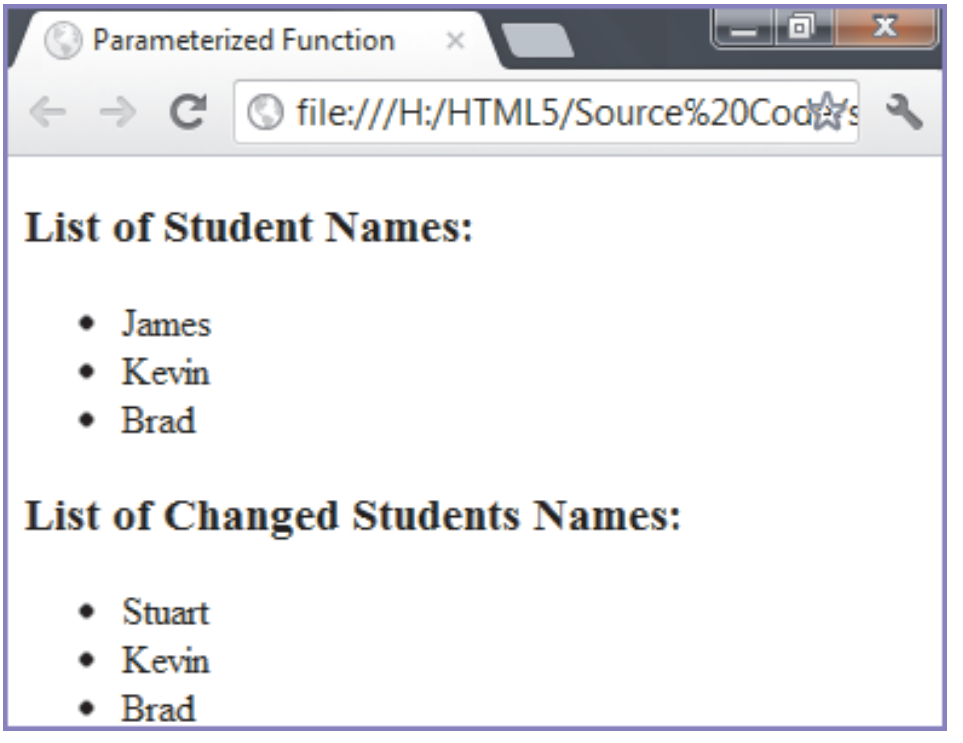



**Figure 17.7: Passing an Array Object to Function**

## *17.2.5 return Statement*

A function operates on its parameters that might lead to some output values. This output needs to be displayed to the user or it needs to be sent back to the calling function. JavaScript allows sending the result back to the calling function by using the `return` statement.

The return statement begins with `return` keyword followed by the variable or value, which needs to be returned to the calling function. The return statement can also be used to halt the execution of the function and to return the control to the calling function. This is required when a particular condition is false or when there are chances of unexpected results during the code execution.

Code Snippet 2 demonstrates the script that calculates the factorial of a number using a function and display the output to the user.

**Code Snippet 2**:

```
<script>
function factorial(num)
{
      if(num==0)
            return 0;
      else if(num==1)
            return 1;
     else
    {
           var result =num;
           while(num>1)
           {
              result = result *(num-1);
              num--;
           }
               return result;
    }
}
var num=prompt('Enter number:','');
var result = factorial(num);
alert('Factorial of '+num+ 'is ' + result + '.');
</script>
```

The code defines a function named **`factorial()`** which takes the **`num`** variable as the parameter. The execution of the script starts from the `prompt()` function, which takes the number from the user and stores it in the **`num`** variable. Next, the **`factorial()`** function is invoked by passing the **`num`** argument. If the user enters the value as 0 or 1, the function returns the value as 0 or 1 respectively. For any other number, the function calculates the factorial and returns the output value by using the return statement. The output is stored in the **`result`** variable, which is displayed to the user.

Figure 17.8 displays the factorial of a number.

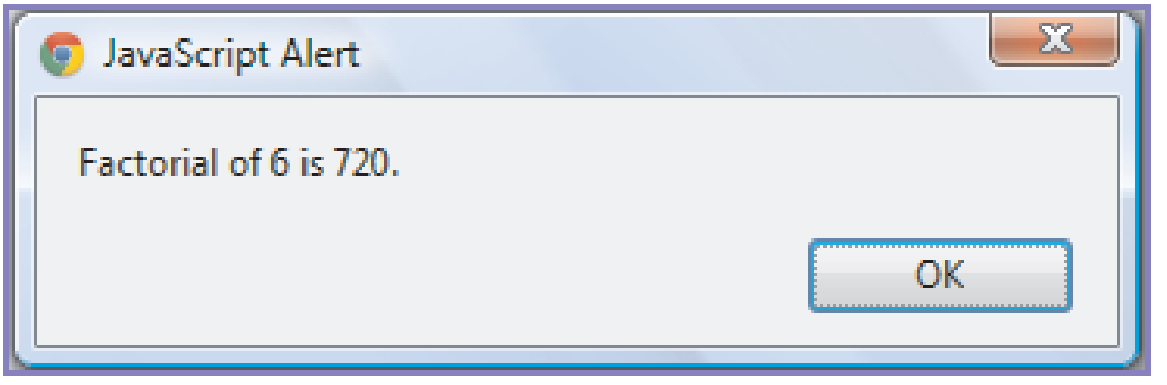



Figure 17.8: Factorial of Number

Similarly, the return statement can be used to return a collection of values stored in arrays. Figure 17.9 shows a **`display_list()`** function which declares and initializes an array named, **`languages`** to store the languages. The function returns an array whose values are then displayed to the user.

```
<script>

    function display_list()
    {
            var languages = new
            Array('English', 'Dutch',
                          'German');
            return languages;
    }

 document.write("<B> languages: </B>" +
                        display_list());
</script>
```

Figure 17.9: Returning Arrays

## *17.3 Objects*

Objects are entities with properties and methods and resemble to real life objects. Properties specify the characteristics or attributes of an object, while methods identify the behavior of an object. For example, consider a real life object namely, **`Car`**.

The attributes of the **`Car`** object can include color, car number, and model. The methods of the car could be `run()` that specifies the running behavior of the car. Similarly, in JavaScript, objects have their own properties and methods.

Figure 17.10 shows objects with their properties and methods.

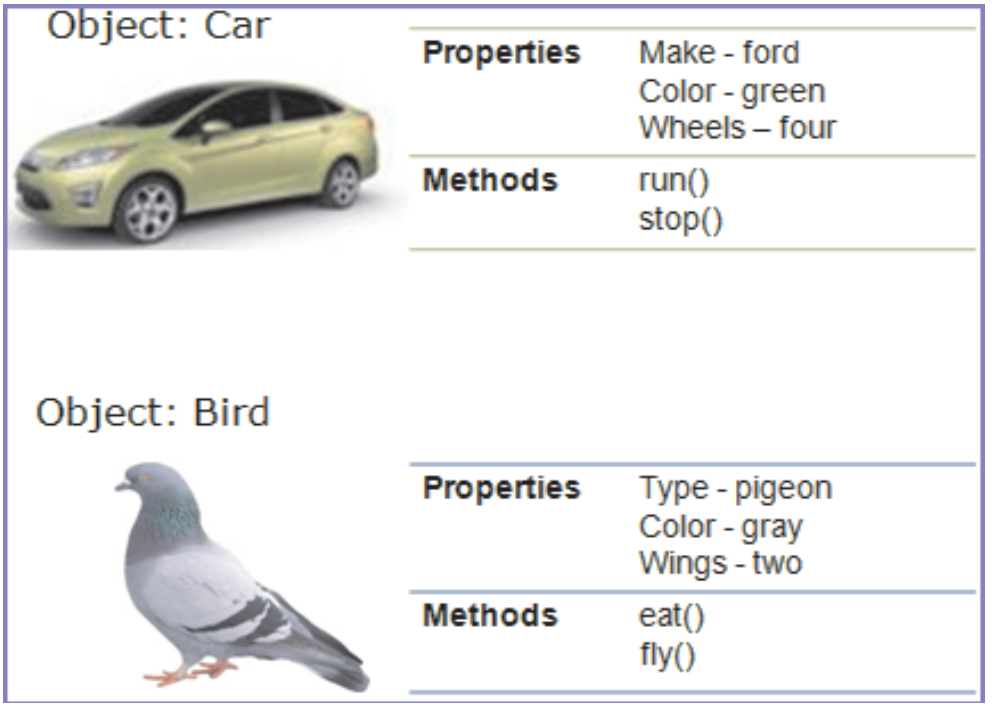



**Figure 17.10: Objects with Properties and Methods**

JavaScript provides built-in objects and allows creating user-defined objects. The description of the object is as follows:

- **Built-in Objects** - Are pre-defined objects which are already defined. Their properties and methods need to be called to fulfill a task. An example of a pre-defined object is the `Array` object.

- **Custom Objects** - Are user-defined objects, which the developer explicitly creates in the script and defines their properties and methods. For example, to store the doctor details, such as name, age, hospital name, and so on, an object named **`doctor`** can be created.

## *17.3.1 Creating Custom Objects*

The `Object` object is the parent object from which all JavaScript objects are derived. The custom objects can be derived from this object by using the `new` keyword.

There are two main methods to create a custom object. In the first method, an object can be created by using the built-in `Object` object, which is also known as the **direct** method.

In the second method, an object can be created by defining a template and initializing it with the `new` keyword.

The syntax to create the object using these methods are as follows:

## ➔ Direct Method

The syntax to create a custom object using the `Object` object is as follows:

**Syntax**:

```
var object_name = new Object();
```

where,

`object_name`: Is the name of the object.

`new`: Is the keyword that allocates memory to the custom object. This is known as instantiation of an object.

`Object`: Is the built-in JavaScript object that allows creating custom objects.

## ➔ Template Method

An object's template refers to a structure that specifies the custom properties and methods of an object. There are two steps in creating an object by using this method. First, the object type is declared using a constructor function. Second, you specify the object of the declared object type by using the `new` keyword.

JavaScript allows creating a reusable template without having to redefine properties and methods repeatedly to fulfill a particular object's requirements. This template is known as the constructor function.

A constructor function is a reusable block that specifies the type of object, its properties, and methods. It might or might not take any parameters. After creating the constructor function, you specify an object of the declared object type using the `new` keyword. The `new` keyword allocates memory for the object and invokes a constructor function.

The syntax to create a constructor function is as follows:

**Syntax**:

```
function object_type(list of parameters)
{
// Body specifying properties and methods
}
```

where,

`object_type`: Indicates the object type.

`list of parameters`: Is optional and specifies the parameters to be passed to a function separated by commas.

The syntax to create the object using the `new` keyword is as follows:

**Syntax**:

```
object_name = new object_type(optional list of arguments);
```

where,

`object_name`: Is the name of the object.

Code Snippet 3 shows the creation of objects using direct method and template method.

**Code Snippet 3**:

```
<script>
//create an object using direct method
var doctor_details=new Object();
//create an object using new keyword
studOne = new student_info ('James', '23', 'New Jersey');
</script>
```

In the code, **`doctor_details`** object is created using the `Object` object. After creating the object, properties and methods can be specified. Similarly, **`student_info`** object is created using `new` keyword. The values 'James', '23', and 'New Jersey' are the properties of the **`student_info`** which are initialized by constructor function during creation.

## *17.3.2 Creating Properties for Custom Objects*

Properties specify the characteristics of an object. They can be specified for objects created through `Object` or template method.

To create and access a property of an object created using `Object` object, specify the object name followed by a period and the property name.

Code Snippet 4 demonstrates the script that creates the `student_details` object and adds properties namely, `first_name`, `last_name`, and `age` along with their values.

**Code Snippet 4**:

```
<script>
var student_details=new Object();
student_details.first_name= 'John';
student_details.last_name= 'Fernando';
student_details.age= '15';
alert ('Student\'s name: ' +student_details.first_name+ ' ' +
      student_ details.last_name);
</script>
```

The code specifies three properties of the **`student_details`** object namely, **`first_name`**, **`last_name`**, and **`age`** along with their values. The values of these properties are displayed in the browser using the `write()` method.

Figure 17.11 shows the output of the **`student_details`** object.

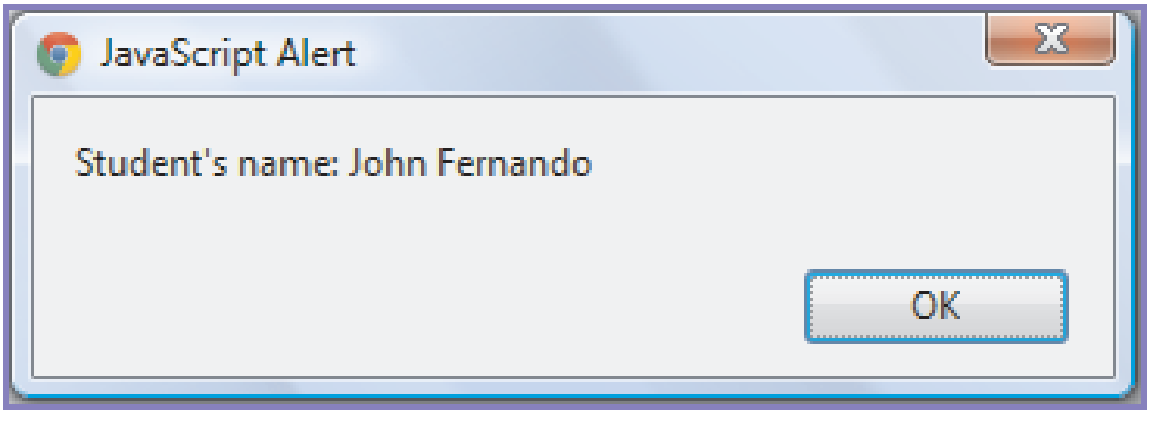



**Figure 17.11: Output – student_details Object**

Similarly, if the template method is implemented to create a custom object, then a constructor function is used to declare properties for an object.

Code Snippet 5 creates the **`employee_info`** object and adds properties in the constructor function.

**Code Snippet 5**:

```
<script>
 // To define the object type
   function employee_info(name, age, experience)
   {
       this.name = name;
       this.age= age;
       this.experience= experience;
   }
// Creates an object using new keyword
   empMary = new employee_info('Mary', '34', '5 years');
   alert ("Name: "+ empMary.name + '\n' +"Age: "+empMary.age+ '\n'
              +"Experience: "+empMary.experience);
</script>
```

The code specifies three properties namely, **`name`**, **`age`**, and **`experience`** along with their values in the constructor function. The object named **`empMary`** is created, which passes the values as the arguments. This invokes the constructor function and initializes the properties to their values. The `this` keyword is a reference to the current object whose properties are being initialized. It is used in the constructor to resolve conflict between the property and the parameter, both of which have the same name. The `this` keyword marks the distinction between the two, while assigning the value to the properties of an object.

Figure 17.12 displays the output of the **`employee_info`** object.

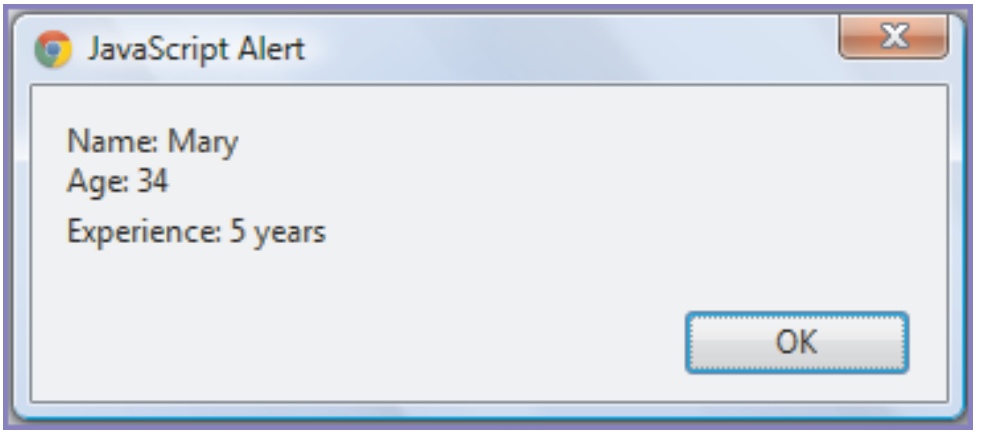



**Figure 17.12: Output of the employee_info Object**

## *17.3.3 Creating Methods for Custom Objects*

Methods are similar to JavaScript functions, but there is a slight difference between them. A method is always associated with an object and is executed by referring to that object. On the other hand, a function is not associated with any object and is executed independently. Similar to functions, the custom methods can also take parameters.

One or more methods can be specified after creating an object using the `Object` object or while creating the template. To invoke a method, they must be specified with the object name followed by a period, method name, and parenthesis with parameters, if any.

Code Snippet 6 demonstrates the code that defines a custom method to calculate the area of a square.

**Code Snippet 6**:

```
<script>
   var square =new Object();
    square.length=parseInt("5");
   square.cal_area=function()
    {
       var area =(parseInt(square.length)*parseInt("4"));
        return area;
    }
   alert ("Area: "+square.cal_area());
</script>
```

The code defines a custom object named **`square`** whose `length` property is set to a numeric value 5. It also defines a custom function named **`cal_area()`**, which calculates the area of the square and returns the result.

Figure 17.13 displays the output of the area of a square.

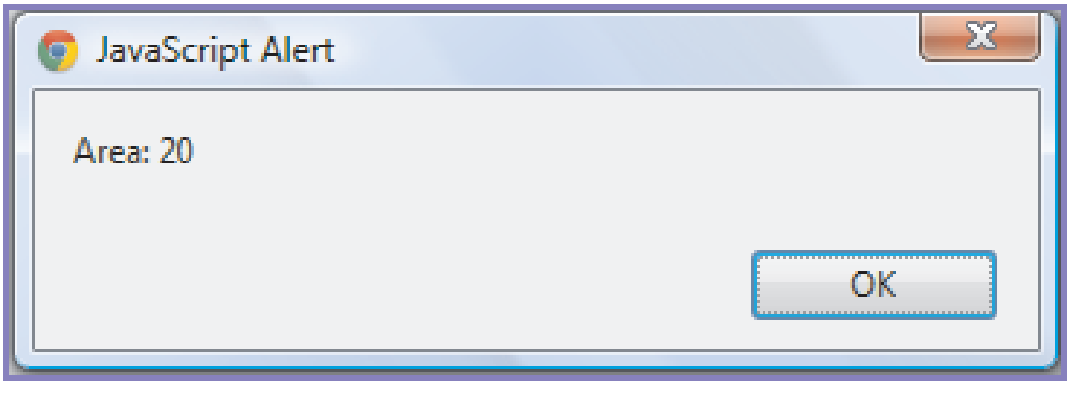



**Figure 17.13: Output of the Area of Square**

Similarly, methods can be specified while using template method for creating a custom object. The implementation of custom methods is same for creating and invoking methods as specified for the direct method.

However, in the template method, an ordinary function is created to implement the functionality. Then, this function is assigned to the custom method. Such functions are known as **method functions**. To invoke the function, you specify the object name followed by a period and the method name.

Code Snippet 7 demonstrates the script that creates a custom method and associates it with the method function.

**Code Snippet 7**:

```
<script>
//Define a Method Function
  functioncal_diameter()
   {
        var diameter =this.radius*2;
        return diameter;
   }
//Define a Constructor Function
   function circle(radius)
   {
       this.radius= radius;
       this.get_diameter=cal_diameter;
   }
//Create the object
circleObj = new circle('10');
alert ('Diameter: '+circleObj.get_diameter());
</script>
```

The code creates an object, **`circleObj`** and invokes the custom method **`get_diameter()`**, which in turn invokes the function, **`cal_diameter()`**.

## *17.4 Built-in Objects*

The object model of JavaScript language forms the foundation of the language. These objects help to provide custom functionalities in the script.

JavaScript treats the primitive data types as objects and provide equivalent object for each of them. For example, if a variable contains a string of characters, then it is treated as `String` object by JavaScript. Similarly, if a variable contains the value `true` or `false`, it is treated as `Boolean` object.

JavaScript objects are categorized as built-in objects, browser objects, and HTML objects.

The built-in objects are static objects which can be used to extend the functionality in the script. Some of these objects are: `String`, `Math`, and `Date`. The browser objects, such as `window`, `history`, and `navigator` are used to work with the browser window, whereas the HTML objects, such as `form`, `anchor`, and so on are used to access elements on a Web page.

### *17.4.1 String Object*

Strings in JavaScript are a set of characters that are surrounded by single or double quotes. These characters can include alphabets, numbers, spaces, and symbols: `%`, `@`, `&`, and so on. The built-in `String` object allows you to perform different text operations on them. Some of the examples of these operations include: searching for a specific character occurrence, retrieving a substring, merging two set of characters, and so on.

The `String` object is instantiated with the `new` keyword, which invokes the predefined constructor function of the `String` object.

The syntax to initialize the `String` object is as follows:

**Syntax**:

```
varobject_name = new String("Set of characters") ;
```

where,

`object_name`: Is the instance of the `String` object

The `String` object provides different properties and methods to manipulate strings.

Table 17.1 lists the properties of the `String` object.

| Property | Description |
|---|---|
| `length` | Retrieves the number of characters in a string. |
| `prototype` | Adds user-defined properties and methods to the String instance. |

**Table 17.1: Properties of the String Object**

Table 17.2 lists the methods of the `String` object.

| Method | Description |
|---|---|
| `charAt()` | Retrieves a character from a particular position within a string. |
| `concat()` | Merges characters from one string with the characters from another string and retrieves a single new string. |
| `indexOf()` | Retrieves the position at which the specified string value first occurred in the string. |
| `lastIndexOf()` | Retrieves the position at which the specified string value last occurred in the string. |
| `replace()` | Matches a regular expression with the string and replaces it with a new string. |
| `search()` | Searches for a match where the string is in the same format as specified by a regular expression. |
| `split()` | Divides the string into substrings and defines an array of these substrings. |
| `substring()` | Retrieves a part of a string between the specified positions of a string. |
| `toLowerCase()` | Specifies the lowercase display of the string. |
| `toUpperCase()` | Specifies the uppercase display of the string. |

**Table 17.2: Methods of the String Object**

Code Snippet 8 demonstrates the script that creates a `String` object and test various methods on it.

**Code Snippet 8**:

```
<script>
varfull_name=new String('David James Taylor');
document.write('Number of Characters are: '+full_name.length+ '<BR/>');
document.write('Character at Position 6 is: '+full_name.charAt(6)+ '<BR/>');
document.write('Student\'s Name and their Father\'s name are: '+full_name.
split(' ',2)+ '<BR/>');
document.write('Student\'s Full Name is: '+full_name.toUpperCase());
</script>
```

Figure 17.14 displays the output of string manipulation.

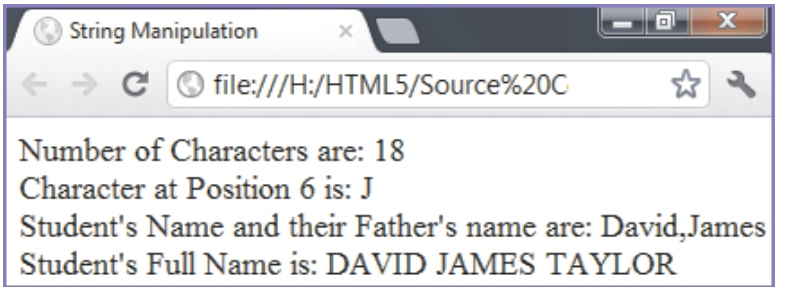



**Figure 17.14: Output – String Manipulation**

## *17.4.2 Math Object*

The `Math` object allows the user to perform mathematical operations on numeric values. The `Math` object is a pre-defined object that provides static properties and methods to perform mathematical operations. The properties and methods are declared as static, thus they can be invoked directly with the object name. In other words, no object instantiation is required for the `Math` object.

The syntax to access the properties of the `Math` object is as follows:

**Syntax**:

```
var variable_name =Math.PropertyName;
```

where,

`PropertyName`: Is the name of the property

Similarly, the syntax to invoke the methods of the `Math` object is as follows:

**Syntax**:

```
var variable_name = Math.MethodName(optional list of parameters);
```

Code Snippet 9 demonstrates the script that calculates area of a circle using the `Math` object.

**Code Snippet 9**:

```
<script>
var full_name=new String('David James Taylor');
document.write('Number of Characters are: '+full_name.length+'<BR/>');
var area =Math.floor(tempArea);
return area;
}
alert('Area of circle is: '+area_circle(5));
</script>
```

In the code, `Math.PI` is a property that stores the value 3.142 into the variable, **`pi`**. The `Math.pow(radius, 2)` method calculates the radius raised to the power 2. Similarly, `Math.floor(tempArea)` method rounds the resultant value to a number less than or equal to the resultant value.

Figure 17.15 shows the area of circle with the radius5.

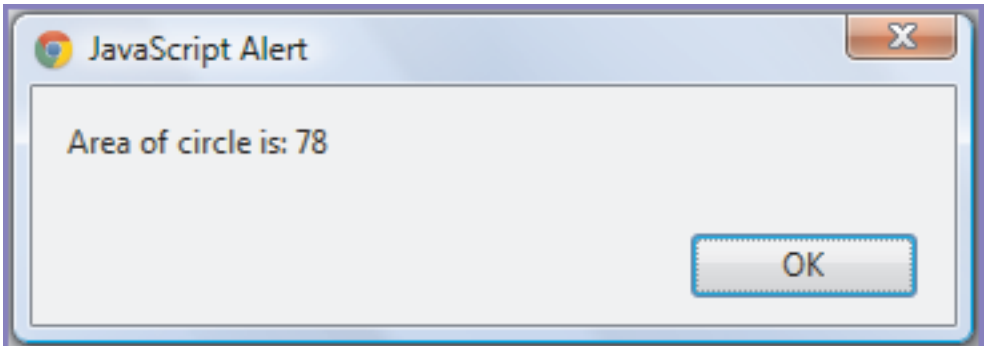



**Figure 17.15: Area of Circle with the radius5**

## *17.4.3 Date Object*

The `Date` object allows you to define and manipulate the date and time values programmatically. It supports both the Universal Time Coordinated (UTC) and Greenwich Mean Time (GMT) conventions. GMT is the standard time zone that includes Greenwich and divides the globe into 24 zones, each with a difference of an hour in time. UTC splits time into days, hours, minutes, and seconds and approximates GMT.

The `Date` object calculates dates in milliseconds from 01 January, 1970. The date and time can be specified by creating an instance of the `Date` object.

The various syntax to instantiate the `Date` object are as follows:

**Syntax**:

a. To instantiate the `Date` object with the current date and time as that of the local machine.

```
var object_name = new Date();
```

where,

`object_name`: Is the instance of the `Date` object.

b. To instantiate the `Date` object by providing passed milliseconds, since 01 January, 1970 as the parameter.

```
var object_name = new Date(milliseconds);
```

c. To instantiate the `Date` object by passing date values and optional time values as the parameters.

```
var object_name = new Date(year, month, day, hour, minutes, seconds,
milliseconds);
```

Here, the `Date` object refers to the month numbers from 0 to 11 and treats the first month as `00`. Therefore, one needs to specify `02` as the month value for the March month. If the optional arguments are not supplied, they are set to `0`.

d. To instantiate the `Date` object by passing date values and optional time values in quotes as the parameter.

```
var object_name = new Date("dateString");
```

Table 17.3 lists the methods of the `Date` object.

| Method | Description |
|---|---|
| `getDate()` | Retrieves a numeric value between 1 and 31 which indicates the day of the month. |
| `getDay()` | Retrieves a numeric value between 0 and 6 which indicates the day of the week. |
| `getTime()` | Retrieves a numeric value which indicates the time passed in milliseconds since midnight 01/01/1970. |
| `getFullYear()` | Retrieves a four digit numeric value which indicates the year in the given date. |

**Table 17.3: Methods of the Date Object**

Code Snippet 10 demonstrates the script that displays the current date in the `mm/dd/yyyy` format.

**Code Snippet 10**:

```
<<script>
function display_date()
{
 var today=new Date();
 var date=today.getDate();
 var month=today.getMonth();
 month++;
 var year=today.getFullYear();
 alert ('Today\'s date is: '+month+'/'+date+'/'+year);
}
display_date();
</script>
```

Figure 17.16 displays the current date.

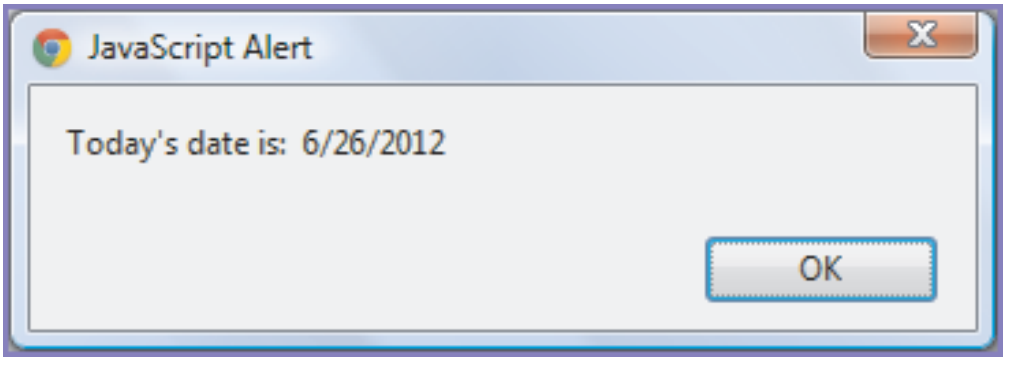



**Figure 17.16: Current Date**

## *17.4.4 with Statement*

The `with` statement allows to remove the object reference for each JavaScript statement. This is done by referring to the common object only once for a set of statements.

The `with` statement starts with the `with` keyword followed by the open and close brackets, which holds the statements that refer to a common object. This increases the readability of the code and also reduces time required in writing each object reference in every related statement.

The syntax to declare the `with` statement is as follows:

**Syntax**:

```
with(object_name)
{
   //Statements
}
```

where,

`object_name`: Is the name of a common object for the set of statements

Figure 17.17 displays `with` statement.

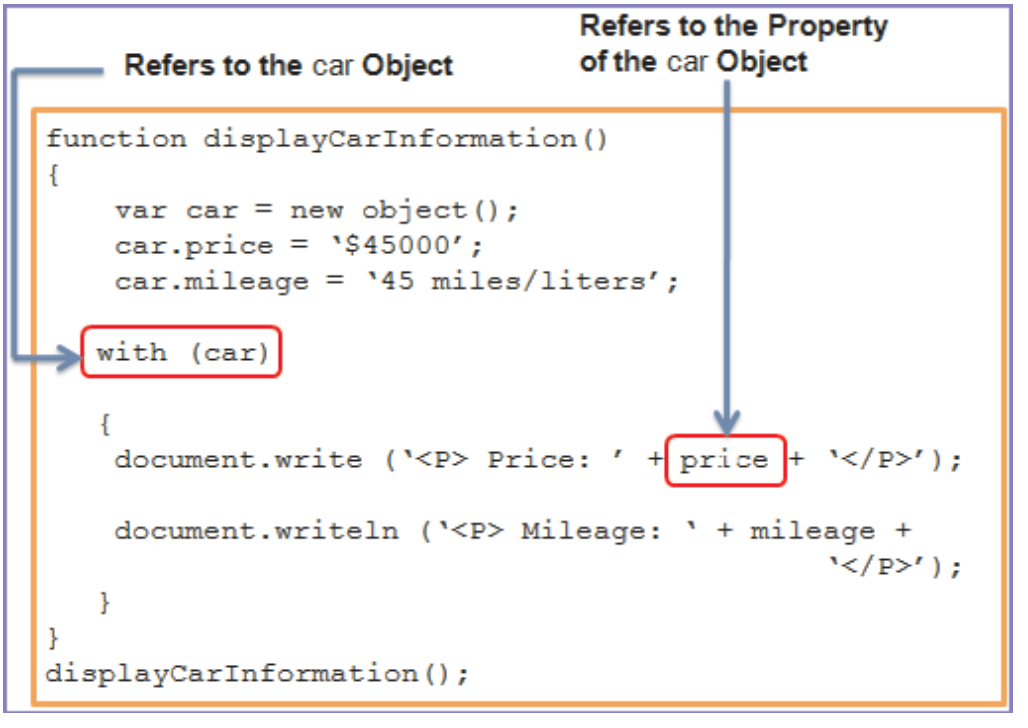



Figure 17.17: with Statement

## *17.5 Browser Objects*

Apart from built-in objects, JavaScript also provides objects to access and manipulate various aspects of a Web browser. These objects are called as browser objects. They exist on all pages displayed in the browser and correspond to elements on a page.

For example, browser objects allow accessing various characteristics of the browser, such as browser window itself, browser history, changing current URL, and moving backward and forward in the browser.

Figure 17.18 shows the hierarchy of browser objects.

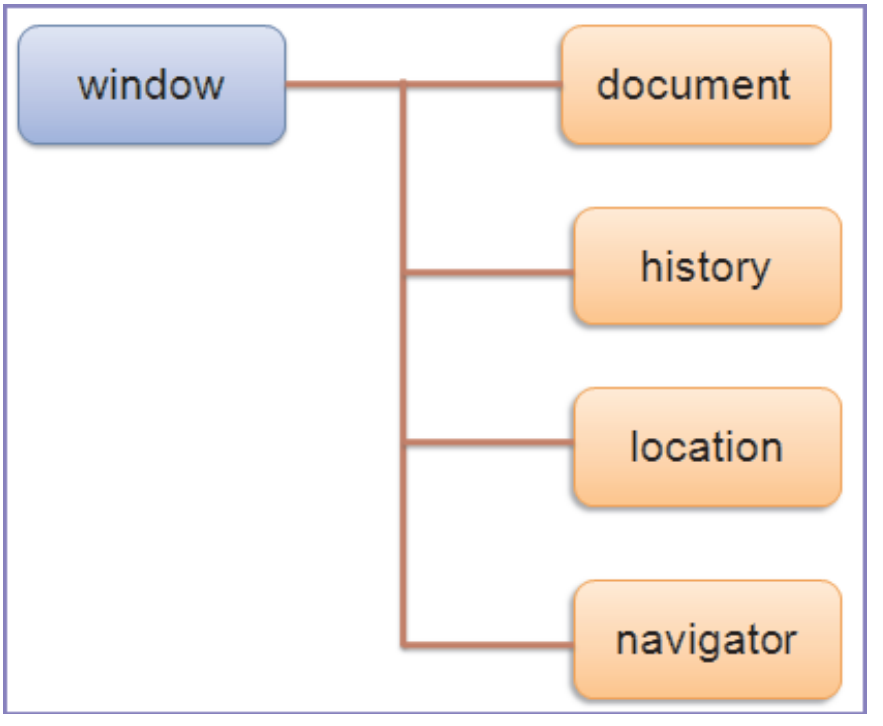



Figure 17.18: Hierarchy of the Browser Objects

The brief description for some of these objects is as follows:

➔ **window Object**

The `window` object is the top level object in JavaScript hierarchy. This means that all the objects in the hierarchy are descendants of the `window` object. The `window` object represents a browser window. It contains browser information, such as the look and feel of the browser, its version, and so on.

The `window` object provides properties that allows setting a default text for the status bar, name of the browser window, retrieve history, and so on.

Table 17.4 lists some of the commonly used properties of the `window` object.

| Property | Description |
|---|---|
| `defaultStatus` | Specifies or retrieves the default text to be displayed in the status bar of the browser window. |
| `document` | Represents an HTML document that contains different elements. |
| `history` | Contains history of the visited Uniform Resource Locators (URLs). |
| `location` | Contains the content of the specified URL. |

Table 17.4: Properties of the window Object

The `window` object provides methods that allows displaying error messages, confirmation boxes, and so on.

Table 17.5 lists the methods of the `window` object.

| Method | Description |
|---|---|
| `alert()` | Displays an alert box that states the message and an OK button. |
| `confirm()` | Prompts a dialog box that displays a message with the OK and Cancel buttons. |
| `createPopup()` | Creates a pop-up window. |
| `focus()` | Focuses the current window. |
| `open()` | Opens the specified file in a new browser window. |
| `prompt()` | Prompts a dialog box that accepts input from the user. |

**Table 17.5: Methods of the window Object**

Code Snippet 11 demonstrates the methods of the `window` object.

**Code Snippet 11**:

```
<!DOCTYPE html>
<head>
<title> window Object </title>

<script>
function new_window()
{
   if(confirm('Do you want to open a new page?'))
   {
        window.open('http://www.aptech-education.com/pages/
        about-us/about-aptech.html', '_parent');
   }
   else
   {
         window.alert('In the Current Window');
   }
}
</script>

</head>
<body>

<input type="button" value="Click to move to the next

                    page"onClick="new_window();"/>
</body>

</html>
```

The code invokes the function **`new_window()`** on a button click. This function asks the user for opening the new page. If user clicks OK, then `about-aptech.html` page is opened in the current window. Otherwise, displays a message to the user.

### ➔ history Object

The `history` object is a part of the `window` object. It contains a set of URLs visited by a user in a browser window. The `history` object is an array that allows referring to a particular URL by specifying its index number in the array. The `length` property allows you to determine the number of URLs in the history list.

Table 17.6 lists the methods of the `history` object.

| Method | Description |
|---|---|
| `back()` | Retrieves and displays the previous URL from the history list. |
| `forward()` | Retrieves and displays the next URL from the history list. |
| `go()` | Displays the specified URL. It accepts a parameter, which can either be a string or a number to go to specific page. |

**Table 17.6: Methods of the history Object**

### ➔ navigator Object

The `navigator` object contains information about a browser used by the client. It allows the user to retrieve browser specific information, such as name, version number, and language.

Table 17.7 lists the properties of the `navigator` object.

| Property | Description |
|---|---|
| `appName` | Retrieves the name of the browser. |
| `appVersion` | Retrieves the version number and platform of the browser. |
| `browserLanguage` | Retrieves the language of the browser. |
| `cookieEnabled` | Determines whether the cookies are enabled in the browser. |
| `platform` | Retrieves the machine type such as Win32, of the client browser. |

**Table 17.7: Properties of the navigator Object**

Code Snippet 12 demonstrates the use of `navigator` object to retrieve information about the browser.

**Code Snippet 12**:

```
<!DOCTYPE html>
<head>
<title> navigator Object </title>
<script>
 function display_browser()
 {
   document.write('Browser name: '+navigator.appName+ '<BR/>');
   document.write('Browser version: '+navigator.appVersion+
                                                '<BR/>');
  document.write('Browser language: '+navigator.
                                 browserLanguage+ '<BR/>');
  document.write('Platform: '+navigator.platform+ '<BR/>');
  if(navigator.cookieEnabled)
  {
        document.write('Cookie is enabled in the browser.');
  }
}
</script>
</head>
<body>
<input type="button" value="Browser Information"
                            onclick="display_browser()"/>
</body>
</html>
```

Figure 17.19 displays the output of the `navigator` object for Opera browser.

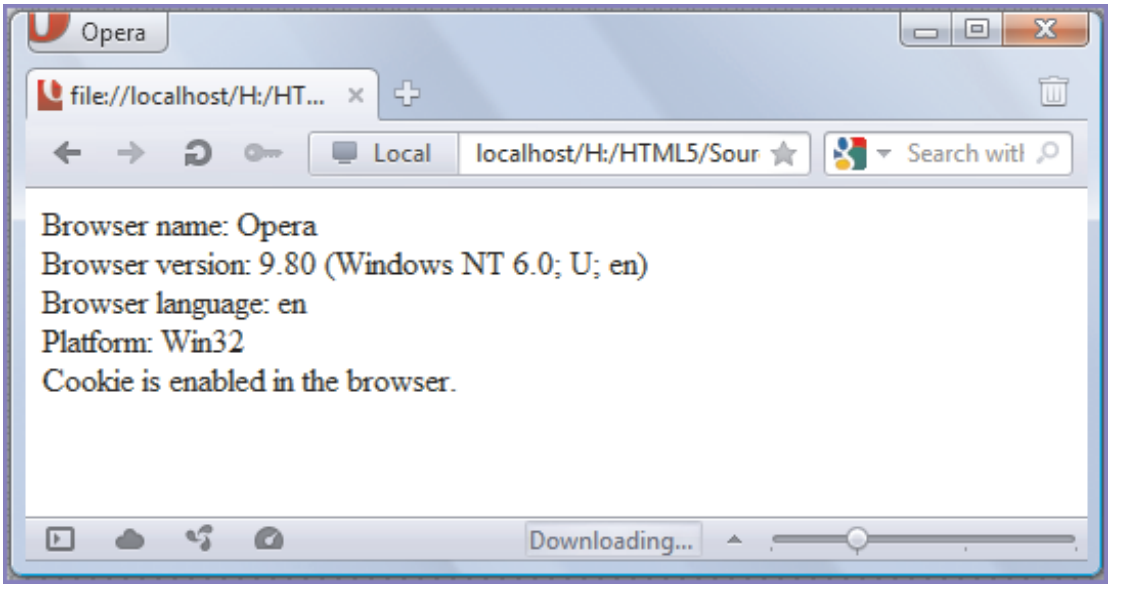



Figure 17.19: Output of the Navigator Object for Opera Browser

### ➔ location Object

The `location` object allows to access complete information of the URL loaded in the browser window. It is a part of the `Window` object. A single URL is composed of different portions, such as the host name, port number, and so on which can be accessed through the `location` object.

Table 17.8 lists the properties and methods of the `location` object.

| Property/Method | Description |
|---|---|
| `host` | Retrieves host name and port number of the URL. |
| `href` | Specifies or retrieves the entire URL. |
| `pathname` | Specifies or retrieves the path name of the URL. |
| `protocol` | Specifies or retrieves the protocol of the URL. |
| `assign()` | Loads a new document with the specified URL. |
| `reload()` | Reloads the current document by again sending the request to the server. |
| `replace()` | Overwrites the URL history for the current document with the new document. |

Table 17.8: Properties and Methods of the location Object

Code Snippet 13 demonstrates the use of `location` object to retrieve the different portions of the specified URL.

**Code Snippet 13**:

```
<!DOCTYPE html>
<head>
<title> location Object </title>

<script>
function display_URLInfo()
{
  alert('Protocol name: '+location.protocol+ '\n' + 'Path name: '
                                  +location.pathname);
}
</script>

</head>
<body>
<input type="button" value="View URL Information"
                          onclick="display_URLInfo()"/>
</body>
</html>
```

Figure 17.20 shows the details of the specified URL.

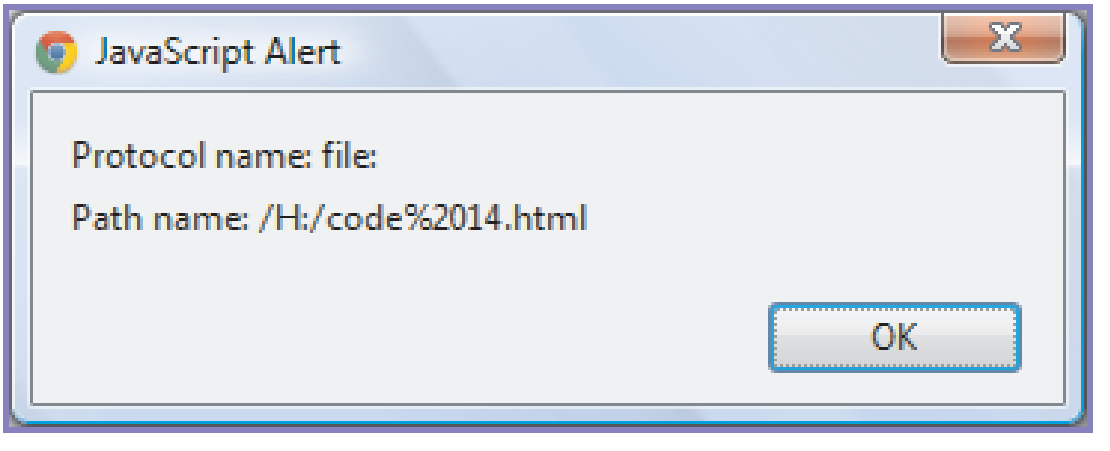



Figure 17.20: Details of Specified URL

## *17.6 Document Object Model (DOM)*

A Web page contains various elements, such as buttons, text boxes, check boxes, and so on. These elements exist in a hierarchy and overall represent an HTML document.

JavaScript allows the user to access HTML elements and also change the existing structure of an HTML document. This can be done by using DOM specification. The DOM is an API that defines the object structure for accessing and manipulating HTML elements. It is used with JavaScript to add, modify, or delete elements and contents on a Web page.

DOM specifications are laid by W3C and are implemented by all the browsers to overcome incompatibility issues. W3C DOM specifications are divided into levels. The level 1 specification of DOM was first defined in 1998. The current DOM specification is level 3.

The DOM reads all the elements contained in an HTML page. It treats the HTML elements as nodes. According to DOM specification, the entire HTML document represents a document node. This document node consists of element nodes, attribute nodes, and text nodes. Thus, the document node is the highest level node and text nodes are the lowest ones. Every node in the node hierarchy has a parent node, which consists of multiple child nodes. For example, `<head>` and `<body>` are the child nodes of `<html>`. All these nodes together form up a node tree and are related to each other.

Code Snippet 14 shows an HTML document.

**Code Snippet 14**:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome</title>
</head>

<body>
    <h1> Introduction </h1>
    <ahref=”#”>Click Here</a>
</body>

</html>
```

Figure 17.21 shows the DOM structure of HTML document.

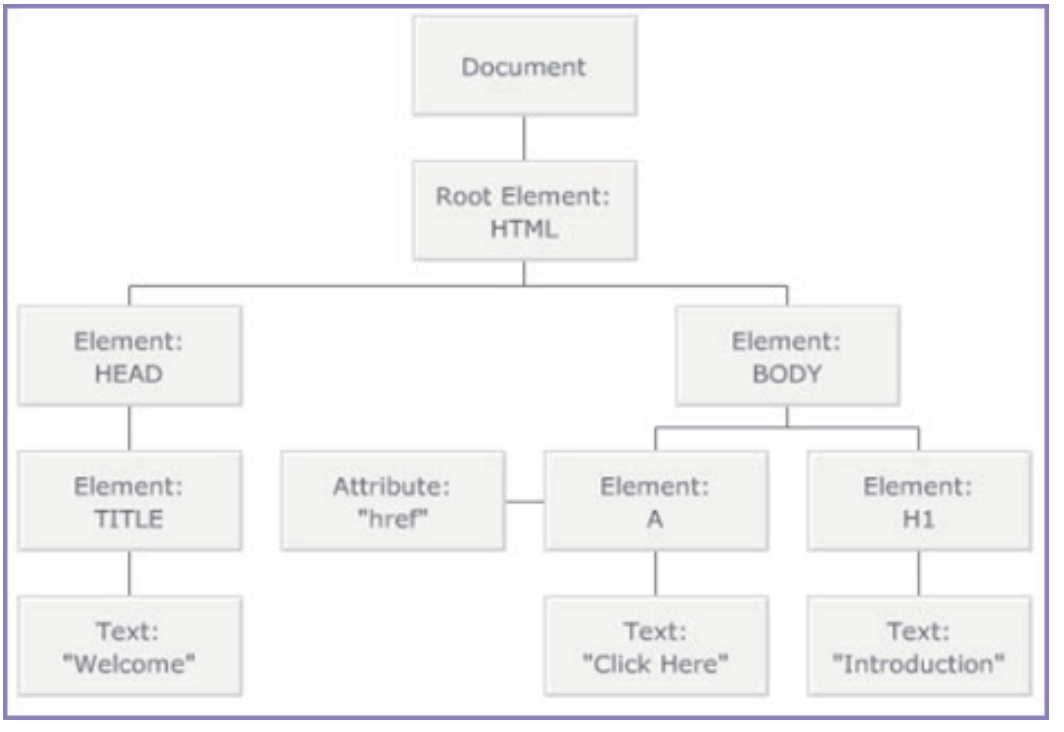



**Figure 17.21: DOM Structure**

All the nodes present in the node hierarchy contain certain properties. These properties provide information about the node. The different node properties are as follows:

- **nodeName** - Represents the name of the node. It contains the tag name of the HTML element in upper case.
- **nodeValue** - Represents the text contained within the node. This property is only available for attribute nodes and not for document and element nodes.
- **nodeType** - Represents the type of the node. For example, the document node, element node, and so on.

The HTML DOM provides standard objects for HTML documents. Some of these objects are as follows:

- Document object
- Form object
- Link object
- Table object

The brief description for some of these objects is as follows:

➔ **Document Object**

The HTML DOM provides a `document` object which is used within JavaScript to access all HTML elements presented on the page. The `document` object is one of the basic JavaScript object. It represents the entire HTML document. It provides access to other elements, such as links, anchors, and so on.

Each HTML page has only one `document` object. This object is created when the `BODY` element is loaded on the Web page. The document object is also the part of the `window` object and is accessed as `window.document`.

The `document` object provides properties that allow the user to specify or retrieve the information about the elements and its content.

Table 17.9 lists the properties of the `document` object.

| Property | Description |
|---|---|
| `body` | Provides access to the BODY element. |
| `title` | Specifies or retrieves the title of the document. |
| `anchors` | Retrieves the collection containing all the anchors contained in a document. |
| `forms` | Retrieves the collection containing all the forms contained in a document. |
| `images` | Retrieves the collection containing all the images contained in a document. |
| `links` | Retrieves the collection containing all the links contained in a document. |

**Table 17.9: Properties of the document Object**

The `document` object provides methods that allow retrieving the HTML elements using the id, name, and tag name.

Table 17.10 lists the methods of the `document` object.

| Property | Description |
|---|---|
| `close()` | Closes a data stream and displays the data collected using the `open()` method. |
| `getElementById()` | Retrieves a collection of HTML elements by using the specified ID. |
| `getElementsByName()` | Retrieves a collection of HTML elements by using the specified name. |
| `getElementsByTagName()` | Retrieves a collection of HTML elements with the specified tag name. |

| Property | Description |
|---|---|
| `open()` | Opens a stream to accept the output from `write()` or `writeln()` methods. |
| `write()` | Writes the text or HTML expression to a document. |

Table 17.10: Methods of the document Object

Code Snippet 15 demonstrates the use of the `document` object to change the image on the click of a button.

**Code Snippet 15**:

```
<!DOCTYPE html>
<head>
<title> Document Object </title>
<script>
 function change_image()
 {
    varimgText=document.getElementById('myImg').alt;
    if(imgText=="ford")
   {
     document.getElementById('myImg').src="ferrari.jpg";
     document.getElementById('myImg').alt ="ferrari";
     document.getElementById('mytext').value ="Ferrari Car";
   }
   else
  {
     document.getElementById('myImg').src="ford.jpg";
     document.getElementById('myImg').alt ="ford";
     document.getElementById('mytext').value ="Ford Car";
  }
 }
</script>
</head>
```

```
<body>

<imgid="myImg"  src="ford.jpg"width="300"
height="300"alt="ford" /><br/>

 Model: <input type="text" id="mytext"

   value="Ford Car"    readonly="readonly"/><br/><br/>

<input type="button" value="Change Image" onclick="change_
image()"/>

</body>

</html>
```

In the code, image and text elements on the page are accessed using `document.getElementById()` method. The method retrieves the elements based on the specified ids and sets new values for their properties. This is all done at runtime, and not through markup. The use of `document.getElementById()` enables to access the elements within JavaScript function and change them dynamically.

Figures 17.22 and 17.23 show the initial image and changed image respectively on clicking `Change Image` button.

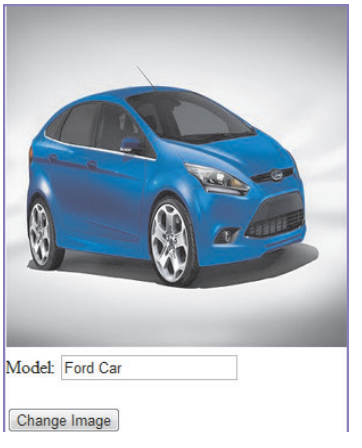



**Figure 17.22: Initial Image**

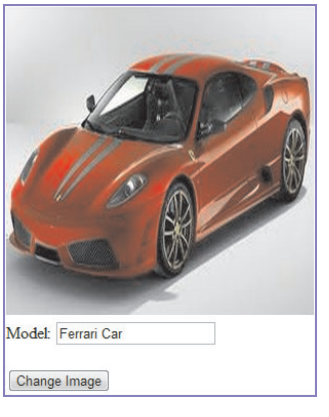



**Figure 17.23: Changed Image**

### ➔ Form Object

Form accepts input from the user and sends data for validation. JavaScript allows you to process and validate the form data. A single HTML document can contain multiple forms. The DOM specification provides a `form` object that represents an HTML form. A `form` object is created for each `<form>` tag in an HTML document.

Code Snippet 16 demonstrates the use of the `form` object that counts number of controls in a form.

**Code Snippet 16**:

```
<!DOCTYPE html>
<head>
<title> Form Object </title>
<script>
function display_length()
{
    var count =document.getElementById("form1").length;
    alert('Number of controls on the form: ' + count);
}
</script>
</head>
<body>
<form id="form1" action="welcome.php">
 First name: <input type="text" name="firstname" value="John"
/><br />
 Last name: <input type="text" name="lastname" value="Smith"
/><br />
 Age : <input type="text" name="age" value="40" /><br />
<input type="button" value="Controls"onClick="display_
length()"/>
</form>
</body>
</html>
```

In the code, a Web page contains a form with input elements, such as text and a button. The form is accessed in JavaScript using the `id` attribute which is set to **`form1`**. Then, the `length` property of the form object is used to retrieve the number of elements in a form.

Thus, the statement `form1.length` returns the value 4, that is stored in the variable **`count`**. Finally, the value of variable **`count`** is displayed in the alert window.

Figure 17.24 shows the number of controls in the form.

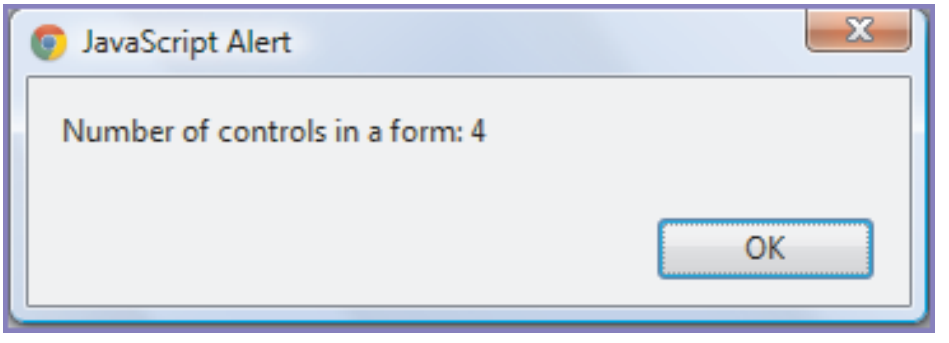



**Figure 17.24: Controls in a Form**

## *17.7 Check Your Progress*

1. A JavaScript function is created under the _____ element.

| | | | |
|---|---|---|---|
| (A) | script | (C) | head |
| (B) | body | (D) | form |

2. Which of the following methods do not change values of the parameters passed to it from the calling function?

| | | | |
|---|---|---|---|
| (A) | Pass By Variable | (C) | Pass By Value |
| (B) | Pass By Reference | (D) | Pass By Object |

3. The _____ object is the parent object from which all JavaScript objects are derived.

| | | | |
|---|---|---|---|
| (A) | String | (C) | Math |
| (B) | Object | (D) | Array |

4. Match the browser objects with their respective description.

| Object | | Description | |
|---|---|---|---|
| (A) | window | 1. | Retrieves the information of the URL loaded in the browser window |
| (B) | history | 2. | Contains information about the browser used by the client |
| (C) | navigator | 3. | Contains a set of URLs visited by the user in a browser window |
| (D) | location | 4. | Represents a browser window |

| | | | |
|---|---|---|---|
| (A) | a-4, b-1, c-2, d-3 | (C) | a-3, b-2, c-1, d-4 |
| (B) | a-1, b-3, c-4, d-2 | (D) | a-4, b-3, c-2, d-1 |

5. Identify the correct `document` object method that retrieves a collection of HTML elements using the specified name.

| | | | |
|---|---|---|---|
| (A) | `getElement()` | (C) | `getElementById()` |
| (B) | `getElementsByName()` | (D) | `getElementsByTagName()` |

## *17.7.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | C |
| 3. | B |
| 4. | D |
| 5. | B |

# Summary

- A function is reusable piece of code which performs calculations on parameters and other variables.
- The return statement passes the resultant output to the calling function after the execution of the called function.
- Objects are entities with properties and methods and resemble to real life objects.
- There are two ways to create a custom object namely, by directly instantiating the object or by creating a constructor function.
- JavaScript provides various built-in objects, such as String, Math, and Date.
- JavaScript also provides browser objects, such as window, history, location, and navigator.
- DOM is a standard technique for dynamically accessing and manipulating HTML elements.
- The DOM provides a document object which is used within JavaScript to access all HTML elements presented on the page.

# *Try it Yourself*

1. Create a Web site for an online store that allow their customers to buy music and movie CDs and DVDs over the Internet. For online shopping, Web site accepts the credit card details from the customer. These credit card details should be validated before allowing the customers to proceed further.

   Assume that you are one of the developer and have to perform the following tasks:

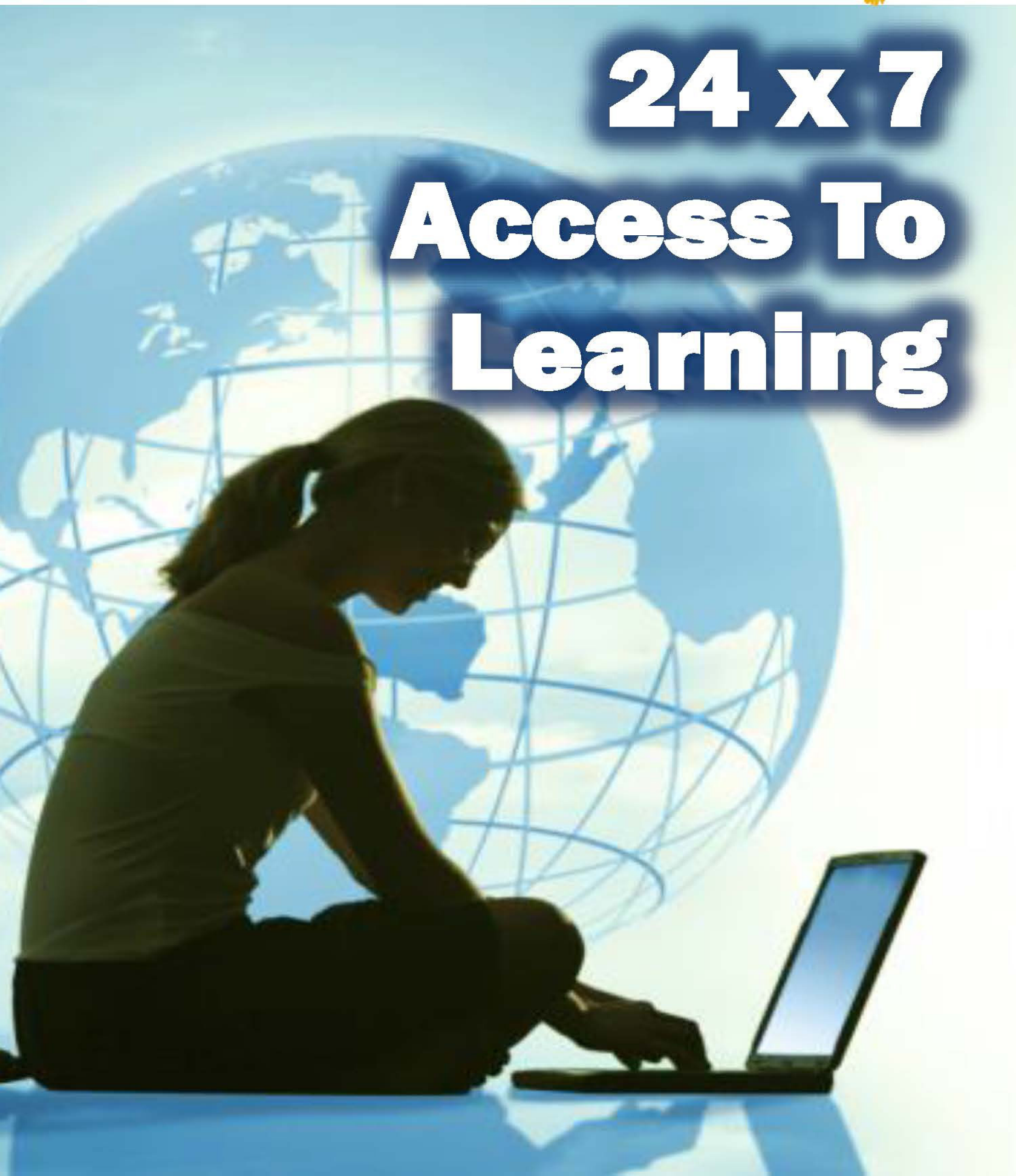
   - Create a Web page that displays images of movie CDs and DVDs. On click of any image, a new window is opened displaying details of the selected movie.
   - Create a Web page with a form containing appropriate controls that allow customers to input their credit card number.
   - Check the credit card details. If the credit card details do not match, an alert box must be displayed stating the same.
   - Allow the customers to navigate through the Web site by providing the Next, Previous, Top, and Bottom buttons.
   - Allow the customers to maintain the history of URLs visited in a browser window.

   Hint: To fulfill these requirements, the developer can make use of the DOM objects.

2. Create a Web page with a label and a button named **`fade`**. When user clicks button, the label along with the text must disappear and again on second click, the label should reappear.

3. Create a expand and collapse panel with a button using HTML, CSS, and JavaScript. Initially, the panel will be in the expanded mode. When user clicks button, the panel will be collapsed and only a bar along with a button is visible. Similarly, when the user clicks button next time, the panel is expanded.

Onlinevarsity



# Session - 18

# Building a Mobile Web Application

Welcome to the Session, **Building a Mobile Web Application**.

This session explains about the different categories and platforms for mobile devices. It also explains about the architectural and design needs of a mobile device. Finally, the session describes the HTML5 support for mobile devices.

In this Session, you will learn to:

- Describe the features of different mobile devices
- List the different types of platforms available for mobile devices
- Explain the design and architectural aspects of a mobile Web site
- Explain the requirements for developing and testing of a mobile Web site
- Explain HTML5 support for a mobile Web site
- List the best practices for optimizing a mobile Web site

## *18.1 Introduction*

Today, access to the Web is not limited to only desktop systems, but is also available on portable and wireless devices, such as mobiles devices. Mobile devices, such as smartphones and tablets are equipped with browsers and network access that provide a better Web experience to their users.

As the number of mobile users has increased, so has the need for a mobile Web experience that is identical to a desktop user experience. In other words, mobile users now look for applications targeting their mobiles that are similar to the ones on their desktops. This has led to the emergence of mobile Web application development.

Different companies follow distinctive strategies for developing mobile applications depending on the need of their users.

## *18.2 Mobile Application Environment*

A mobile device, also known as a handheld device, is a small portable computing device with a small display screen and keyboard. A mobile device has an operating system on which various types of application software are executed. These application software are also known as apps. The most commonly used apps are mobile browsers that display the Web pages.

### *18.2.1 Types of Mobile Devices*

The different categories of mobile phones available in the market are as follows:

- **Basic mobile devices** - These are very basic models with only call and Short Message Service (SMS) facility. They do not provide support for Web browsers or network access.

- **Low-end mobile devices** - These types of models provide more features than a basic mobile device, typically Web support. There is a large market for these mobile devices and they are preferred by users who do not need heavy Internet usage. They also include a basic camera and a basic music player.

  Manufacturers, such as Nokia, Motorola, Sony Ericsson, Samsung, and so forth have gained popularity for offering low cost handset in the global market.

Figure 18.1 shows the low-end mobile devices available from different manufacturers.

Figure 18.1: Low-end Mobile Devices

➔ **Mid-end mobile devices** - These types of mobile devices have gained popularity due to their increased user experience and moderate cost. Some key features of these devices include: medium sized-screen, HTML supported browser, a decent camera, games, and support for applications. They have a proprietary Operating System (OS) that is not well-known and is also not portable across various platforms.

Figure 18.2 shows the mid-end mobile devices available from different manufacturers.

Figure 18.2: Mid-end Mobile Devices

➔ **High-end mobile devices** - These types of mobile devices have advanced features, such as an accelerometer, advanced camera features, and Bluetooth. They have a better look and feel as compared to mid-end mobile devices.

Figure 18.3 shows the high-end mobile devices.

**Figure 18.3: High-end Mobile Devices**

➔ **Smartphones** - These are mobile devices with multitasking capabilities. These devices have a full browser support similar to desktop browsers with wireless LAN and 3G connection. Apart from these, they have several advanced features that are as follows:

- Digital Compass
- Global Positioning System (GPS)
- Touch screen
- Camera with video recording
- TV out
- Bluetooth
- Accelerometer

Some of the popular smartphones available in the market are Apple iPhone, a wide range of BlackBerry smartphones, HTC Magic smartphone, Nokia Lumia, Samsung Galaxy series, and so on.

Figure 18.4 shows the various smartphone devices.

Figure 18.4: Smartphones

➔ **Tablets and notebooks** - These devices are larger than mobile phones. They are mobile computers with a touch screen virtual keyboard and stylus or digital pen. Features of tablets include: multi-touch display, better user experience, high quality screen resolution, better Web support, and multitasking OS with high speed.

Some of the tablets available in the market are BlackBerry PlayBook Tablet PC, Samsung Galaxy Tab, and HCL Me Tab.

Figure 18.5 shows different types of tablet devices.

Figure 18.5: Tablets

## *18.2.2 Mobile Platforms*

A mobile device platform is similar to a software platform. It is basically responsible to interact with the device hardware and run software/services on the mobile device. The mobile platforms are categorized as proprietary and open source. Proprietary platforms are those which are designed and developed by the mobile device manufacturers. These platforms are developed for specific devices and are not supported on all platforms. Open source platforms are those which are freely available to the users. The users can download the source code and alter them as per their requirements.

The brief description of the platforms available on mobile devices is as follows:

1. **Palm OS** - It is a proprietary mobile OS developed by Palm Inc. and was used for Personal Digital Assistants (PDAs). The Palm OS was later on extended to support smartphones. Currently, Palm Inc. has developed webOS, which is based on the Linux kernel. webOS is now acquired by Helwlett-Packward and is used with various devices, such as Pre, Pixi, and Veer phones as well as HP TouchPad tablet.
2. **Blackberry OS** - It is a proprietary mobile OS developed by Research in Motion (RIM) and is based on Java platform. It is primarily used by Blackberry smartphone devices.
3. **iOS** - It is a mobile OS developed by Apple Inc. and was initially referred to as iPhone OS. It is derived from Mac OS X, which is based on the UNIX platform. It was originally developed for iPhone and iPadTouch, but later extended to support other devices, such as iPad and Apple TV. iOS cannot be installed on non-Apple platforms.
4. **Symbian** - It is an open source mobile OS developed for mobile phones. It includes a user interface framework, libraries, and component tools. Symbian is the most widely supported platform for Nokia mobile devices.
5. **Windows Mobile** - It is a mobile OS that runs on top of the Windows Mobile platform.
6. **Linux** - It is an open source OS and is supported by smartphones that are manufactured by Motorola.
7. **Android** - Android is an open source OS developed by Google. It is currently used by smartphones and tablet computers. It can be customized by device manufacturers and community developers to extend the functionality of the devices.

## *18.2.3 Design Aspects of Mobile Web Site*

The design process of a mobile Web site is similar to that of a traditional Web site developed for desktop Web browsers. Still, there are some differences between them that need to be taken into consideration.

An ideal mobile Web site is supported as well as rendered properly by maximum possible browsers and OS.

Some of the basic considerations needed for designing a Web sites for intended mobile device are as follows:

- Resolution and Physical Dimension
- Page Orientation
- Input methods

### Resolution and Physical Dimension

The primary concern in designing a mobile Web site design is the screen resolution. The resolution means the number of pixels (width and height) on the screen of the mobile device. As there are no standards defined for screen resolution, it varies depending on its model and manufacturer.

Table 18.1 lists the resolutions of mobile devices based on their categories.

| Category | Resolutions (in pixels) |
|---|---|
| Low-end mobile devices | 128 x 160 or 128 x 128 |
| Mid-end mobile devices | 176 x 220 or 176 x 208 |
| High-end devices | 240 x 320 |
| Smartphones | 240 x 480, 480 × 320, 640 × 480, or 960 × 640 |

**Table 18.1: Mobile Categories with Resolution**

The resolution of mobile devices is measured in terms of the physical dimensions of the screen. The screen dimensions are either measured diagonally in terms of inches/centimeters or in terms of width and height.

The relation between the physical dimension and resolution is termed as Pixels per Inch (PPI) or Dots per Inch (DPI). The higher DPI results in good print-quality graphics on the mobile device.

Thus, the design of a Web site must be flexible and adjustable to meet the dimensions of the screen on which it is displayed.

Table 18.2 lists the resolution and display sizes of different mobile devices.

| Manufacturer | Model | Screen Size | Resolution (in pixels) | Type |
|---|---|---|---|---|
| Apple | iPad 3 | 9.7” | 2048×1536 | Tablet |
| Apple | iPhone 3GS | 3.5” | 480×320 | Smartphone |
| Apple | iPhone 4S | 3.5” | 960×640 | Smartphone |
| BlackBerry | Torch 9810 | 3.2” | 640×480 | Smartphone |
| HP | TouchPad | 9.7” | 768×1024 | Tablet |
| Samsung | Galaxy S 4G | 4” | 480×800 | Smartphone |
| Samsung | Galaxy S II | 4.52” | 800×480 | Smartphone |
| Nokia | Lumia 800 | 3.7” | 480x800 | Smartphone |

Table 18.2: Resolution and Screen Sizes of Mobile Devices

### ➔ Page Orientation

The mobile devices are also categorized based on their orientation, vertical and horizontal. The vertical orientation devices are also referred to as portrait devices with taller display. Similarly, the horizontal orientation devices are referred as landscape devices with wider display.

Figure 18.6 shows the mobile devices with vertical and horizontal screens.

Figure 18.6: Mobile Devices with Vertical and Horizontal Screens

Today, smartphones and tablets can switch between landscape and portrait views to present the better viewing of a Web page. This rotation capability of changing the view from landscape to portrait or vice-versa is due to the hardware accelerators available in the phones.

Thus, a mobile Web site must be aware of these rotations and should be capable to provide a good user experience in both the orientations.

### ➔ Input Methods

There are different methods to input data on the mobile devices. A mobile device can support more than one input method.

Some of the possible input methods for a mobile device are as follows:

- Numeric keypad
- Alphanumeric keypad (Simple or QWERTY)
- Virtual keypad on screen
- Multi-touch
- External keypad
- Voice and handwriting recognition

Figure 18.7 shows a mobile device with QWERTY keyboard and touch screen.

**Figure 18.7: Mobile Device with QWERTY Keyboard and Touch Screen**

## *18.2.4 Architectural Aspects of a Mobile Web Site*

The Web site developed for a mobile device is a collection of Web pages. Thus, it is essential to understand a few architectural concepts that can help to create meaningful mobile services.

The architecture for designing a mobile Web site begins with the understanding of the information and service that is offered to the mobile user. Some of the concepts that relates to its architecture are as follows:

➔ **Navigation**

Navigation is the path followed by a user to travel in the Web site. As compared to the navigation tree of a desktop site, almost 80% of the information of a desktop site will not be useful to a mobile Web site. Thus, the main focus should be on 20%.

- Design Web pages based on the use cases. For example, some of the common use cases are: performing search for products and prices, performing search for nearest store location, or viewing a Contact us page.
- Arrange Web pages depending on the frequent requirements of the mobile users. This can be achieved through statistical information, or usability tests performed to keep the order updated.
- Restrict the depth of a mobile page to three clicks for a specific use case.
- Design minimum input controls for the form pages.
- Desktop Web site normally has a welcome screen. In case of mobile Web sites, avoid developing welcome screens.
- While designing a service, decide its usability. For example, before designing a service, such as locating a user, determine its usefulness as a service.
- Approximate the number of mobile pages required to separate services, after the home page.

➔ **Perspective**

The perspective of a mobile user is different from a desktop user in terms of needs and accessibility. Hence, a user-centric design approach should be followed for designing mobile Web sites. This ensures that a user completes the task easily and successfully.

Some of the possible users' contexts are as follows:

- What is the location of the user?
- Why is a mobile Web site accessed by the user?
- What are the needs of the user?
- What solution is offered by a mobile application to solve the user's problem?
- Where is the user present while accessing a Web site? For example, a user can access the Web site while walking on the street, at a work place, at a public place, or so on.

➔ **Enhancement**

Enhancement is a simple and powerful technique that can be adopted while designing a mobile Web site. This technique defines compatibility of Web site and allows access to basic content, services, and functionality on all type of mobile devices. Also, it provides a better Web experience on devices with higher standards.

Some of the core principles for enhancing mobile Web sites are as follows:

- Basic content and functionality are accessible in all browsers.
- Enhanced layout and behavior must be provided through external style sheets and JavaScript that are linked with the Web pages.
- Markup elements used on the pages must have proper semantic.
- Web browser settings on a user's device should be considered.

➔ **Use of Web Standards**

The Web standards, such as HTML, CSS, and JavaScript followed in the mobile Web site design must be correctly used. This increases the possibility of displaying pages on large number of devices. The well-formedness of the markup tags used on a page can be achieved by validating them.

Apart from validating HTML pages, use of certain HTML elements can be avoided while designing the Web pages for mobile devices.

The brief description of these elements is as follows:

- **Use of HTML tables** - As the screen size of mobile devices is small, so the use of tables in layouts should be avoided. The reason to avoid tables is as it makes the scrolling difficult and also slows down the page loading in the browser. Also, tables are not rendered properly by the mobile devices.
- **Pop-up windows** - The Web sites with pop-up windows makes the site impractical to work with. Also, all mobile browsers do not provide the support for them. Even, browsers providing support for pop-up windows may result in closing of the existing window.
- **Use of graphics** - The use of graphics increases download time of the pages. Also, they can obstruct the layout of the old mobile browsers, resulting in incorrect display of the page.
- **Use of frames** - Many mobile devices do not provide the support for frames due to usability problems. Also, the HTML5 new specification does not provide the support for frames.

## *18.3 Setting up the Environment for Mobile Applications*

Desktop Web applications are created and tested in the environment for which they are developed. Mobile Web applications are developed to be run on different mobile devices. Hence, they need to be tested in several different environments.

The tools required to develop a mobile Web application are namely, Integrated Development Environment (IDE) and emulators. These are described as follows:

➔ **IDE**

An IDE is a tool used for coding the markup, JavaScript, and CSS. Today in the market, there are different tools which provide the facility to easily build a mobile Web application.

Some of these tools are as follows:

- Adobe Dreamweaver
- Microsoft Expression Web
- Aptana Studio
- Eclipse
- Editplus (text editor)

Latest versions of these tools provide better support for mobile markups. They also provide support for validating pages against mobile Web standards.

➔ **Emulators**

The testing of a mobile Web application can be done using an emulator. An emulator is a software that translates the compiled code to the native platform on which the application is executed.

The emulator runs as a desktop application that allows testing and debugging of a mobile application. It offers the environment similar to a real mobile device on which an application will be executed.

In other words, it imitates the features, such as hardware and OS of a mobile device, to test and debug an application.

Emulators are developed by manufacturers and are often offered free to users. They are either standalone applications or bundled with a Software Development Kit (SDK) for native development.

Some of the popular emulators that either run as standalone applications or in an SDK are as follows:

- Android
- iOS
- webOS
- Blackberry
- Windows Phone
- Opera Mobile

## *18.4 HTML5 Support on Mobiles*

Today, a majority of smartphones and tablets are providing good support for HTML5. Most Android and iOS mobile devices as well as tablets use browsers that are based on Webkit. The Webkit is a layout engine supported by browsers, such as Google Chrome and Apple Safari to render Web pages.

Some of the new features introduced in HTML5 are supported by several mobile devices. This reduces the need of any third party plug-in to get those features on the mobile devices. The features suited for mobile devices are as follows:

- Video
- Audio
- Drag and drop
- Accessing browser history
- Geolocation API for accessing location
- Web storage API to save data on mobile devices
- Offline Web applications (Applications with no Internet connection)

### *18.4.1 HTML5 Markup*

The Web pages developed for a mobile Web application have the same structure as traditional Web pages.

A Web page contains the following sections:

- Heading Structure
- Document Structure

## *18.4.2 Heading Structure*

The heading structure is represented by a `<head>` element defined in an HTML Web page. It defines a `<meta>` tag that is used specifically for mobile browsers.

The brief description for some of the tags defined under `<head>` element is as follows:

➔ **Meta Tag**

Two factors that need to be considered while designing a mobile Web application are its initial display (zoom) scale and orientation. Thus, it is necessary to inform mobile browsers to consider these factors while displaying a Web page.

This can be achieved by using a `<meta>` tag. A `<meta>` tag indicate that the document is optimized for mobile devices and are used to control the display scale, while displaying HTML content on the device. It is specific to mobile browsers.

Table 18.3 lists some of the variations of `<meta>` tag used for different mobile browsers.

| Meta Tag | Description | Supported Mobile Browser |
|---|---|---|
| `<meta name=”HandheldFriendly” content=”true”/>` | Indicates that the content is designed for small-screen handheld devices. The value true prevent the browsers from scaling the content | BlackBerry and others |
| `<meta name=”MobileOptimized” content=”width” />` | Accept width (in pixels) to place the content and forces the layout to one column in the browser | Windows Mobile and Windows Phone |
| `<meta name=”Apple-mobile-web-app-capable” content=”yes”/>` | Indicates that the Web application will run in a full-screen mode | Safari |
| `<meta name=”Format-detection” content=”telephone=no”/>` | Automatic detection of phone numbers is enabled or disabled on Web pages | Safari running on iOS |

**Table 18.3: Meta Tags**

Similarly, a non-standard variation of `<meta>` tag is specified by giving an alternate `<link>` tag. This tag is mostly used with desktop Web pages and defines an alternative URL for displaying the same content on different medias, such as handled devices.

```
<link rel=”alternate” media=”handheld” href=”http://mysite.com” />
```

### ➔ Viewport Meta Tag

This is a technique used to inform the browser that the Web page is optimized for a mobile device. A viewport is the rectangular display area on the screen, where the content of a Web page are displayed by the browser. It contains attributes, such as width and height that can be set to larger or smaller values depending on the total visible area on the screen.

The viewport meta tag is supported on many smartphones, such as iPhone, Android based phones, and browsers, such as Internet Explorer Mobile, Opera Mini, and Opera Mobile.

Table 18.4 lists the attributes of the viewport meta tag.

| Attribute | Description | Value |
|---|---|---|
| width | Defines the horizontal size of the viewport in pixels | Integer value (in pixels) or constant device-width |
| height | Defines the vertical size of the viewport in pixels | Integer value (in pixels) or constant device-height |
| initial-scale | Sets the scale of the page for its initial display. Large value indicates zoomed in where as smaller value indicates zoomed out | Floating value between 0.1 to n |
| minimum-scale | Defines the minimum zoom scale of the viewport | Floating value between 0.1 to n |
| maximum-scale | Defines the maximum zoom scale of the viewport | Floating value between 0.1 to n |
| user-scalable | Allows scaling of application on the mobile devices. In other words, users can zoom in and out in the application | no or yes |

**Table 18.4: Viewport Meta Tag Attributes**

Code Snippet 1 demonstrates the viewport meta tag to set device width for a mobile Web page.

**Code Snippet 1**:

```
<!DOCTYPE html>
<head>
<title>Mobile</title>
```

```
<!-- <meta name="viewport" content="width=device-width, user-
scalable=no"> -->
</head>

<body>
 <header> Mobile Design </header>

 <NAV>
 <a href="home.html">Home</a> | <a  href="aboutUs.html">
  About Us</a> |
  <a href="contactUs.html">Contact Us</a>
 </NAV>

 <section id="intro">
   <p>This is the introductory text to my mobile Web application.
   </p>
  <p>
    Mobile development is more than cross-browser, it should be
    cross- platform. The vast number of mobile devices makes
    thorough testing a practical impossibility, leaving
    developers nostalgic for the days when they only had to support
    legacy browsers.
 </p>
 </section>

</body>
</html>
```

In the code, the viewport width has been set to "`device-width`" which sets the device width to 320px. As all phones does not support the same width, so setting "`device-width`" allows the mobile browsers to set width according to the device width. Also, setting the attribute `user-scalable=no` prevents the user from increasing the display scale of the application. The default width taken for the viewport is 980px which is approximately the desktop size.

Figure 18.8 displays the Web page on **Opera Mobile Emulator**, before setting the viewport meta tag.

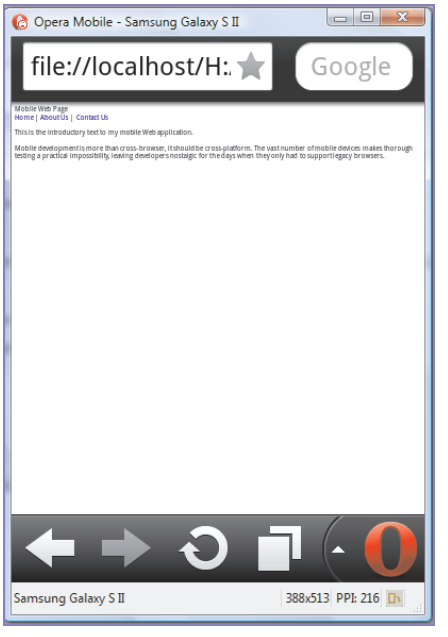



Figure 18.8 : Output - Without Viewport

Figure 18.9 displays the Web page on **Opera Mobile Emulator** after removing the comments from Code Snippet 1. The code sets the viewport meta tag.

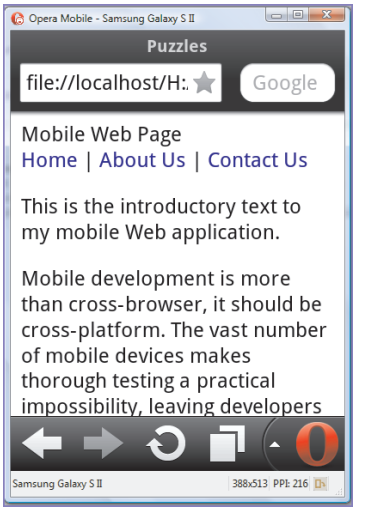



Figure 18.9: Output - Setting Viewport

➔ **Titles** - Apart from `<meta>` tag, `<head>` element also contains a `<title>` tag which should be taken into consideration, as the mobile browser is small as compared to a desktop browser. The text selected for `<title>` tag should be meaningful, short, and precise. It should be between four and eight words, as some old mobile devices truncate the long titles after 10 or 12 words.

➔ **Icons** - To add icons to a mobile Web page, images in GIF or PNG format could be used. These formats are compatible with mobile devices, as they are easy to export and are optimized in size.

For example,

```
<link rel="icon" type="image/png" href="mobile.png" />
```

From HTML5 onwards, Android supports the apple-touch-icon-precomposed meta tag in order to display high-resolution icons.

## *18.4.3 Document Structure*

The document structure is represented by a <body> element in an HTML Web page. The <body> element of a mobile Web application defines the content that are displayed to the user. Some of the elements used in the <body> element of a mobile Web page are as follows:

➔ **Layouts**

The HTML5 new tags that provide semantics for the layout of an HTML document are as follows:

- <article> - An independent portion of the document or site
- <aside> - Content that is tangential to the main part of the page or site
- <figcaption> - Caption for a figure
- <figure> - A figure or quotation pulled out of the flow of text
- <footer> - The footer of a document or section
- <header> - The header of a document or section
- <hgroup> - A group of headings
- <nav> - A navigation section
- <section> - Identifies a block of content

➔ **Images**

Images can be used in mobile Web applications for pictorial representation. Almost all mobile browsers understand formats, such as GIF, JPEG, and PNG.

However, images should not be used for setting background, buttons, links, or presenting titles on a mobile Web page. This is because images increase the number of request to the Web server and also load time of the Web page.

The <img> tag is used to display image on a Web page. The attributes of <img> tag, such as width, height, and alt should be specified, as it reduces the rendering time of the image.

Code Snippet 2 demonstrates a mobile Web page with an `<img>` tag.

**Code Snippet 2**:

```
<!DOCTYPE html>
<head>
<title>Images</title>
 <meta name="viewport" content="width=device-width, user-scalable=no"/>
</head>
<body>
 <article>
  <h2> Gift Basket </h2>
  <img src="gift.jpg" width="200" height="200" alt="Gift Items" />
 </article>
</body>
</html>
```

Figure 18.10 displays image on a mobile Web page.

Figure 18.10: Image Element

## ➔ Lists

A mobile Web application supports different types of lists. The various list types are as follows:

a. Ordered lists - Used for navigational menus and are defined using `<ol>` tag on a Web page.

b. Unordered lists - Used for presenting objects of same type and are defined using `<ul>` tag on a Web page.

c. Definition lists - Used for presenting information as key/value pairs and are defined using `<dl>` tag on a Web page.

Code Snippet 3 demonstrates the use of a definition list to present the capacity list of different hardware components on a mobile Web page.

**Code Snippet 3**:

```
<!DOCTYPE html>
<head>
  <title>List on Mobile</title>
  <meta name="viewport" content="width=device-width, user-
  scalable=no"/>
 <link rel="apple-touch-icon-precomposed" href="mobile.png" />
</head>
```

```
<body>
 <header>
  <h5> Hardware Components Capacity List </h5>
 </header>
 <section>
  <dl>
   <dt>RAM Memory</dt>
   <dd>4.00GB</dd>
   <dt>Hard Disk</dt>
   <dd>500GB</dd>
   <dt>LAN</dt>
   <dd>Wifi, Bluetooth</dd>
   <dt>CPU</dt>
   <dd>2.93GHz</dd>
  </dl>
 </section>
</body>
</html>
```

In the code, the `<dl>` tag is used to represent the information as key and value pair format.

Figure 18.11 displays the definition list on a mobile Web page.

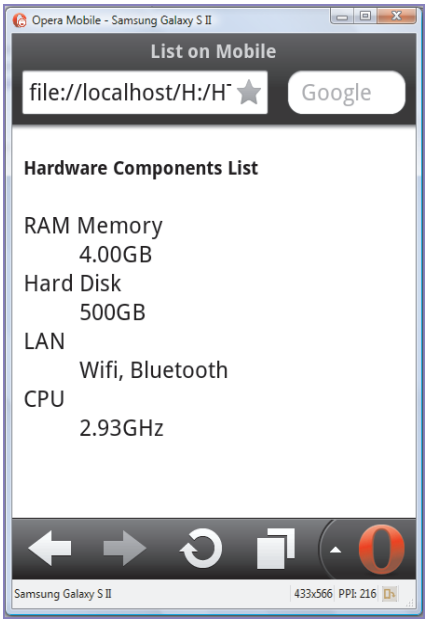



**Figure 18.11: Output – Definition List**

### ➔ Links

Hyperlinks are used to link pages in a Web application. They are one of the important elements on Web pages and this holds true in case of mobile Web applications too.

A hyperlink is defined using `<a>` tag with `href` attribute. The `href` attribute is set to the URL of a resource. Apart from `href`, the `<a>` tag should also have `accesskey` attribute specified with it. The `accesskey` attribute is a keyboard shortcut and is useful for mobile devices that have support for access keys.

Code Snippet 4 demonstrates the use of `<a>` and `<ul>` tag to create a navigation list on a mobile Web page.

**Code Snippet 4**:

```
<!DOCTYPE html>
<head>
<title>Navigation list</title>
<meta name="viewport" content="width=device-width, user-scalable=no"/>
<link rel="apple-touch-icon-precomposed" href="mobile.png" />
</head>
<body>
<header>
 <h5> Main Menu </h5>
</header>
<NAV>
 <ul>
   <li><a title="Comprehensive Animation" href="comprehensive.
    html" accesskey="1">Comprehensive Animation Pro</a></li>
   <li><a title="2D and 3D" href="animation_3d.html"
                accesskey="2">2D &amp; 3D Animation </a></li>
   <li><a title="3D Animation" href="animation_2d.html"
               accesskey="3">Animation &amp; 3D</a></li>
 </ul>
</NAV>
</body>
</html>
```

Figure 18.12 displays the navigation list on a mobile Web page.

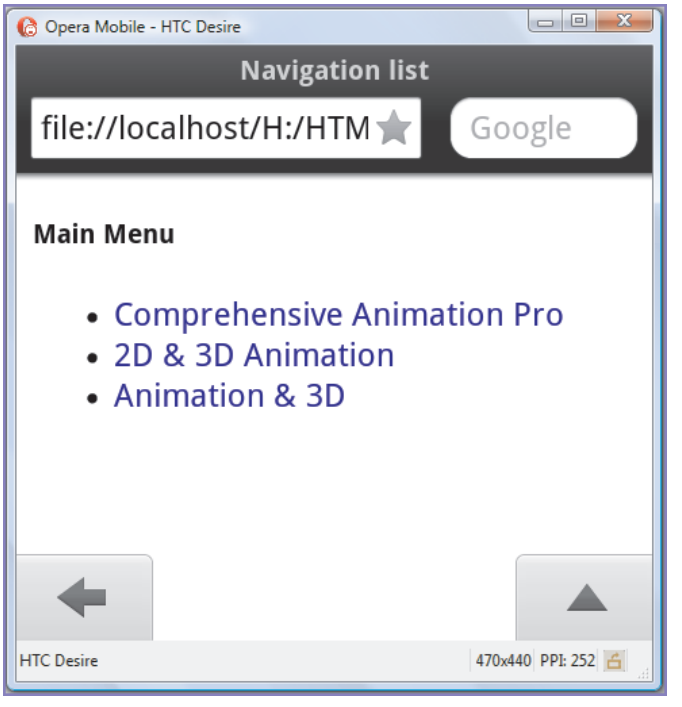



Figure 18.12: Output – Navigational List

### ➔ Tel Scheme

As mobile devices are basically phones, hence, links can be created to perform phone call actions. This is achieved using the `tel:<phone number>` scheme embedded with a hyperlink. The `tel` scheme is useful in situations, such as accessing helpdesk systems or voicemail systems.

Code Snippet 5 demonstrates a `tel` scheme defined on a mobile Web page.

**Code Snippet 5**:

```
<!DOCTYPE html>
<head>
 <title>Mobile Application</title>
 <meta name="viewport" content="width=device-width,minimum-
                                scale=1.0,maximum-scale=1.0"/>
</head>
```

```
<body>
 <header>
  <section><img src="logo.png" border="1" alt="Arena Animation"
    title="Arena Animation Home"/> <b> Animation | Multimedia </b>
  </section>
  <nav>
   <ul>
     <li><a title="Comprehensive Animation"
         href="#">Comprehensive Animation Pro</a></li>
     <li><a title="2D and 3D" href="#">2D &amp; 3D Animation
          </a></li>
     <li><a title="3D Animation" href="#">Animation &amp; 3D
         </a> </li>
  </ul>
 </nav>
 </header>
 <section id="intro">
  <p> Arena Animation is leader in animation and multimedia education  with the
widest network of centers across the country. Over a span of 14 years, 250,000
students and professionals have, through Arena, found their calling in
animation, graphics, print publishing, web  designing & films.</p>
 </section>


 <footer>
  <p>
   <a href="tel:+91 22 2827 2300">Contact us </a> <br/>
   Copyright &copy; 2012 Aptech Ltd.</p>
 </footer>
</body>
</html>
```

Figure 18.13 displays the output of a Web page in a tablet selected in an **Opera Mobile Emulator**.

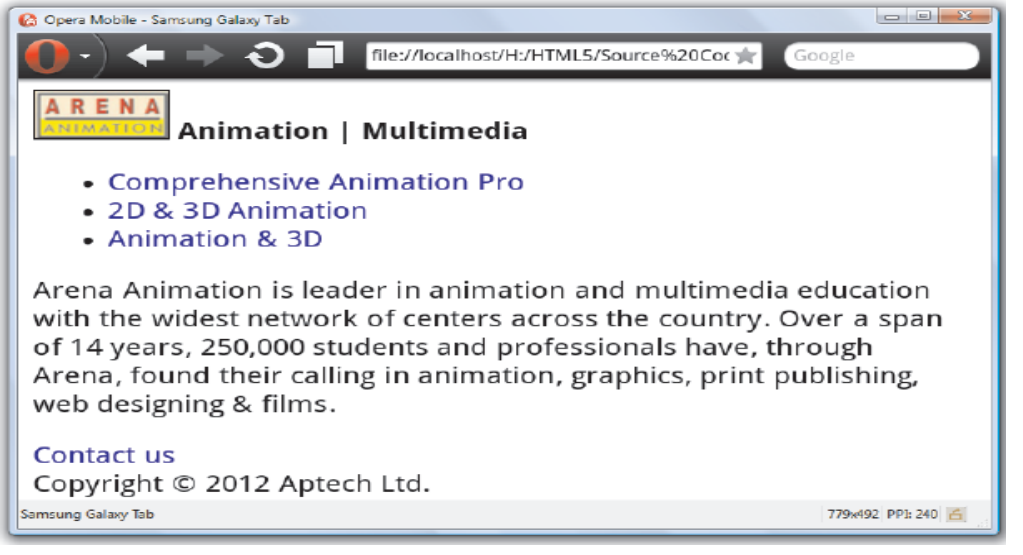



Figure 18.13: Output – Opera Mobile Emulator

## *18.5 CSS for Mobile*

CSS3 provides properties for adding colors, selectors, borders, backgrounds, and so on for effective appearance of a Web page.

Most modern mobile browsers support following features of CSS3:

- Rounded corners
- Images with borders
- Adding shadow effect on text and boxes
- Animations
- Transitions
- Multi-column layout

Apart from W3C specifications that are laid for CSS3, modern browsers have provided new styles for CSS3. These styles are specific to each browser. Therefore, to add these styles on a Web page, the relevant properties need to be prefixed with the browser specific keyword.

The property prefixed with the keyword represents the browser on which it is supported. Table 18.5 lists the keywords with their supported browsers.

| Keyword | Browser |
|---|---|
| -moz | Firefox |
| -ms | Internet Explorer |
| -o | Opera |
| -webkit | Google Chrome and Safari |

Table 18.5: Keywords Supported by Browsers

Code Snippet 6 demonstrates the CSS3 properties for a Web page that was developed in Code Snippet 5. These properties have been explored in the earlier sessions.

**Code Snippet 6**:

```
<!DOCTYPE html>
<head>
 <title>Mobile Application</title>
 <meta name="viewport" content="width=device-width"/>
 <style>

 html, body {
   margin: 0;
   padding: 0;
   border: 0;
   font-size: 100%;
   font-weight: normal;
   vertical-align: baseline;
   background: transparent;
 }

 body {
   line-height: inherit;
 }
 #nav {
   width:500px;
   height:60px;
```

```
   background-color:#A4A4FF;
  margin:0px;
  margin-top:5px;
  padding:0px;
  }
 ul#navigation{
  margin:0px;
  border-left:1px solid #c4bbe7;
  border-right:1px solid #c4dbe7;
  padding-top:5px;
}
ul#navigation li{
  display:inline;
  font-size:12px;
  font-weight:bold;
  margin:0;
 padding:0;
  float:left;
  position:relative;
  border-top:1px solid #c4dbe7;
  border-bottom:2px solid #c4dbe7;
}
ul#navigation li a{
  padding:10px;
  color:#616161;
  text-shadow:1px 1px 0px #fff;
  text-decoration:none;
  display:inline-block;
  border-right:1px solid #fff;
  border-left:1px solid #C2C2C2;
  border-top:1px solid #fff;
```

```
     background: #f6f6f5;
-webkit-transition:color 0.2s linear, background 0.2s linear;
  -moz-transition:color 0.2s linear, background 0.2s linear;
  -o-transition:color 0.2s linear, background 0.2s linear;
  transition:color 0.2s linear, background 0.2s linear;
 }
 ul#navigation li a:hover {
  background:#f8f8f8;
  color:red;
}
 p {
 border: 0;
 font-size: 100%;
 font-weight: normal;
 vertical-align: baseline;
 background: transparent; }

 a {
  margin: 0;
  padding: 0;
  font-weight: normal;
 }
 #intro {
  background-color: yellow;
  border: solid black 2px;
  width: 600px;
  height: 150px;
 }
 </style>
</head>
```

```
    <body>
<header>
 <section><img src="logo.png" border="1" alt="Arena Animation"
 title="Arena Animation Home"/> <b> Animation | Multimedia </b>
 </section>
 <nav id="nav">
  <ul id="navigation">
   <li><a title="Comprehensive Animation" href="#">Comprehensive
                                Animation Pro</a></li>
    <li><a title="2D and 3D" href="#">2D &amp; 3D Animation </a></li>
    <li><a title="3D Animation" href="#">Animation &amp; 3D</a></li>
  </ul>
 </nav>
</header>
<section id="intro">
  <p style="font-size: 12px; font-style: italic; color:#0000FF">
  Arena Animation is leader in animation and multimedia education
 with the widest network of centers across the country.
 Over a span of 14 years, 250,000 students and professionals have,
 through Arena, found their calling in animation, graphics, print
  publishing, web designing & films.</p>
</section>
<footer>
 <p>
    <a href="tel:+91 22 2827 2300">Contact us </a> <br/>
    Copyright &copy; 2012 Aptech Ltd.
 </p>
</footer>
</body>
</html>
```

Figure 18.14 displays the output of a mobile Web page with the CSS3 styles applied.

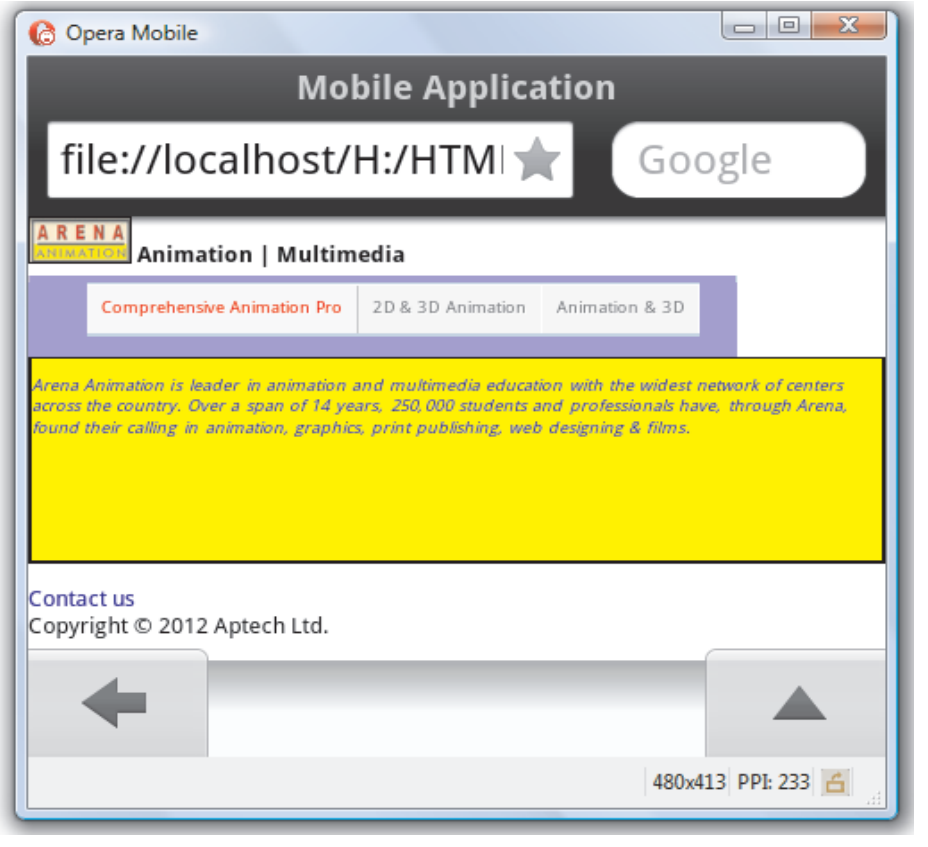



Figure 18.14: Mobile Web Page with Styles Applied

### ➔ Media Queries for Browser Detection

Media queries are used to target specific features, such as screen width, orientation, and resolution of the devices. The use of a media query is to display HTML pages on various devices, such as computers and mobile devices with different styles based on their media types.

Earlier, CSS2 allowed creating style sheets for specific media types, such as screen and print. CSS3 has been enhanced further with the features of media queries. In media queries, expressions are added for specific media type, then checking for condition is done, and finally, respective style sheet is applied to a Web page.

Media queries are used in two ways that are as follows:

1. Inline within a CSS style sheet
2. In the `<link>` tag as "`media`" attribute

Code Snippet 7 shows the markup to apply a style sheet named `screen.css` to a device with screen and set the viewing-width of the area to 480.

**Code Snippet 7**:

```
<link type="text/css" rel="stylesheet" media="only screen and (max-device-
width: 480px)" href="screen.css" />
```

Some browsers can also interpret the CSS media queries placed inside a `.css` file.

Code Snippet 8 shows the code to change the background color of a Web page depending on the device width.

**Code Snippet 8**:

```
@media only screen and (max-device-width: 480px) {
body {
  background-color: #666;
}
}
```

CSS media queries can also be used to define style sheets for different orientations.

Code Snippet 9 shows the markups to serve style sheets based on the orientation of the device.

**Code Snippet 9**:

```
<link rel="stylesheet" media="all and (orientation: portrait)"
href="portrait_orientation.css" />
<link rel="stylesheet" media="all and (orientation: landscape)"
href="landscape_orientation.css" />
```

## *18.6 Optimizing a Site for a Mobile*

The needs of mobile users are different from desktop users, in terms of screen size and connectivity issues. Thus, mobile Web sites should be optimized for better performance.

Some of the best practices that can be followed for mobile applications are as follows:

1. Design of a mobile Web site that is simple to fit on small screens. The simplest layout which can be followed for mobile Web sites is a single column display.
2. Avoid horizontal scrolling, as some phones do not support horizontal scrolling and hide the content on the screen.
3. Use buttons instead of providing many tiny links, as this can annoy the mobile users. This is also useful in situations where scaling of the page is disabled.
4. Offer mobile users with the choice for viewing the site in the full version.
5. Create cookies to store the user's choice for viewing the full version of the site. This helps to restore

the same view on every visit to the Web site.

6. Avoid creating complex forms with many input fields, as data entry can be difficult on mobile devices compared to the desktops. Also, relevant input fields should be used for accepting the data.
7. Limit the use of images due to bandwidth restrictions on mobile devices. Also, large and high-resolution images must be avoided.
8. Add mobile specific functionalities, such as built-in GPS facility or call-in action links. This makes the Web sites similar to native applications.
9. Use of good foreground and background colors is important as they makes the sites readable on small screens.
10. Select technologies that are compatible with old mobile devices. Also, provide alternatives for functionalities, such as cookies, tables, style sheets, fonts, colors, and so on.
11. Avoid use of pop-up windows, tables for layout, frames, and image maps in the mobile Web site design.

## *18.7 Check Your Progress*

1. _____ is the mobile device with multitasking capabilities.

| | | | |
|---|---|---|---|
| (A) | Smartphone | (C) | Laptop |
| (B) | Telephone | (D) | High-end mobile |

2. Which of the following elements should be avoided while designing a mobile Web site?

| | | | |
|---|---|---|---|
| (A) | `table` | (C) | `frame` |
| (B) | `image` | (D) | `form` |

3. Identify the factors that need to be considered while designing a mobile Web application.

| | | | |
|---|---|---|---|
| (A) | Display scale | (C) | Input mechanism |
| (B) | Orientation | (D) | Operating system |

4. Match the different mobile operating systems with their respective descriptions.

| Mobile OS | | Description | |
|---|---|---|---|
| (A) | Android | 1. | Based on Java platform |
| (B) | iOS | 2. | Based on windows mobile platform |
| (C) | Palm | 3. | Derived from MAC OS based on UNIX platform |
| (D) | BlackBerry | 4. | Open source OS developed by Google |

| | | | |
|---|---|---|---|
| (A) | a-4, b-1, c-2, d-3 | (C) | a-3, b-1, c-2, d-4 |
| (B) | a-1, b-3, c-4, d-2 | (D) | a-4, b-3, c-2, d-1 |

5. Which of the following is the non-standard variation of `<meta>` tag?

| | | | |
|---|---|---|---|
| (A) | Viewport meta tag | (C) | Media queries |
| (B) | WebKit extensions | (D) | link tag |

## *18.7.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | A, C |
| 3. | A, B |
| 4. | D |
| 5. | D |

# Summary

- A mobile device is a small portable computing device with a small display screen and keyboard.
- The different categories of mobile devices available in the market are: basic model, low-end mobile devices, mid-end mobile devices, high-end mobile devices, smartphones, and tablets.
- A mobile platform is basically responsible to interact with the device hardware and run software/ services on the mobile device.
- Different platforms for mobile devices include: Palm OS, Blackberry, iOS, Symbian, Windows Mobile, and Android.
- An ideal mobile Web site is supported and rendered properly by maximum possible browsers and OS.
- Two factors that need to be considered while designing mobile Web application are its initial display (zoom) scale and orientation.
- The use of media query is to display HTML pages on different devices with different styles based on their media types.

# *Try it Yourself*

1. Create a mobile Web site for a restaurant. The Web site should display details of the restaurant, such as cuisine, its specialty, and prices of food items served there. It should also allow the user to book a table or order food from the Web site.

   Some points to be followed while designing the application are as follows:

   a. The Web site need to be displayed on all types of mobile devices and should fit according to the display size.

   b. Use CSS3 properties to enhance the appearance of the Web site.

   c. Provide a facility for a user to make a phone call and place an order on the Web site.

THE WORLD HAS GONE DIGITAL. WHAT ABOUT YOU?

Onlinevarsity

YOUR DIGITAL WORLD OF LEARNING

# Session - 19

## Canvas and JavaScript

Welcome to the Session, **Canvas and JavaScript**.

This session describes the new <canvas> element in HTML5. It also explores the procedure to draw lines, use color, and transparency in Web pages. This session further explains how to work with drawing objects, images, text, and create Web page events with JavaScript and jQuery.

In this Session, you will learn to:

- Describe Canvas in HTML5
- Explain the procedure to draw lines
- Explain the procedure to use color and transparency
- Explain the procedure to work with various drawing objects
- Describe working with images and text
- Describe the procedure to create Web page events with JavaScript and jQuery
- Describe the process of including external content in Web pages

## *19.1 Introduction*

Canvas is one of the most interesting features added in HTML5. The <canvas> element supports advanced graphics and interactions in many forms. In earlier versions of HTML, you could achieve this by using plug-ins. Using the <canvas> element eliminates the need for such plug-ins and makes working with graphics easier and more efficient. The <canvas> element is a drawing area where the user can draw graphics, use images, add animations, and also add text for enhancing the user experience on Web pages.

## *19.2 Canvas Element*

The <canvas> element in HTML5 can be used to draw much more than just rectangles on Web sites - it can be used to dynamically draw graphics using JavaScript. This improves the overall performance of Web sites and avoids the requirement to download images from the sites. The <canvas> element is represented like a rectangle on a page and allows the user to draw arcs, text, shapes, gradients, and patterns. By using <canvas>, the user can draw many complex shapes and also apply various effects and transformations.

The <canvas> in HTML5 is like the , <table>, or <a> tag except that the content used in it is rendered through JavaScript.

The <canvas> element is simple and easy to use with JavaScript. The <canvas> element does not contain any drawing abilities, instead, the drawing is done using a JavaScript code. To make use of the <canvas> element, a user has to add the <canvas> tag on the HTML page.

Code Snippet 1 demonstrates the use of <canvas> element.

**Code Snippet 1**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title>Canvas</title>
  <style>
      canvas{border: medium double red; margin: 4px}
     </style>
  </head>
 <body>
  <canvas width="278" height="200"></canvas>
 </body>
</html>
```

In the code, the `<style>` element is used to display the border of the `<canvas>` element. The `height` and `width` attributes specify the size of the `<canvas>` element on the page.

Figure 19.1 displays the `<canvas>` element.

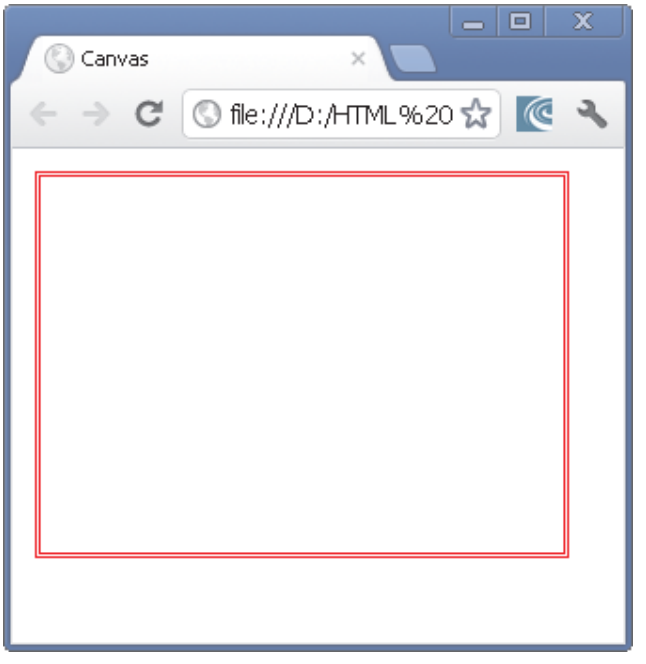



**Figure 19.1: <Canvas> Element**

To draw a `<canvas>` element, the user can use a context object. The context object contains the drawing functions for a specific style of graphics. Two-Dimensional (2d) context is used to work with 2d operations.

The `<canvas>` element in DOM exposes the `HTMLCanvasElement` interface. This interface provides the methods and properties for changing the presentation and layout of canvas elements. The `HTMLCanvasElement` has a `getContext(context)` method that returns the drawing context for the canvas.

Code Snippet 2 demonstrates the 2d context object for the canvas.

**Code Snippet 2**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title>Canvas</title>
  <script>
   window.onload=function() {
```

```
  var canvas = document.getElementById('mCanvas');
  var ctext = canvas.getContext('2d');
  ctext.beginPath();
  ctext.rect(18, 50, 200, 100);
  ctext.fillStyle = "DarkBlue";
  ctext.fill();
  };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="578" height="200"></canvas>
</body>
</html>
```

In the code, the `height` and `width` attributes define the height and width of the `canvas` element respectively. In the initializer function, the DOM object is accessed through the `id` attribute and gets a `2d` context by using the `getContext()` method. Here, the rectangle is created by using the `rect(18, 50, 200, 100)` method with `x`, `y`, `height`, and `width` parameters and is positioned at left corner of the page.

Figure 19.2 displays the canvas.

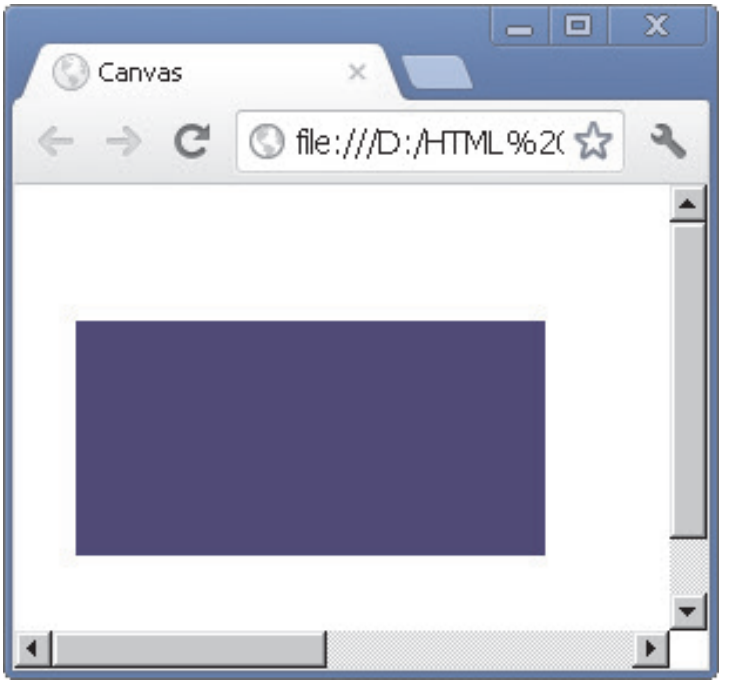



**Figure 19.2: Canvas**

## *19.3 Drawing a Line in Canvas*

You can create lines in a canvas using the `stroke()`, `beginPath()`, `lineTo()`, and `moveTo()` methods.

The following is the syntax to create a line in canvas:

**Syntax**:

```
ctext.beginPath();
ctext.moveTo(x,y);
ctext.lineTo(x,y);
ctext.stroke()
```

where,

`ctext` - specifies a context object

`beginPath()` - Specifies a new drawing path

`moveTo()` - Specifies the creation of new sub path to the given position

`lineTo()` - Specifies the drawing of a line from the context position to the given position

`stroke()` - Specifies how to assign a color to the line and display it

Code Snippet 3 demonstrates creating a line in HTML5 canvas.

**Code Snippet 3**:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title>Line</title>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
border: 1px solid red;
   }
  </style>
<script>
```

```
  window.onload=function(){
   var canvas=document.getElementById("mCanvas");
   var ctext=canvas.getContext("2d");
   ctext.beginPath();
   ctext.moveTo(100,150);
   ctext.lineTo(250,50);
   ctext.lineWidth=5;
   ctext.strokeStyle="blue";
   ctext.stroke();
  };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="360" height="200"></canvas>
</body>
</html>
```

In the code, the `height` and `width` attributes are defined. The initializer function has the DOM object which is accessed through the `id` attribute and gets a `2d` context by using the `getContext()` method. The `beginPath()` method is called through the context object to draw the path of the line. The `moveTo(100, 150)` method is called that creates a new path for the given point to place the drawing cursor. This method moves the position of the window to the upper-left corner by giving the `x` and `y` coordinates. The `lineTo(250, 50)` method is called to draw the line from the context point to given point. The `lineWidth` property is specified as `5` to define the width of the line on the canvas. The `strokeStyle` property sets the color of the line to blue. The `stroke()` method assigns the color to the line.

Figure 19.3 displays a line drawn in a canvas.

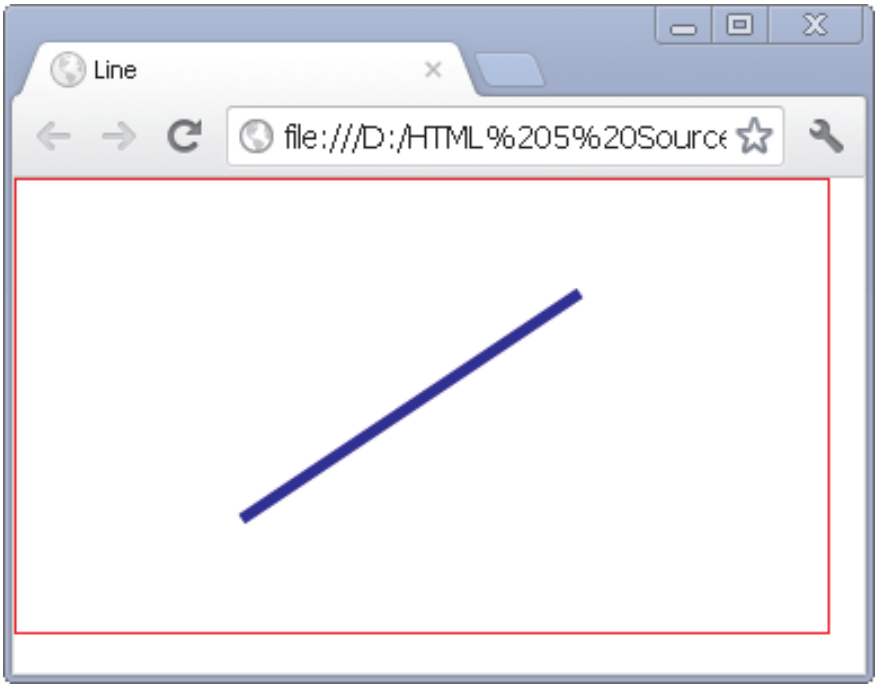



**Figure 19.3: Line Drawn in a Canvas**

## *19.4 Working with Drawing Objects in Canvas*

HTML5 canvas allows the user to work with different types of drawing objects. Following objects can be drawn on a canvas element:

### ➔ Rectangle

With HTML5 canvas, the user can create a rectangle using the `rect()` method. The HTML5 canvas is placed by using the `x` and `y` parameters and appropriately sized through `height` and `width` properties. There is a collection of methods and properties that are used to draw different types of shapes. Table 19.1 lists the common properties and methods of various shapes.

| Properties and Methods | Description |
|---|---|
| `fillStyle` | The values can be gradient, pattern, or a CSS color. The default property style is solid black, but the user can set the color according to the requirements. |
| `filRect(x,y,width, height)` | Enables the user to draw a rectangle with the existing fill style. |
| `strokeStyle()` | The values can be gradient, pattern, or a CSS color. |
| `strokeRect(x,y, width, height)` | Enables the user to draw a rectangle with the existing stroke style. This property is used to draw the edges of the rectangle. |

| Properties and Methods | Description |
|---|---|
| `clearRect(x, y, width, height)` | Used to clear the pixels in a rectangle. |

Table 19.1: Common Properties and Methods of Various Shapes

Code Snippet 4 demonstrates how to create a rectangle in HTML5 canvas.

**Code Snippet 4**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   #mCanvas {
    border: 1px solid green;
   }
   body {
    margin: 0px;
    padding: 0px;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById('mCanvas');
    var ctext = canvas.getContext('2d');
    ctext.beginPath();
    ctext.rect(30, 50, 150, 100);
    ctext.fillStyle = "Magenta";
    ctext.fill();
    ctext.lineWidth = 5;
    ctext.strokeStyle = 'black';
    ctext.stroke();
```

```
  };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="278" height="200"></canvas>
</body>
</html>
```

In the code, the `height` and `width` attributes are defined. The initializer function has the DOM object which is accessed through the `id` attribute and gets a `2d` context by using the `getContext()` method. The `beginPath()` method is called through the context object to draw the rectangle. The `rect(30, 50, 150, 100)` method takes `x`, `y`, `height`, and `width` as the parameters. The `fillStyle` property fills the rectangle with magenta color. The `fill()` method is used to paint the rectangle. The `lineWidth` property is specified as `5` to define the width of line on the canvas. The `strokeStyle` property sets the stroke style of the rectangle to black. The `stroke()` method assigns the color to the rectangle.

Figure 19.4 displays a rectangle drawn on the canvas.

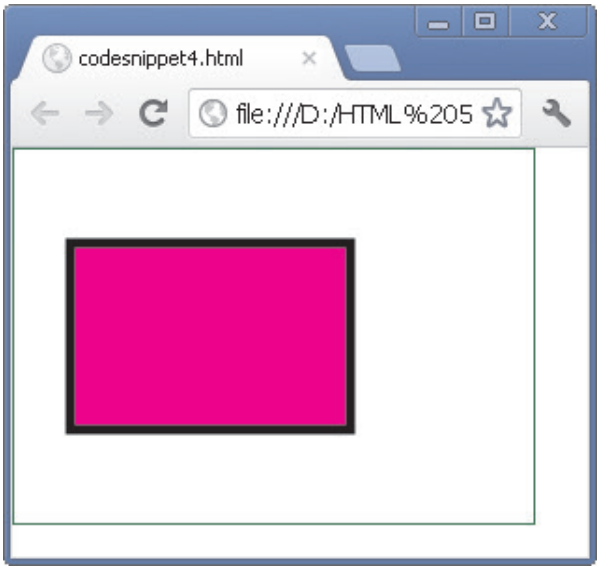



**Figure 19.4: Rectangle Drawn on the Canvas**

## ➔ Arcs

With HTML5 canvas, the user can create an arc by using the `arc()` method. Arcs are represented using a start angle, an end angle, a radius, a center point, and the drawing direction (anticlockwise or clockwise).

The syntax to draw an arc in HTML5 is as follows:

**Syntax**:

```
arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

where,

`x`, `y` - Specifies the coordinates of the center of an arc

`radius` - Specifies the distance from the center to any point on the circle

`startAngle`, `endAngle` - Specifies the start and end points in the arc

`anticlockwise` - Draws the arc clockwise or anticlockwise and accepts a `boolean` value

Code Snippet 5 demonstrates how to create an arc in HTML5 canvas.

**Code Snippet 5**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid black;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("mCanvas");
    var ctext = canvas.getContext("2d");
    var x = canvas.width / 2;
    var radius = 75;
    var startAngle = 1.1 * Math.PI;
    var endAngle = 1.9 * Math.PI;
    var ctrClockwise = false;
```

```
  ctext.beginPath();
   ctext.arc(x, y, radius, startAngle, endAngle, ctrClockwise);
   ctext.lineWidth = 25;
   // line color
   ctext.strokeStyle = "DarkGreen";
   ctext.stroke();
  };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="278" height="250"></canvas>
</body>
</html>
```

In the code, the `beginPath()` method is called through the context object to draw an arc by using the `arc()` method which has `x`, `y`, and `radius` as the parameters. The `x` and `y` are the coordinates of the circle, radius is the distance from the center to draw the arc on the canvas. The `startAngle` and the `endAngle` are the start and end points of the `arc` respectively. The `anticlockwise` specifies the direction of the arc between the two start and end points.

Figure 19.5 displays an arc in HTML5 canvas.

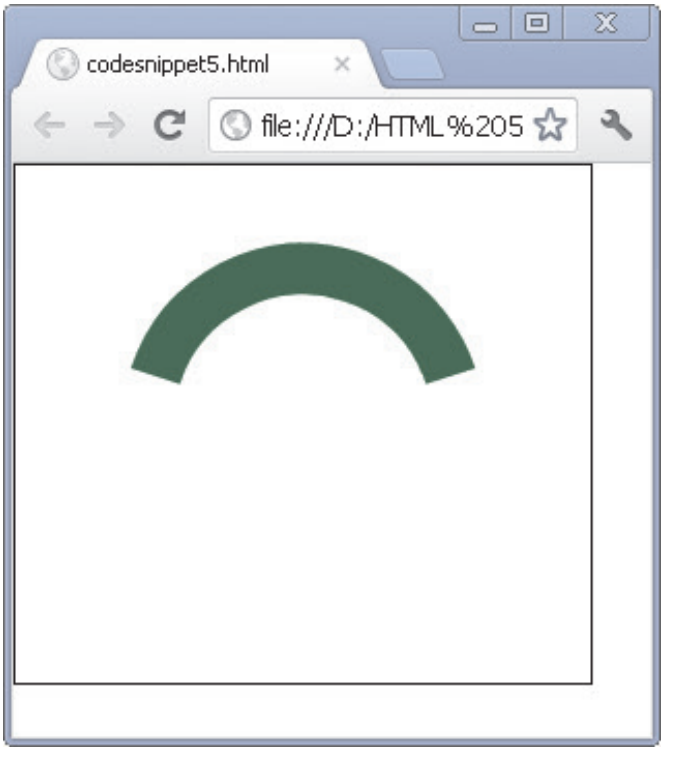



Figure 19.5: Arc in HTML5 Canvas

### ➔ Circle

In HTML5, you can draw a circle using the `arc()` method. You have to set the start angle with `0` and the end angle is specified as `2 * PI`.

The following is the syntax to draw a circle in HTML5 is as follows:

**Syntax**:

```
arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

where,

`x`, `y` - Specifies the coordinates of the center of a circle

`radius` - Specifies the distance from the center to any point on the circle

`startAngle`, `endAngle` - Specifies the start and end points in the circle

`anticlockwise` - Draws the circle clockwise or anticlockwise and accepts a `boolean` value

Code Snippet 6 demonstrates how to create a circle using HTML5.

**Code Snippet 6**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
#mCanvas {
    border: 1px solid blue;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("mCanvas");
    var ctext = canvas.getContext("2d");
    var ctrX = canvas.width / 2;
    var ctrY = canvas.height / 2;
    var radius = 70;
    ctext.beginPath();
    ctext.arc(ctrX, ctrY, radius, 0, 2 * Math.PI, false);
    ctext.fillStyle = "DarkOrchid";
    ctext.fill();
    ctext.lineWidth = 4;
    ctext.strokeStyle = "black";
    ctext.stroke();
};
  </script>
</head>
```

```
<body>
 <canvas id="mCanvas" width="356" height="150"></canvas>
 </body>
</html>
```

In the code, a circle is defined by using the `arc()` method which has `ctrX`, `ctrY`, and `radius` as the parameters. To define the arc with the points the `startAngle` is set to `0` and the `endAngle` is specified as `2*PI`. The `anticlockwise` defines the direction of the path of an arc between the two start and end points.

Figure 19.6 displays a circle in HTML5 canvas.

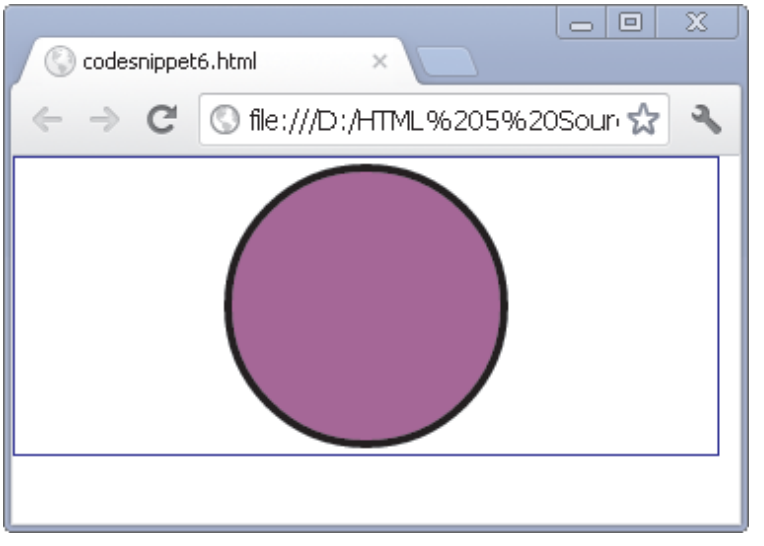



Figure 19.6: Circle in HTML5 Canvas

### ➔ Bezier Curves

Using HTML5 canvas, you can create a Bezier curve using the `bezierCurveTo()` method. Bezier curves are represented with the two control points, context points, and an end point.

Code Snippet 7 demonstrates how to create a Bezier curve using HTML5.

**Code Snippet 7**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
```

```
padding: 0px;
  }
  #mCanvas {
   border: 1px solid maroon;
  }
 </style>
 <script>
  window.onload = function() {
   var canvas = document.getElementById("mCanvas");
   var ctext = canvas.getContext("2d");
   ctext.beginPath();
   ctext.moveTo(188, 130);
   ctext.bezierCurveTo(140, 10, 388, 10, 288, 100);
   ctext.lineWidth = 15;
   // line color
   ctext.strokeStyle = "purple";
   ctext.stroke();
  };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="378" height="200"></canvas>
</body>
</html>
```

In the code, the Bezier curve uses the `bezierCurveTo()` method. This method defines the current context point, two control points, and an end point. The context point uses the `moveTo()` method. The first portion of the curve is tangential to the imaginary line defined in the context point and first control point. The second portion of the curve is tangential to the imaginary line which is defined by the second control point and the ending point.

Figure 19.7 displays a Bezier curve in canvas.

Figure 19.7: Bezier Curve in Canvas

### ➔ Quadratic Curves

HTML5 canvas allows the user to create quadratic curves using the `quadraticCurveTo()` method. Quadratic curves are represented through the context point, an end point, and a control point.

Code Snippet 8 demonstrates how to create a quadratic curve using HTML5.

**Code Snippet 8**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid #9C9898;
   }
```

```
window.onload=function() {
   var canvas=document.getElementById("mCanvas");
   var ctext=canvas.getContext("2d");
   ctext.beginPath();
   ctext.moveTo(178, 150);
   ctext.quadraticCurveTo(220, 0, 320, 150);
   ctext.lineWidth=15;
   // line color
   ctext.strokeStyle="Fuchsia";
   ctext.stroke();
  };
 </script>
 </head>
<body>
 <canvas id="mCanvas" width="378" height="200"></canvas>
 </body>
</html>
```

In the code, the control point defines the curve of the quadratic by two tangential lines that are connected to both the context point and the end point. The context point is represented using the `moveTo()` method. This method moves the control point from the context point and the end point to create a sharper curve. It also moves the control point close to the context point and end point to create broad curves.

Figure 19.8 displays a quadratic curve in a canvas.

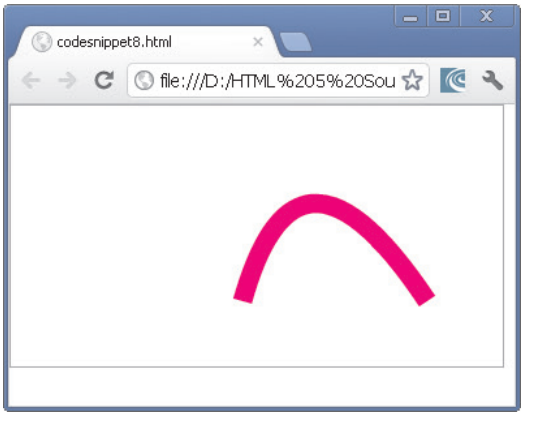



**Figure 19.8: Quadratic Curve in a Canvas**

## *19.5 Working with Images*

In HTML5, the user can draw image objects on canvas using the `drawImage()` method. The `drawImage()` method can also draw parts of an image and increase or reduce the size of the image. This method accepts nine parameters, depending on editing that is required on the image. The image object can be a video, an image, or another canvas element.

Code Snippet 9 demonstrates how to create an image using HTML5.

**Code Snippet 9**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid #9C9898;
   }
</style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("mCanvas");
    var ctext = canvas.getContext("2d");
    var imgObj = new Image();
    imgObj.onload = function() {
    ctext.drawImage(imgObj, 69, 50);
    };
imgObj.src = "bird.jpg";
   };
  </script>
</head>
```

```
<body>
 <canvas id="mCanvas" width="368" height="300"></canvas>
 </body>
</html>
```

In the code, the `onload` property is used. The source of the object is defined by using the `src` property. The image has to be loaded first and then instantiated with the `drawImage()` method. This method takes image object as the parameter with the `x` and `y` coordinates of the image.

Figure 19.9 displays an image drawn on a HTML5 canvas.

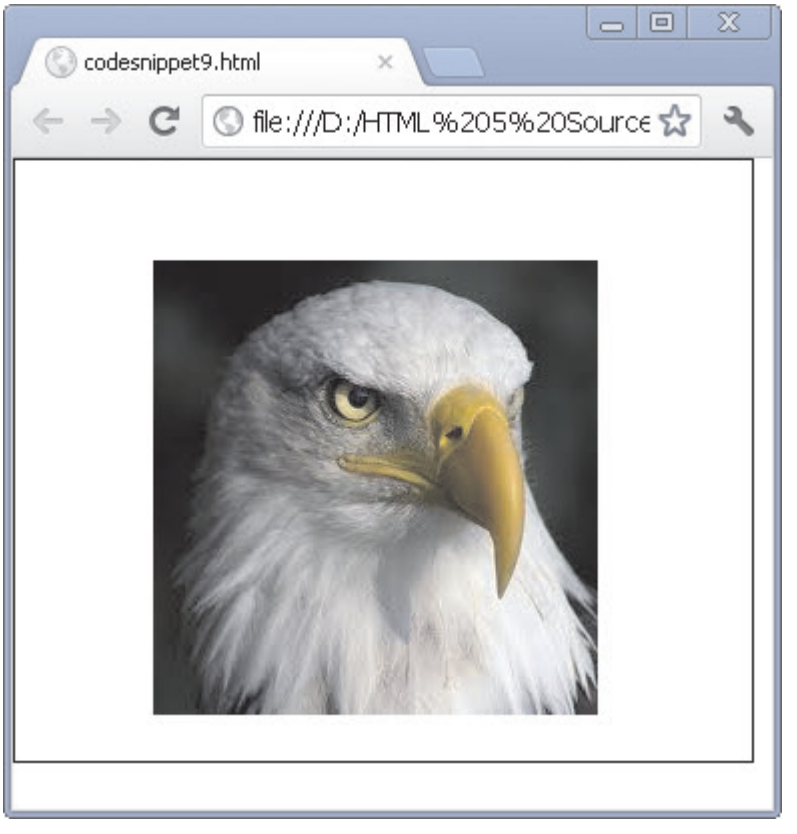



**Figure 19.9: Image Drawn on a HTML5 Canvas**

You can also set the size of the image by adding two parameters `width` and `height` to the `drawImage()` method.

Code Snippet 10 demonstrates how to resize images with `height` and `width` attributes using HTML5.

**Code Snippet 10**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid black;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("mCanvas");
    var ctext = canvas.getContext("2d");
    var x = 69;
    var y = 50;
    var w = 150;
    var h = 137;
    var imgObj = new Image();
    imgObj.onload = function() {
    ctext.drawImage(imgObj, x, y, w, h);
    };
    imgObj.src = "bird.jpg";
   };
  </script>
 </head>
 <body>
```

```
<canvas id="mCanvas" width="278" height="200"></canvas>
 </body>
</html>
```

In the code, the `drawImage()` method accepts two additional parameters `height` and `width` for resizing the image.

Figure 19.10 displays resizing an image in a HTML5 canvas.

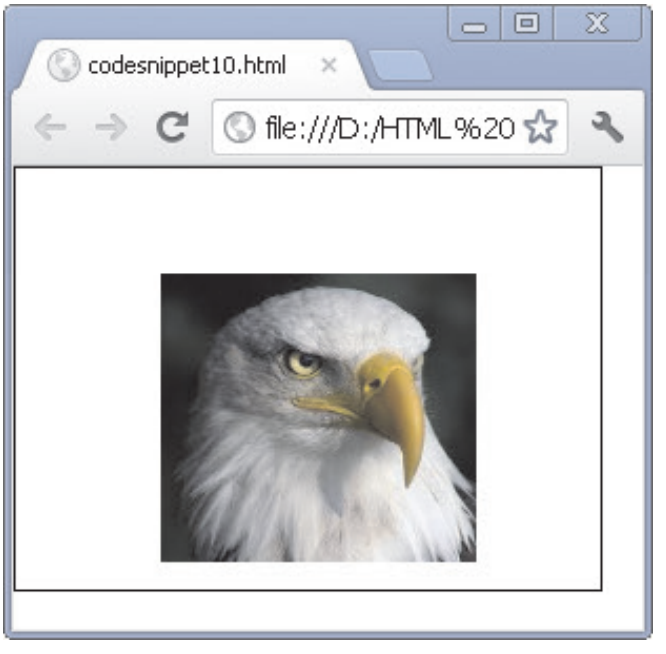



**Figure 19.10: Resizing an Image in a HTML5 Canvas**

## *19.6 Working with Text*

HTML5 canvas enables you to set the font, style, and size of text by using the font properties. The font style can be italic, normal, or bold. For setting the text color, you can use the `fillStyle` property of the canvas.

Code Snippet 11 demonstrates how to set the font, size, style, and color of the text on a HTML5 canvas.

**Code Snippet 11**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
```

```
margin: 0px;
   padding: 0px;
  }
  #mCanvas {
   border: 1px solid blue;
  }
 </style>
 <script>
  window.onload = function() {
   var canvas = document.getElementById("mCanvas");
   var ctext = canvas.getContext("2d");
   ctext.font = "italic 30pt Calibri";
      ctext.fillStyle = "MediumVioletRed";
   ctext.fillText("Welcome to HTML5!", 40, 100);
      };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="380" height="170"></canvas>
</body>
</html>
```

In the code, the font text is specified as `Calibri`, style as `italic`, and size is set to `30pt`. The `fillStyle` property specifies the text color and the `fillText` property is used to set the text on the canvas.

Figure 19.11 displays the working with text in a HTML5 canvas.

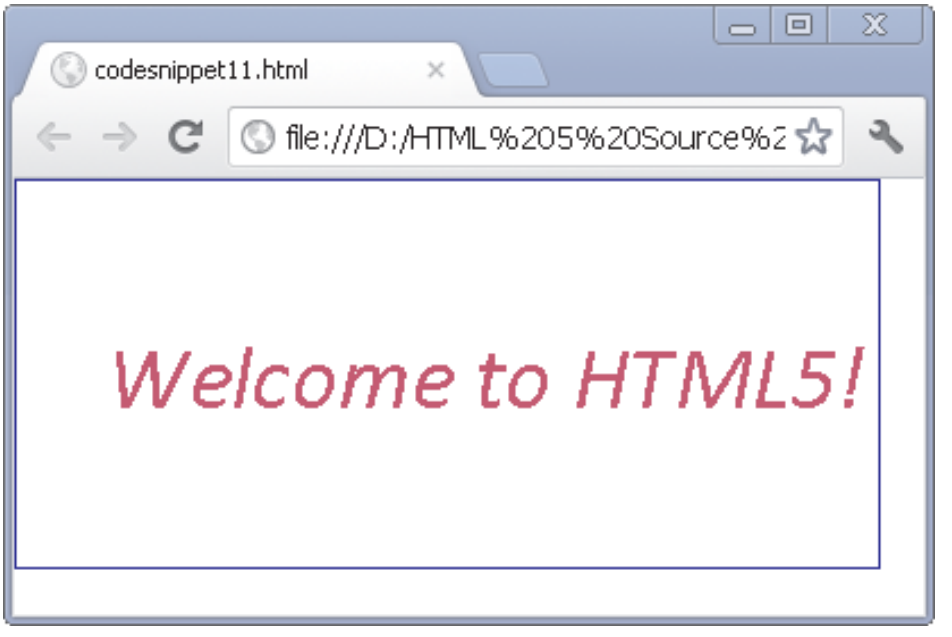



**Figure 19.11: Working with Text in a HTML5 Canvas**

In HTML5 canvas, the user can set the stroke color by using the `strokeText()` method and `strokeStyle` property of the canvas context.

Code Snippet 12 demonstrates the use of stroke text in HTML5 canvas.

**Code Snippet 12**:

```
<!DOCTYPE HTML>
 <html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid black;
 }
  </style>
  <script>
 window.onload = function() {
```

```
var canvas = document.getElementById("mCanvas");
   var ctext = canvas.getContext("2d");
   var x = 80;
   var y = 110;
   ctext.font = "40pt Calibri";
   ctext.lineWidth = 2;
   // stroke color
   ctext.strokeStyle = "Brown";
   ctext.strokeText("HTML5", x, y);
  };
 </script>
 </head>
 <body>
  <canvas id="mCanvas" width="360" height="200"></canvas>
 </body>
</html>
```

In the code, the stroke color is set by using the `strokeStyle` property and the `strokeText()` method.

Figure 19.12 displays the stroke text in HTML5 canvas.

**Figure 19.12: Stroke Text in HTML5 Canvas**

## *19.7 Using Transparency for Text in Canvas*

There are two ways to set the transparency for the text and shapes. The first method is to use the `strokeStyle` and `fillStyle` by using the `rgb` function. The second method is to use `globalAlpha` drawing state property, which can be applied universally. The `globalAlpha` property is a value that ranges between 0 (fully transparent) and 1 (fully opaque).

Code Snippet 13 demonstrates the use of `globalAlpha` property.

**Code Snippet 13**:

```
<!DOCTYPE HTML>
<html>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #mCanvas {
    border: 1px solid black;
   }
  </style>
<script>
  window.onload = function() {
   var canvas = document.getElementById("mCanvas");
   var ctext = canvas.getContext("2d");

    ctext.fillStyle = "Indigo";
    ctext.strokeStyle ="black";
    ctext.lineWidth=2;
    ctext.font = "italic 30pt Calibri";
    ctext.fillText("HTML5", 40, 100);
    ctext.strokeText("HTML5", 40, 100);
```

```
        ctext.fillStyle="blue";
        ctext.globalAlpha=0.5;
        ctext.fillRect(100, 10, 150, 100);
    };
 </script>
</head>
<body>
 <canvas id="mCanvas" width="350" height="170"></canvas>
</body>
</html>
```

In the code, the `fillStyle` and `strokeStyle` is used to color the text. The 'HTML5' text `lineWidth` is specified as `2` and the font-family is set to `Calibri` with `italic` style and font-size to `30pt`. The `fillText` property fills the color and `strokeText` property applies the stroke color to the HTML5 text. The `fillStyle` is set to blue and `globalAlpha` property is set to `0.5`. The `fillRect(100, 10, 150, 100)` specifies the `x`, `y`, `height`, and `width` of the rectangle.

Figure 19.13 displays the transparency in text.

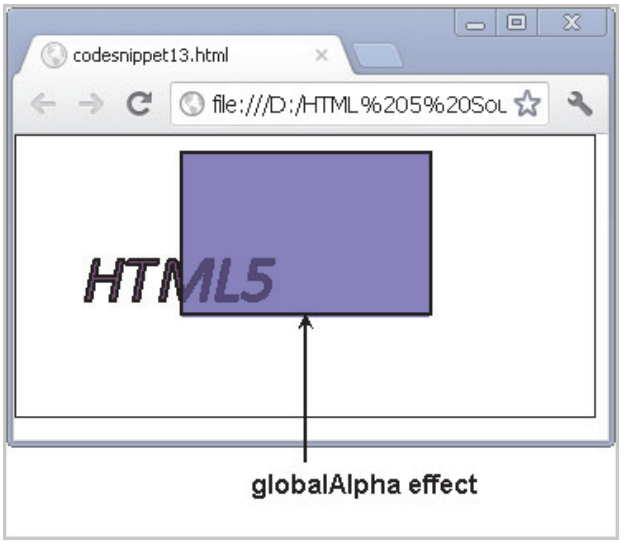



**Figure 19.13: Transparency in Text**

## *19.8 Using Events with jQuery*

jQuery also offers different events to deal with common interactions when the user moves the mouse or switch between two actions while clicking.

The following are the events:

➔ **hover() event**

The mouseenter and mouseleave are the two events often used together. For example, when a user moves a mouse over a menu, a tooltip appears and when the user moves the mouse off the menu, the tooltip disappears. Combining these two events is very common, therefore, jQuery provides a `hover()` function that accepts two parameters. The first parameter executes when the mouse moves over the element and the second function executes when the mouse moves away from the element.

Code Snippet 14 demonstrates the hover event.

**Code Snippet 14**:

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery-1.7.2.min.js"></script>
<script>
$(document).ready(function(){
$("p").hover(function(){
$("p").css("background-color","red");
},function(){
$("p").css("background-color","maroon");
});
});
</script>
</head>
<body>
<p>Hover the mouse on this line.</p>
</body>
</html>
```

In the code, the `hover()` method is used. When the mouse is placed on the text, then the background color changes to `red`. Similarly, when the user moves the mouse outside the text, the `background-color` changes to `maroon`.

Figure 19.14 displays the mouseenter effect.

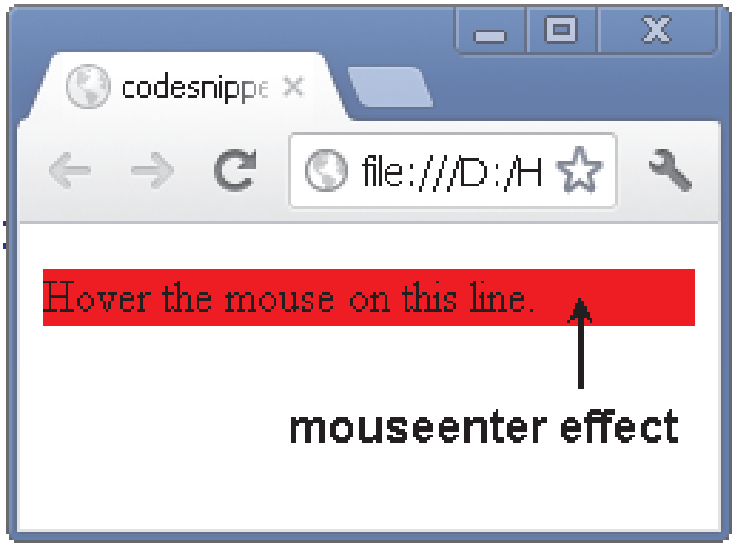



Figure 19.14: Mouseenter Effect

Figure 19.15 displays the mouseleave effect.

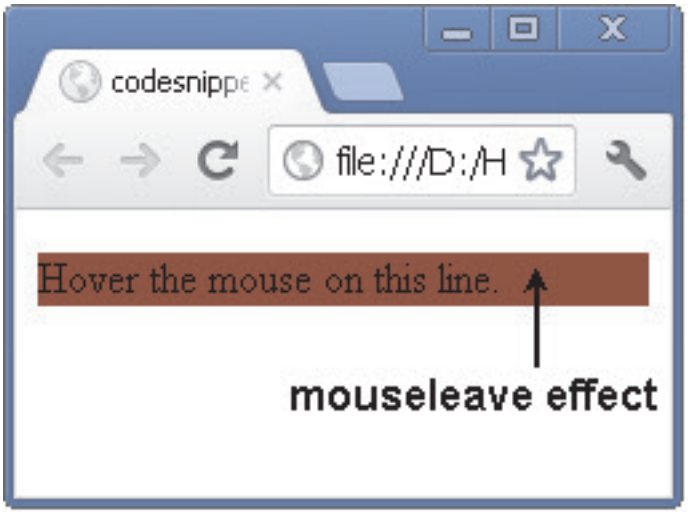



Figure 19.15: Mouseleave Effect

### ➔ The toggle() event

The `toggle()` event works in a similar manner as that of the `hover()` event, except that it responds to mouse clicks. The `toggle()` function accepts more than two functions as arguments. For example, you want to perform some action on the first click, another action on the second click, and one more action on the third click. All the functions passed to the `toggle()` event will react to its corresponding click action.

Code Snippet 15 demonstrates the toggle event.

**Code Snippet 15**:

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery-1.7.2.min.js"></script>
<script>
$(document).ready(function(){
$("p").toggle(function(){
$("body").css("background-color","blue");},
function(){
$("body").css("background-color","pink");},
function(){
$("body").css("background-color","grey");}
);
});
</script>
</head>
<body>
<p>Click to change the colors.</p>
</body>
</html>
```

In the code, the `toggle()` method is used. When the user clicks the text then the `background-color` of the document is changed to blue, pink, and grey respectively.

Figure 19.16 displays the toggle effect to blue.

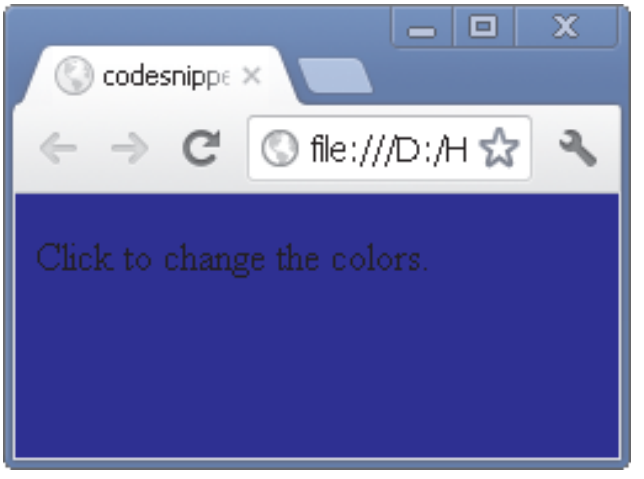



**Figure 19.16: Toggle Effect to Blue**

Click the text and it will change the `background-color` to pink.

Figure 19.17 displays the toggle effect to pink.

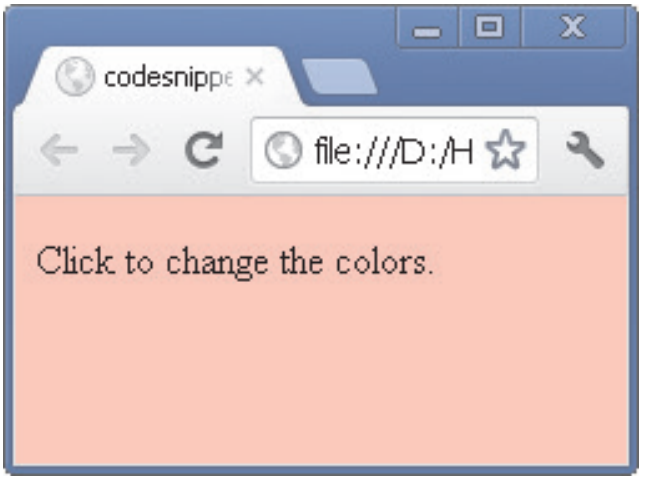



**Figure 19.17: Toggle Effect to Pink**

Click the text and it will change the `background-color` to grey.

Figure 19.18 displays the toggle effect to grey.

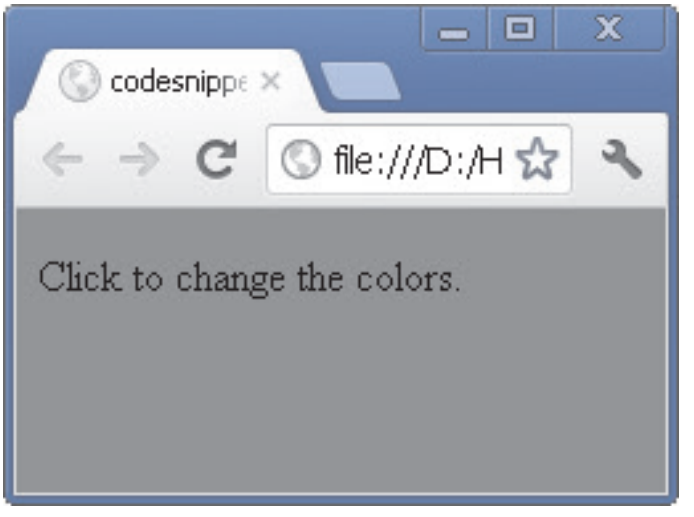



**Figure 19.18: Toggle Effect to Grey**

## *19.9 Inclusion of External Content in Web Pages*

HTML5 introduces the `<eventsource>` tag that allows the user to push external content in the Web page. This model is referred to as push model. Since the `<eventsource>` tag is not supported in many browsers, users make use of the `<embed>` tag for this purpose. The `<embed>` tag is a new element in HTML5 and it is represented as a container for an interactive content or an external application. The `<embed>` tag is often used to add elements such as image, audio, or video on a Web page.

Code Snippet 16 demonstrates the use of the `<embed>` tag.

**Code Snippet 16**:

```
<embed src="mymovie.mp3" />
```

In the code, the `src` attribute specifies the path of an external file to embed.

## *19.10 Check Your Progress*

1. The _____ element used in HTML5 is not only limited to draw rectangles on the Web sites, but also allows the user to draw graphics using JavaScript.

| | | | |
|---|---|---|---|
| (A) | Canvas | (C) | Rectangle |
| (B) | Line | (D) | Arc |

2. The _____ property can be a gradient, pattern, or a CSS color.

| | | | |
|---|---|---|---|
| (A) | fillText | (C) | strokeStyle |
| (B) | strokeText | (D) | fillStyle |

3. Which of the following methods is used to draw a circle in HTML5 canvas?

| | | | |
|---|---|---|---|
| (A) | `line()` | (C) | `arc()` |
| (B) | `stroke()` | (D) | `rect()` |

4. Which of the following events respond to mouse clicks?

| | | | |
|---|---|---|---|
| (A) | `hover()` | (C) | `toggle()` |
| (B) | `pressed()` | (D) | `changed()` |

5. Which of the following methods are used to draw an image object on canvas?

| | | | |
|---|---|---|---|
| (A) | `drawImage()` | (C) | `hover()` |
| (B) | `draw()` | (D) | `onload()` |

## *19.10.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | D |
| 3. | B |
| 4. | C |
| 5. | A |

# Summary

- The <canvas> element is a drawing area where the user can draw graphics, use images, add animations, and also add text for enhancing the user experience on Web pages.
- To create a line, on a canvas one can use the stroke(), beginPath(), lineTo(), and moveTo() methods.
- Arcs are represented using a start angle, an end angle, a radius, a center point, and the drawing direction (anticlockwise or clockwise).
- With HTML5 canvas, the user can create a rectangle using the rect() method.
- Bezier curves are represented with the two control points, context points, and an end point.
- HTML5 canvas allows the user to create quadratic curves using the quadraticCurveTo() method.
- HTML5 canvas enables the user to draw image object on canvas using the drawImage() method.

# *Try it Yourself*

1. Jack is developing a Web site for his company named Decant Technologies, headquartered at Germany. The company Web site was developed on HTML 4. Now, they have decided to upgrade the Web site using HTML5. The Web site deals with different types of software products such as educational, applications, games, computer accessories, anti-virus, and many more. Recently, the company has launched some new software for kid's education and Jack has to add some shapes, images, and text to the new Web pages. Jack has decided to use the canvas element. Help him to develop the application.

2. Robin is creating a Web site for his company named Maxim Technologies, headquartered at California. He wants to create an HTML5 Web site for his company. The company Web site deals with different types of mobile products such as laptops, smartphones, ipads, tablets, and notebooks. Robin wants to develop their company Web site home page and add the details about the company. He also wants to add different types of images, text, videos, and arrange the content in an organized manner. He has decided to use canvas for the home page. Help him to develop the application.

# Onlinevarsity

# ON ALL DEVICES

# Session - 20

# HTML5 Web Storage

Welcome to the Session, **HTML5 Web Storage**.

This session explains about the Web storage API that provides functionality for storing data on the client-side. The session also explains the Indexed DB API which allows hosting of Web databases locally within the user's browser.

Further, the session explains the differences between the native apps and HTML5 apps. Finally, it explores the process of converting HTML5 apps to native apps.

In this Session, you will learn to:

- Explain Web storage in HTML5
- Explain session storage
- Explain local storage
- Explain the Indexed DB API
- Describe a native app
- Explain the difference between native apps and HTML5 apps
- Describe the advantages of native and HTML5 apps
- List the steps to convert HTML5 apps to native apps

## *20.1 Introduction*

Consider an e-mail client, such as Gmail. To log in to your mail account in Gmail, you need to enter your username and password.

Figure 20.1 shows the login screen of Gmail.

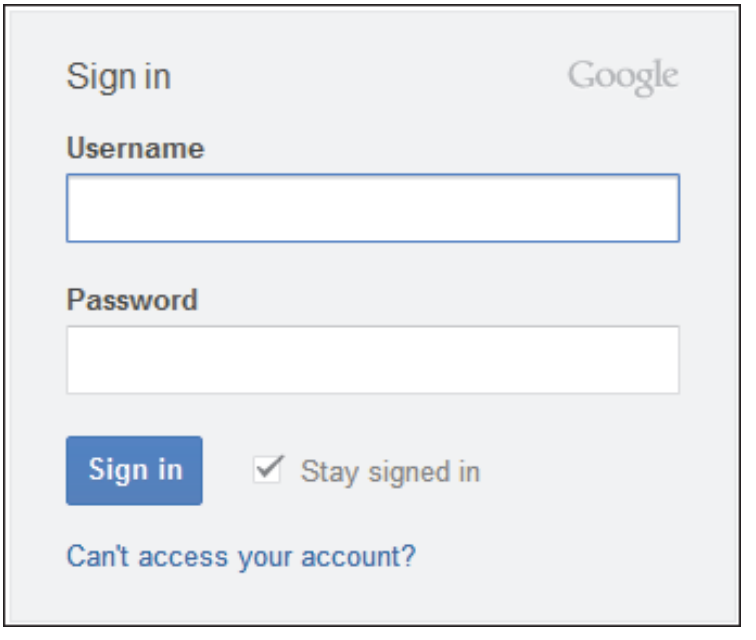



**Figure 20.1: Login Screen of Gmail**

As shown in figure 20.1, the **Stay signed in** check box, specifies that the login details, such as Username and Password, should be remembered by the computer.

Traditionally, over the last few decades, Web applications have been using cookies to store small amounts of information on a user's computer. A cookie is a file that stores user-related information and may either be temporary or permanent. Thus, in this case, a cookie can be created for login details which can be saved for a specified period on a user's computer.

However, the drawbacks of cookies are as follows:

- Cookies slow down the performance of Web application, as they are included with every HTTP request.
- Cookies cannot be considered as safe means for transmission of sensitive data.
- Cookies cannot store large amount of information, as they have a limitation of size of 4 KB.

To overcome these drawbacks and offer a solution to store data on the client-side, W3C has designed a specification named Web Storage API.

The Web storage provides the functionality using which data can be stored on the client-side for a session or beyond the session.

## *20.2 Web Storage in HTML5*

Web storage is a W3C specification. It provides functionality for storage of data on the client-side that is on user's machine. This data can cater for both temporary as well as permanent needs. Certain browsers also refer to it as 'DOM Storage'. The advantage of such storage is that it offers more control than traditional cookies, and is easy to work with.

Web storage was originally a part of the HTML5 specification, but now it is present in a separate specification. It enables to store a maximum of 5 MB of information per domain.

HTML5 Web applications make use of Web storage to implement client-side persistent storage.

### *20.2.1 Web Storage versus Cookies*

There are some key differences between cookies and Web storage that are as follows:

- Cookies are meant to be read on the server-side, whereas Web storage is available only on the client-side. The server cannot read or write to it directly.
- Cookies are sent along with each HTTP request to the server, whereas Web storage data is not carried over to the server.
- Cookies can result in bandwidth overhead and thus lead to high costs, as they are sent with each HTTP request. The Web storage is stored on the user's hard drive, so it costs nothing to use.
- As mentioned earlier, with cookies, the information data that could be stored is 4 KB, whereas with Web storage, a large amount of data can be stored upto 5 MB.

### *20.2.2 Browser-specific Web Storage*

Web storage is browser-specific. If a user visits a site in Google Chrome, any data will be stored to Google Chrome's Web storage store. Similarly, if the user revisits that same site in Firefox, the data saved earlier through Google Chrome will be unavailable. The location where the Web storage data is stored depends on the browser. Each browser's storage is separate and independent, even if it is present on the same machine.

HTML5 Web storage is implemented natively in most Web browsers, so one can use it even when a third-party browser plug-in is not available.

Table 20.1 lists the support of various browsers for HTML5 Web storage.

| Browser | Version |
|---|---|
| **IE** | 8.0+ |
| **Firefox** | 3.6+ |
| **Safari** | 4.0+ |
| **Chrome** | 5.0+ |
| **Opera** | 10.5+ |

**Table 20.1: HTML5 Web Storage Support**

## *20.3 Exploring Web Storage*

The two types of HTML5 Web storage are namely, session storage and local storage. Both session and local storage enable to store around 5 MB of data per domain. Before going into the details of these types, you need to determine, if the current version of your browser has support for HTML5 Web storage.

To check for browser support of HTML5 Web storage, a property named `localStorage` or `sessionStorage` is available as a global variable for the `window` object. If there is no support, the `localStorage` or `sessionStorage` property will be undefined.

Code Snippet 1 demonstrates the script to check the support for HTML5 Web storage in the browser.

**Code Snippet 1**:

```
<!DOCTYPE html>
<html>
<head>
 <title>Support for Web Storage</title>
 <script>
  function checkSupport()
  {
   if (('sessionStorage' in window) && window['sessionStorage'] !== null)
   {
    alert("Your browser supports Web Storage");
    return;
   }
   alert("Your browser does not support Web Storage");
```

```
}
 </script>
</head>
<body onload="checkSupport();">
</body>
</html>
```

Here, in the code, the `if` statement checks whether a property named `sessionStorage` exists in the global `window` object. If the property exists, it means that session storage is supported and an appropriate message is displayed to indicate the same. If however, the property does not exists, it means session storage is not supported on that browser, and an appropriate message is displayed to indicate the same.

Figure 20.2 shows the Web storage support.

**Figure 20.2: Web Storage Support**

## *20.3.1 Session Storage*

Session storage keeps track of data specific to one window or tab and discards it as soon the user closes the tab (or window) that he/she was working with. Thus, even if you are visiting the same site in two different windows, each window will have its own individual session storage object. This means that each window contains separate session storage object with distinct data. Session storage lasts for the entire duration of the session and hence, is not persistent.

Session storage makes use of named key/value pairs. The data is stored using the named key, whereas the data is retrieved by referring to that key. Both the key-value pairs are enclosed within double quotes.

The key is a string, whereas the value stored in the key can be of any type of data, such as string, boolean, integer, or float. Regardless of the type of data that is stored, it is actually stored internally as a string.

Therefore, storing and retrieving data of other types requires the use of functions to convert them into the appropriate data types.

For example, the function, `parseInt()` is used to convert data into an appropriate JavaScript data type.

Table 20.2 lists some examples of named key/value pairs belonging to various data types.

| Key | Value |
|---|---|
| Name | Sarah |
| book | C Programming |
| Email | info@me.com |
| car | Toyota Innova |
| age | 28 |
| uservalid | true |

**Table 20.2: Key/Value Pairs**

There are several operations that can be performed with the `sessionStorage` object. These operations are described as follows:

➔ **Storing and retrieving data**

The `setItem()` and `getItem()` methods are used to store and retrieve data from session storage respectively.

The syntax to use the `setItem()` and `getItem()` methods is as follows:

**Syntax**:

- **To assign data**

```
sessionStorage.setItem(key, value);
```

where,

`key`: Is the named key to refer to the data

`value`: Is the data to be stored

- **To retrieve data**

```
var item = sessionStorage.getItem(key);
```

where,

`item`: Is the variable into which the data will be saved

`key`: Is the named key to refer to the data

Code Snippet 2 demonstrates how to set and retrieve a name using `sessionStorage` object.

**Code Snippet 2**:

```
<!DOCTYPE html>
<html>
<head>
 <title>Working with Session Storage</title>
 <script>
  function testStorage()
  {
   if (('sessionStorage' in window) && window['sessionStorage'] !== null)
   {
    sessionStorage.setItem('name', 'Sarah');
    alert('The name is: ' + sessionStorage.getItem('name'));
   }
  }
  </script>
</head>
<body onload="testStorage();">
</body>
</html>
```

The code stores the string literal 'Sarah' in the key, **`name`**. This is done using the `setItem()` method. Then, the string literal is retrieved and displayed as an alert using the `alert()` method on the browser page. To retrieve the string literal, the `getItem()` method has been used in the code.

Figure 20.3 shows the output for storing and retrieving name on the Web page using `sessionStorage` object.

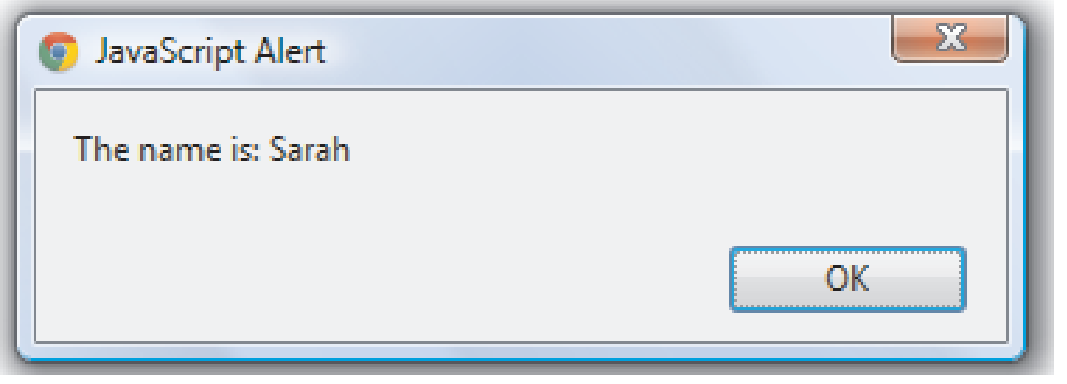



**Figure 20.3: Output – Storing and Retrieving Using Session Storage**

**Note** - Web Storage API is supported in the Internet Explorer version 9 onwards.

Key names need not necessarily be given as strings. Even, a number can be specified as a key name, though internally while storing, it will be converted to a string.

Code Snippet 3 shows the script that uses number as the key.

**Code Snippet 3**:

```
<script>
 function testStorage()
 {
  if (('sessionStorage' in window) && window['sessionStorage'] !== null)
   {
    sessionStorage.setItem(6, 'The book was wonderful');
    var feedback = sessionStorage.getItem(6);
    alert(feedback);
    }
 }
</script>
```

Here, in the code, the key is set to a numeric value **6**, but that does not mean that it is the sixth key. The name of the key is defined as **6** that is stored internally in the browser. The `getItem()` method is used to retrieve the value stored with that key. Finally, the value having the key as **6** is displayed in the alert window.

Figure 20.4 shows the output that retrieves value stored at the key, **6**.

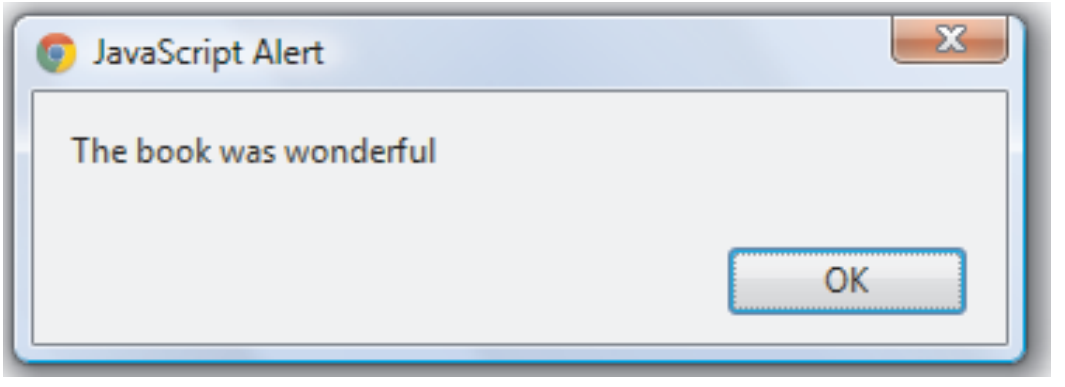



**Figure 20.4: Output – Retrieves Value Stored at Key 6**

Similar to other JavaScript objects, the `sessionStorage` object can be treated as an associative array. Instead of using the `getItem()` and `setItem()` methods, an array subscript can be used to retrieve and assign the data. For example, `sessionStorage.setItem('name','John');` can be written as `sessionStorage['name'] ='John';`

Similarly, `alert (sessionStorage.getItem('name'));` can be written as `alert (sessionStorage['name']);`

As mentioned earlier, it is also possible to store non-string values into session storage.

Code Snippet 4 demonstrates the storage of an integer value for age and is later retrieved by using `parseInt()` method along with the `getItem()` method.

**Code Snippet 4**:

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title> Session Storage </title>
 <script>
  function store() {
   if (('sessionStorage' in window) && window['sessionStorage'] !== null) {
      var name = document.getElementById('name').value;
      sessionStorage.setItem('username', name);
      var data = sessionStorage.getItem('username');
```

```
    sessionStorage.setItem('age',
    document.getElementById('age').value);
    var agevalue = parseInt(sessionStorage.getItem('age'));
     alert(data + 'is ' + agevalue + 'years old');
    }
   }
 </script>
</head>
<body>
  <form name="myform">
  <label> Enter Name: </label>
   <input type="text" id="name">
  <br />
  <label> Enter Age: </label>
  <input type="text" id="age">
  <br/>
  <input type="button" value="Submit" onclick="store()"/>
 </form>
 </body>
</html>
```

In the code, the `parseInt()` method converts the specified argument to an integer format.

Figure 20.5 shows the output for storing and retrieving the form data having integer values.

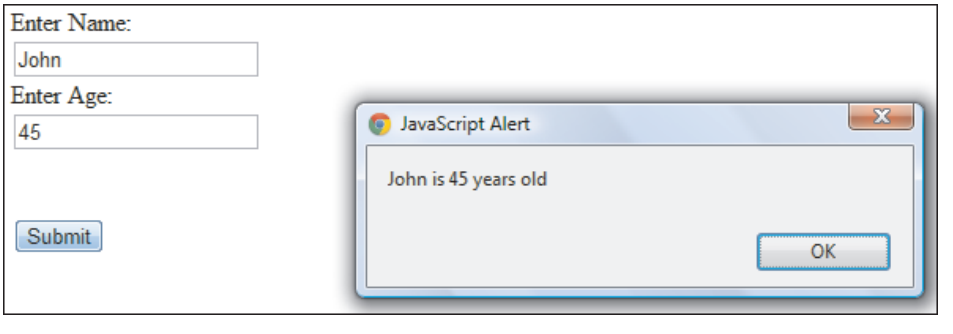



**Figure 20.5: Output – Form Data Having Integer Values**

➔ **Removing and Clearing Data**

It is also possible to remove and clear data from the session storage. The `removeItem()` method is used to remove a particular item from the list.

The syntax for the `removeItem()` method is as follows:

**Syntax**:

```
sessionStorage.removeItem(key);
```

where,

`key`: Is the named key for the data

To remove the value associated with the key, **`username`** set in Code Snippet 4, the statement must be specified as follows:

```
sessionStorage.removeItem('username');
```

Similarly, in order to clear all items present in the session storage, use the `clear()` method as shown:

```
sessionStorage.clear();
```

Also, the `length` attribute determines the number of key/value pairs present in the storage.

```
var itemcount = sessionStorage.length;
```

## *20.4 Local Storage*

Unlike session storage, local storage enables to save data for longer periods on the user's computer, through the browser. The data is persistent and can be retrieved when a user visits the site again. In other words, local storage is used, if data needs to be stored for more than a single session. A simple scenario would be to count the number of times a person has visited a Web page.

In terms of methods, local storage works in a similar fashion as session storage. It uses the same functions, such as `setItem()`, `getItem()`, `removeItem()`, and `clear()`.

Code Snippet 5 demonstrates the use of local storage to store the value of **`username`** field and later, retrieve the value in another Web page.

**Code Snippet 5**:

```
<!DOCTYPE html>
<head>
 <title> Local Storage </title>
     <script>
     function store()
      {
```

```
   if (('localStorage' in window) && window['localStorage'] !== null)
   {
     var username = document.getElementById('username').value;
     localStorage.setItem('username', username);
                 }
      else
      {
          alert ('your browser does not support storage');
       }
  }
  function cancel_store()
  {
   if (('localStorage' in window) && window['localStorage'] !== null)
   {
     localStorage.removeItem('username');
       }
   else
       {
       alert ('your browser does not support storage');
       }
  }
      </script>
</head>
 <body>
 <form method="get" action="success.html">
 Username: <input type="text" id="username" value="" size="20"
onblur="store()"/>
<input type="submit" value="Submit"/>
<input type="reset" Value="Cancel" onclick="cancel_store()"/>
```

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

Here, in the code, the support of **`localStorage`** object is checked in the current browser. If it is supported, then the contents of the **`username`** box are retrieved and stored in a variable named **`username`**. Then, the content of this variable is assigned to the local storage object with the key set as **`username`**. If **`localStorage`** object is not supported, an appropriate message is displayed in the alert window.

Also, the function **`cancel_store()`** is invoked when the user clicks **Cancel**. In the **`cancel_store()`** function, the `removeItem()` method removes the specified key and its value from local storage.

When the **Submit** button is clicked, the user is redirected to the Web page, **`success.html`**, which displays the value stored with the key, **`username`**.

Code Snippet 6 shows the **`success.html`** page that retrieves value from the local storage and displays it in the browser.

**Code Snippet 6**:

```
<!DOCTYPE html>
 <head>
  <title> Local Storage </title>
      <script>
      function print()
      {
        var username = localStorage.getItem('username');
           document.getElementById('lblMsg').innerHTML = 'Username: is <b>'+
                      username+'</b>';
  }
      </script>
      </head>

      <body onload="print()">
   <label id="lblMsg"></label><br>
</body>
</html>
```

Here, in the code, `getItem()` method of local storage retrieves the value from the **`username`** key and stores in the variable **`username`**. Then, the value of the variable **`username`** is displayed in the `<label>` tag.

Figure 20.6 shows the output of the Web page with the user input, **`John`**.

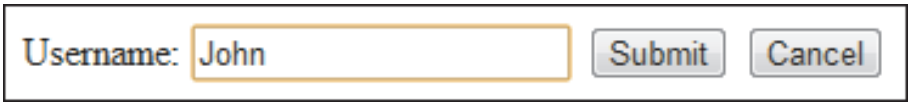



**Figure 20.6: Output – Web page with User Input**

Figure 20.7 shows the output of **`success.html`** Web page that displays the value for **`username`** key stored in the local storage.

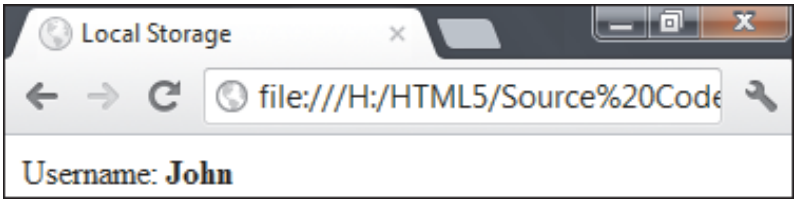



**Figure 20.7: Output – Displaying Value from Local Storage**

## *20.5 Indexed Database API*

A database is an organized collection of data. Databases, such as relational database stores the data in the form of tables. A table comprises rows and columns that are used to store data. The representation of data from a table is in the form of records.

HTML5 has introduced a new Web Storage API which can host Web databases locally within the user browser. However, Web databases are not like relational databases in terms of functionality.

Indexed Database API is a specification also known as IndexedDB. It is basically an object store that can be used to store and manipulate data on the client-side. The object store is the primary storage mechanism that stores the object in the database managed locally within the browser. It enables to create an object store of a particular type in which objects can be persisted using JavaScript. Thus, IndexedDB enables to create Web applications with rich query abilities and which can work both online and offline.

IndexedDB supports two types of API namely, synchronous and asynchronous. The synchronous API can be used with WebWorkers, whereas asynchronous API can be used for Web applications. Currently, no major browsers provide the support for synchronous API.

The IndexedDB API is implemented using `window.indexedDB` object. As the current specification is still in the evolving stage, browsers implement the `IndexedDB` object with their own prefixes. For example, Chrome browser uses the `webkit` prefix, whereas Mozilla supports `–moz` prefix.

Table 20.3 lists the browser support for the IndexedDB API.

| IE | Firefox | Chrome | Safari | Opera | iOS Safari |
|---|---|---|---|---|---|
| 6.0 | 3.6 | - | - | - | 3.2 |
| 7.0 | 8.0moz | - | - | - | 4.0-4.1 |
| 8.0 | 9.0moz | 16.0webkit | 5.0 | - | 4.2-4.3 |
| 9.0 | 10.0moz | 17.0webkit | 5.1 | 11.6 | 5.0 |
| 10.0ms | 11.0moz | 18.0webkit | 6.0 | 12.0 | - |
| - | 12.0moz | 19.0webkit | - | - | - |

**Table 20.3: Browser Support for IndexedDB API**

## *20.5.1 IndexedDB API*

Some of the basic constructs of IndexedDB API are as follows:

- **Database** - The IndexedDB database comprises more than one object store. Each database contains a name that identifies the origin of the database and a version number which identifies the lifetime of the database.

- **Object Store** - Object store is the main mechanism to store data in a database. They hold the data stored in the database in the form of records.

- **Keys** - Each record stored in the database is identified by a unique key.

- **Values** - Values are the data stored in the records.

- **Key Path** - A key path is a string that defines how the browser should extract key from a value. The key from a value can be extracted either in the object store or index. An empty string, a JavaScript identifier, and so on is some of the valid key paths.

- **Index** - It is used when the data from the object store is retrieved based on some other values other than a key. It is a specialized object store which searches for records in another object store known as referenced object store.

- **Transaction** - Any addition or retrieval of the data in a database is performed by using transaction. In other words, it is a mechanism used for interacting with the database. Each transaction has a mode, scope, and a request list. The mode determines the type of transaction that can be performed, scope identifies the object store on which the transaction will be performed, and finally request list determines the requests that have been made.

- **Requests** - Operations, such as reading or writing on the database is performed using a request. Each request contains attributes, such as flag, source object, result, and error.

- **Cursor** - Cursor is a mechanism used to retrieve multiple records from a database.

- **Key Range** - Records from the object stores and indexes are retrieved using keys or key ranges. A key range refers to retrieval of data between specified bounds based on the keys.

## *20.5.2 Implementing IndexedDB API*

The steps to implement the IndexedDB API in a Web application are as follows:

1. Open a database
2. Create an object store
3. Start a transaction
4. Perform some database operations, such as add and retrieve
5. Work with the retrieved results

- **Opening a Database**

  Code Snippet 7 shows the code to open a database.

  **Code Snippet 7**:

```
var indexedDB = window.indexedDB || window.webkitIndexedDB ||
window.mozIndexedDB || window.msIndexedDB;


var request = indexedDB.open("CompanyDB", 1);


request.onsuccess = function (event) {
 . . .
};
request.onerror = function (event) {
  console.log("IndexedDB error: " + event.target.errorCode);
};
```

  Here, the code detects the support for IndexedDB API in different browsers and creates the `indexedDB` object. The `indexedDB` object contains a method named `open()` which opens the `CompanyDB` database.

In case, if the database exists, then `open()` method simply opens it, otherwise creates the database.

The `open()` method returns an `IDBRequest` object named `request`. The `request` object provides handlers, such as `success` and `error`. These handlers are invoked depending on the success or failure of opening the database. The `onsuccess()` handler contains an event of type `success` as its argument. Similarly, `onerror()` handler is invoked with an `error` event as its argument.

### ➔ Updating Version of a Database

After the database is opened, it can be structured by providing a version number. This helps to set up the database.

The version number will be specified to a database in the `onsuccess()` function.

Code Snippet 8 shows the code that specifies the version number to the database **`CompanyDB`**.

**Code Snippet 8**:

```
var setVrequest = db.setVersion("1.99");

setVrequest.onsuccess = function(event) {

. . .

}
```

### ➔ Creating the Object Store

The structure of IndexedDB database facilitates the storage of multiple object stores. In other words, there can be more than one object stores in the database. Object store is created using `createObjectStore()` method. The `createObjectStore()` methods accepts two arguments namely, the store name and a `parameter` object. The `parameter` object is used for defining an optional property which is important. In this case, a key path is defined that is used for identifying unique objects in the object store. For example, an employee store contains the `id` property as key path, which will be unique for each object and must be present for each object.

Code Snippet 9 demonstrates the code to create an object store named **`employee`** in the **`CompanyDB`** database.

**Code Snippet 9**:

```
var employeeData = [
     { name: "John Smith", email: "john@company.com" },
     { name: "Jill Patrick", email: "jill@company.com" },
          { name: "Rock Ethan", email: "rock@company.com" },
          { name: "Daniel Andrew", email: "daniel@company.com" }
];
     var objectStore = db.createObjectStore("employee", {
                    keyPath: "id", autoIncrement: true });
for (i in employeeData) {
       objectStore.put(employeeData[i]);
       alert("Record added");
}
```

The code creates an array named `employeeData` containing name and e-mail values. Then, the `createObjectStore()` method creates an employee store with its key path set to `id` attribute. The key path is used with `autoIncrement` option that automatically generates id for each of the objects. All the individual objects in the object store are identified based on the `id`. Finally, the `for..in` loop stores the data in the **`employee`** object store.

### ➔ Creating a Transaction

To perform database operation, such as retrieving data from the object store, IndexedDB provides a `IDBTransaction` object. This object can be created in three mode namely, read-only, read-write, and snapshot. The read-write mode is used to update the object, whereas read-only mode is used for other operations.

Code Snippet 10 demonstrates the code to retrieve data from the **`employee`** object store using the `get()` function of the `transaction` object.

**Code Snippet 10**:

```
var trans = db.transaction(["employee"], IDBTransaction.READ_WRITE).
objectStore("employee");

          var request = trans.get(2);

    request.onerror = function(event) {

      // Handle errors!

   };

   request.onsuccess = function(event) {

     // Do something with the request.result!

    alert("Name: " + request.result.name);

   };
```

In the code, the `transaction()` method accepts two parameters. The second parameter is optional. The first parameter is the list of the object stores that are extended by the `transaction` object. In this case, there is a single object store named **`employee`**, created in the database. The optional second parameter specifies the type of the transaction, that is, read-only, read-write, or snapshot. Here, the transaction type is defined as `IDBTransaction.READ_WRITE`. This type allows reading as well as writing in the database. The **`employee`** object store is retrieved from the `transaction` object on which operations are performed. Here, the `get()` method is invoked on the **`employee`** object store which returns the value against the key path 2. Finally, the result of the `get()` method is stored in the request object on which callback functions, such as `onsuccess` and `onerror` are invoked.

### ➔ Opening a Cursor

Cursor is used to retrieve multiple records from an object store. They can be used when the value of key path is not known. They are part of a transaction and are opened for a particular object store.

Code Snippet 11 demonstrates the code to retrieve multiple records from the **`employee`** object store.

**Code Snippet 11**:

```
store = db.transaction("employee").objectStore("employee");
store.openCursor().onsuccess = function(event) {
 var cursor = event.target.result;
 if (cursor) {
  alert("Name for id " + cursor.key + " is " + cursor.value.name);
  cursor.continue();
 }
};
```

Here, in the code, the `transaction` object is created for the **`employee`** object store. Then, the `openCursor()` function is invoked on the object store. If the cursor is successfully opened, a cursor object is returned which will retrieve the data from the object store.

Code Snippet 12 shows the complete code for opening, creating, and retrieving data from the object store using the IndexedDB API.

**Code Snippet 12**:

```
<!DOCTYPE html>
<html>
 <head>
<title>IndexedDB API</title>
<!--[if IE]>
 <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></
script>
 <![endif]-->
     <script>
     // Detect the support for IndexedDB API
     var indexedDB = window.indexedDB || window.webkitIndexedDB || window.
mozIndexedDB || window.msIndexedDB;
  var IDBTransaction = window.IDBTransaction || window.
                                webkitIDBTransaction;
```

```
        var db;
        var transaction;
        var store;
        var objectStore;
var employeeData = [
     { name: "John Smith", email: "john@company.com" },
     { name: "Jill Patrick", email: "jill@company.com" },
     { name: "Rock Ethan", email: "rock@company.com" },
     { name: "Daniel Andrew", email: "daniel@company.com" }
    ];

function initDb() {
     var request = indexedDB.open("CompanyDB", 1);
request.onsuccess = function (evt) {
             db = request.result;
             var setVrequest = db.setVersion("1.99");
     setVrequest.onsuccess = function(e) {
            if(db.objectStoreNames.contains("employee")) {
                     db.deleteObjectStore("employee");
                    alert ('Existing Employee Object Deleted');
            }
            objectStore = db.createObjectStore("employee", {
            keyPath: "id", autoIncrement: true });
            alert ('Employee Object Created');
            objectStore.createIndex("name", "name", { unique:
                                                    false });
            objectStore.createIndex("email", "email", { unique:
                                                     true });
```

```
        for (i in employeeData) {
                    objectStore.put(employeeData[i]);
                     alert("Record added");
           }
     };
 };
 request.onerror = function (evt) {
      console.log("IndexedDB error: " + evt.target.errorCode);
 };
}

function employee_details()
{
  store = db.transaction("employee").objectStore("employee");
 store.openCursor().onsuccess = function(event) {
   var cursor = event.target.result;
    if (cursor) {
              alert("Name for id " + cursor.key + " is " + cursor.
                                          value.name);
              cursor.continue();
              }
 };
}
function search_employee()
{
   trans = db.transaction(["employee"], IDBTransaction.READ_
                      WRITE).objectStore("employee");
   var request = trans.get(2);
   request.onerror = function(event) { // Handle errors! };
```

```
    request.onsuccess = function(event) {
          // Work with the request.result
           alert("Name: " + request.result.name);
    };
}
window.addEventListener("DOMContentLoaded", initDb, false);
</script>
</head>
 <body>
 <input type="button" value="Print Employees Details" onclick="employee_
details()"/><br/>
 <input type="button" value="Search Employee Based on Key" onclick="search_
employee()"/>
 </body>
</html>
```

The program execution starts with the creation of **`employee`** object store in which records are added.

The alert window displays the message, '`Record added`' while storing the records in the object store. Then, the application allows the user to query the database by clicking the **Print Employee Details** and **Search Employee Based on Key** buttons.

Figure 20.8 shows the output for the **`employee`** object store in the Chrome browser, when a user clicks **Search Employee Based on Key** button.

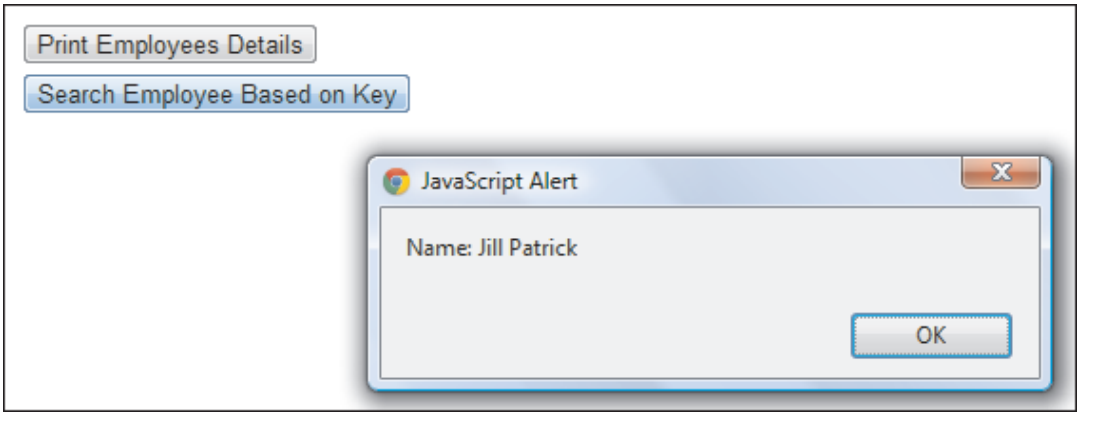



Figure 20.8: Output – Retrieve Employee Object Based on Key

## *20.5.3 Limitations of IndexedDB API*

The IndexedDB API is used for client-side storage of data, but it has some design limitations. These limitations are as follows:

- Internationalized sorting deals with sorting of string data. As the database does not follow any international order for storing data, internationalized sorting is not supported by the API.
- The IndexedDB API does not synchronize client-side database with the server-side databases. If required, then a code needs to be written to perform server-side synchronization.
- The IndexedDB API supports querying the client-side database, but does not support the use of operators, such as `LIKE` that is used by Structured Query Language (SQL). SQL is a query language used by server-side databases to perform operation in the database.

## *20.6 Converting HTML5 Apps to Native Apps*

A native application is also known as native app. It is an application program that is built to be used on a particular device or platform.

A native app, when compared with Web app, is installed on a device and has a faster response, because it has a direct user interface.

Figure 20.9 displays the different platforms that support native apps.

**Figure 20.9: Different Platform Supporting Native Apps**

For example, BlackBerry Messenger (BBM) is a native app available on BlackBerry mobile devices.

## *20.6.1 Differences Between Native Apps and HTML5 Apps*

With HTML5 release, a discussion started about HTML5 which involved whether HTML5 can develop native mobile apps (that is apps designed especially for Blackberry, iPhone, Android, and so on).

HTML5 is the latest version of HTML language providing a simple building block for Web pages. This is the first version of markup language that supports the use of multimedia without using any additional plug-in and is supported by many devices and computer systems.

HTML5 Web apps are accessible and used on any devices through Web browser similar to the mobile Web site. The Web apps have the ability of offline access which means that the user need not have a network connection.

Table 20.4 lists the differences between the native apps and HTML5 apps.

| Native Apps | HTML5 Apps |
|---|---|
| Native Apps runs on iOS and Android devices that can be downloaded or purchased from the online app stores. | HTML5 Apps runs on a Web server, usually in a Web browser. |
| Native Apps use programming language, such as Java for Android devices and Objective C for iOS devices. | Web developers use HTML, JavaScript, and CSS. They need to acquire the skills of Java and objective C for writing native applications. |

**Table 20.4: Difference Between Native Apps and HTML5 Apps**

## *20.6.2 Advantages of HTML5 Apps*

The main advantage of using HTML5 is to create applications that execute on a wide range of devices easily. App development on HTML5 is cheaper as compared to native app development. Developers do not have to learn any new programming language and the development becomes much easier.

There are many reasons to develop HTML5 applications rather than native applications. Some of the reasons are as follows:

➔ **Users cannot identify the differences**

There are many instances where the users cannot identify whether they are working on a hybrid HTML5-native application or a fully native application or an HTML5 application.

- **Users adjust styles for devices**

  HTML5 apps can be viewed on any devices that contains Web browser. Users can also use CSS3 for applying styles to change the look according to their requirements. HTML5 apps look similar to Web pages by using the same code.

- **Upcoming functionalities**

  HTML5 does not support all features on a device, but it is coming up with new functionalities. There are several APIs that exist for working on local databases, Web storage, drag-and-drop, offline Web pages, and many more. New APIs are being planned in future to provide Web pages access to the software and hardware of a device.

- **Improving performance**

  Many developers learn new methods to improve the performance of Web pages. These same methods are useful to mobile HTML5 apps. There are many apps, such as timer, mail, databases, and news apps that need not be faster.

- **Independent device**

  HTML5 apps work on mobile devices as well as Web browsers, as it is important for development purpose. There are millions of laptop and desktop users than mobile users. If the developers want that their application to be used by a large number of users, then they should design and develop the application for both mobile users as well as desktop users.

- **Developers are not locked in app stores**

  There are a number of app stores existing for Android and BlackBerry. If developers wish to sell the apps, then they are required to provide them into as many stores or marketplaces as possible. By using HTML5, developers are not restricted to an app store. Instead, they can create applications and sell them like any other Web application.

HTML5 app acts a substitute to native apps, though in several conditions, native apps can be used for better purpose. Developers can create hybrid applications that are a combination of native and HTML apps.

## *20.6.3 Advantages of Native Apps*

The major advantage of native apps over HTML5 apps is that they are faster than HTML5 apps. Similar to normal Web pages, HTML5 apps are slow, because these apps work on HTTP that uses a request/response cycle mechanism. When an HTTP request is made, it takes more time for the applications to execute as it has to wait for the request to go and return back with a response.

Native apps provide many more benefits over HTML5 apps. These benefits are as follows:

➔ **Providing access to device hardware**

Many mobile devices contain hardware, such as accelerometer, GPS, a camera, and so on. The GPS is accessible through Geolocation API. At present, there are no APIs available for accelerometers, cameras, or any other device hardware for HTML5 apps.

➔ **Uploading Files**

Native apps can access the file system in Android and some files in iOS. However, the HTML5 file API does not work on Android or iOS.

➔ **Push Notifications**

The push notifications are sent always with an open Internet Protocol (IP) connection to applications on the iOS device. Android also has a same feature named Cloud to Device Messaging.

➔ **Accessing device files**

Native apps communicate with files on devices, such as contacts and photos. However, these files cannot be seen from HTML5 apps.

➔ **Superior graphics than HTML5**

HTML5 has a `<canvas>` element, but it will not be able to create a full 3D experience.

➔ **Offline access**

HTML5 provides access to offline Web applications. However, a native app is stored on local machine, so the users do not require access to the Web to work with the application.

➔ **In-app purchasing and advertising**

HTML5 apps allow developing in-app stores and advertising. Native apps have these features pre-built in them through app stores. Selling the apps in app store is easy, as HTML5 apps are Web pages that are difficult to sell.

Native apps have an additional benefit, that is trust. Several users are comfortable using application downloaded from app stores than using a Web page. These app stores are preferred than search engines, by the users for finding tools.

## *20.6.4 Converting HTML5 Apps to Native Apps*

Users have a choice of developing their application in HTML5 and then converting them into a native app. This choice has many combined benefits of HTML5 apps and native apps. Users can use tools to convert HTML5 app to a native app.

The following are the best tools used for converting an HTML5 app to native app:

- **PhoneGap**

  PhoneGap is an HTML5 app that allows the user to create native apps with Web technologies and is accessible to app stores and APIs. PhoneGap controls the Web technologies.

- **Appcelerator**

  Appcelerator is a cross-platform mobile application development support. It allows the users to create Android, iOS, and mobile Web apps. Native applications are developed using a JavaScript code base with Eclipse as the IDE.

## *20.7 Check Your Progress*

1. Identify the methods that are used to store and retrieve the data from session storage.

| | | | |
|---|---|---|---|
| (A) | `setItem()` | (C) | `retreiveItem()` |
| (B) | `getItem()` | (D) | `displayItem()` |

2. _____ runs on iOS and Android devices that can be downloaded or purchased from the online app stores.

| | | | |
|---|---|---|---|
| (A) | Native apps | (C) | Web browser |
| (B) | Web apps | (D) | Web Server |

3. The _____ is an HTML5 app tool that allows the user to author native apps with Web technologies and is accessible to app stores and APIs.

| | | | |
|---|---|---|---|
| (A) | Appcelerator | (C) | PhoneGap |
| (B) | Eclipse | (D) | Opera |

4. _____ is a mechanism to store data in a database in the IndexedDB API.

| | | | |
|---|---|---|---|
| (A) | Object store | (C) | Cursor |
| (B) | Index | (D) | Database |

5. Which of the following is the correct code to check the support for session storage in the browser?

| | |
|---|---|
| (A) | `function checkSupport()`<br>`{`<br>`if ((sessionStorage in window) && window[!sessionStorage] !== null)`<br>`{`<br>`alert("Your browser supports Web Storage");`<br>`return;`<br>`}`<br>`}` |
| (B) | `function checkSupport()`<br>`{`<br>`if (('sessionStorage' in window))`<br>`{`<br>`alert("Your browser supports Web Storage");`<br>`return;`<br>`}`<br>`}` |
| (C) | `function checkSupport()`<br>`{`<br>`if (('sessionStorage' in window) \|\| window['sessionStorage'] !== null)`<br>`{`<br>`alert("Your browser supports Web Storage");`<br>`return;`<br>`}`<br>`}` |

| | |
|---|---|
| (D) | `function checkSupport()`<br>`{`<br>`  if  (('sessionStorage'  in  window)  && window['sessionStorage'] !== null)`<br>`  {`<br>`  alert("Your browser supports Web Storage");`<br>`  return;`<br>`  }`<br>`}` |

## *20.7.1 Answers*

| | |
|---|---|
| 1. | A, B |
| 2. | A |
| 3. | C |
| 4. | A |
| 5. | D |

# Summary

- Web Storage is a W3C specification that provides functionality for storing data on the client-side for both temporary as well as permanent needs.
- HTML5 Web applications make use of Web storage to implement client-side persistent storage and they are: session storage and local storage.
- Session storage keeps track of data specific to one window or tab.
- The setItem() and getItem() methods are used to store and retrieve the data from session storage.
- Local storage enables to save data for longer periods on the user's computer, through the browser.
- IndexedDB API is basically an object store that can be used to store and manipulate data on the client-side.
- A native application also called as native app is an application program that is built for a particular device or platform.

# *Try it Yourself*

1. Develop an HTML5 registration Web form with buttons, such as Add, Clear, Display, and Submit. When the user clicks Add, the data entered in the form should be persisted. When the user clicks clear, all the data stored on that page should be deleted. If user clicks Display, the page should print all the data stored in that session. Similarly, on clicking Submit, the next Web page will be displayed with the confirmation details of the user.
2. Modify the **`CompanyDB`** database application designed earlier in this session. Add the functionality to delete the records from the **`employee`** object store. Also, design a form to accept the employee details and add those details to the **`employee`** object store.

# Session - 21

# HTML5 Geolocation and APIs

Welcome to the Session, **HTML5 Geolocation and APIs**.

This session explains about the new APIs supported by HTML5. The session begins with the Geolocation API and Google Maps API used to determine and display the location on a map. Also, it explains the Drag and Drop mechanism which is used to perform the drag-and-drop operations. Finally, the session concludes with a description of Application Cache.

In this Session, you will learn to:

- ➔ Explain geolocation and its use in HTML5
- ➔ Explain the Google Maps API
- ➔ Explain the drag-and-drop operations in HTML5
- ➔ Explain the concept of Application Cache

## *21.1 Introduction*

Consider a scenario where you are visiting a new city and are unaware of specific locations and routes. You want to get information regarding hotels in your locality, such as their exact address, tariffs, and so on. In such a situation, an application which can provide relevant information about the hotels based on your current location would be useful.

A feature that can detect location and list relevant information based on that location is called Geolocation. Geolocation is a term used to identify the geographic location of a person, place, or an object.

Today, modern devices, such as computers, smartphones, tablets, and so on provide Internet-enabled browsers through which the geographic locations of a user or an object can be detected.

## *21.2 Geolocation*

Geolocation in computing terminology indicates a feature that determines the current location of a user on devices. The location of the user is represented as a single point that comprises two components: latitude and longitude. The components can be used further to retrieve more information for the user, such as businesses in the neighborhood or other users within the same coverage area.

Figure 21.1 shows the representation of latitude and longitude with respect to a location on the globe.

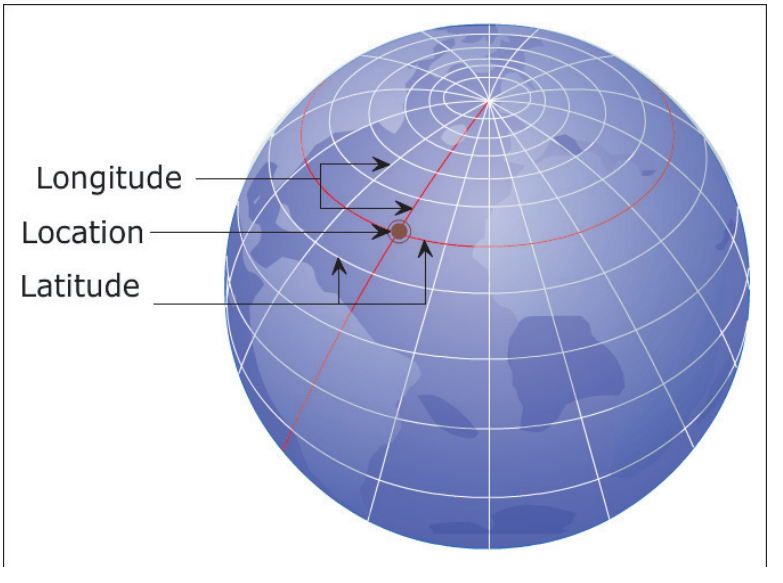



**Figure 21.1: Latitude and Longitude**

There are different sources through which devices can determine the information about the location. These are as follows:

- **GPS** - GPS is a satellite navigation system that provides information about the location on any part of the globe. The GPS system is maintained by the government of the United States and is used by modern mobile devices with GPS capability.
- **IP Address** - Location information can be derived from IP Address. The IP Address is assigned to devices, such as desktops, printers, and so on connected on a network.
- **GSM/CDMA Cell IDs** - These are used by the cell phones.
- **WiFi and Bluetooth MAC address** - These are used by devices that have wireless network connection.
- **User Input** - It is a software tool which can be used on any device requesting for location information. The information retrieved by the tool is based on the data provided by the user, such as, a zip code.

## *21.3 Geolocation API*

In HTML5, the Geolocation API is a specification provided by W3C. It provides a consistent way to develop location-aware Web applications.

The Geolocation API provides a high-level interface that can be used by developers to retrieve location information related to the hosting devices. The interface hides the details, such as how the information is gathered or which methods were used to retrieve the information. This helps the developer to concentrate on geographic information rather than its processing methods.

The object that holds implementation of the Geolocation API is the `Geolocation` object. This object is used in JavaScript to retrieve the geographic information about the devices programmatically. The browser processes the script and returns the location to the Geolocation API.

The Geolocation API is supported on most of the modern browsers available on desktop and mobile phones.

Table 21.1 lists the browsers providing support for Geolocation API.

| Browser | Version Support |
|---|---|
| Safari | 5.0+ |
| Chrome | 5.0+ |
| Firefox | 3.5+ |
| Internet Explorer | 9.0+ |
| Opera | 10.6+ |
| iOS (Mobile Safari) | 3.2+ |

| Browser | Version Support |
|---|---|
| Android | 2.0+ |
| BlackBerry | 6+ |

Table 21.1: Browsers Supporting Geolocation API

**Note** - It is not necessary that information retrieved by the Geolocation API is the actual location of the device. For example, if the satellites are invisible to GPS, then it may not return the accurate location information.

## *21.3.1 Implementing Geolocation Object*

The `Goelocation` object is available as a new property of the `navigator` object. The `navigator` object is a browser object that allows a user to retrieve information about the specific location.

The syntax shows how to create a `Geolocation` object in JavaScript.

**Syntax**:

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

where,

`window`: Is the top level object in JavaScript object hierarchy

Code Snippet 1 demonstrates the script that tests the existence of `Geolocation` object within a browser.

**Code Snippet 1**:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Testing Support for Geolocation in Browsers</title>
  <script>
 function display_location_enabled()
 {
    // Default message
     var str = "Geolocation is not supported in this browser";
```

```
    if (window.navigator.geolocation)
      {
        str = "Geolocation is supported in this browser";
      }
  alert (str);
 }
 </script>
</head>
<body>
  <input type="button" value="Geolocation Support"
        onClick="display_location_enabled()"></input>
</body>
</html>
```

In the code, the `if` statement checks existence of the `geolocation` property in the browser. If the browser provides an implementation for the property, then an alert window displays the message '`Geolocation is supported in this browser`'. Otherwise, the default message is displayed.

Figure 21.2 shows the existence of `Geolocation` object in the Chrome browser.

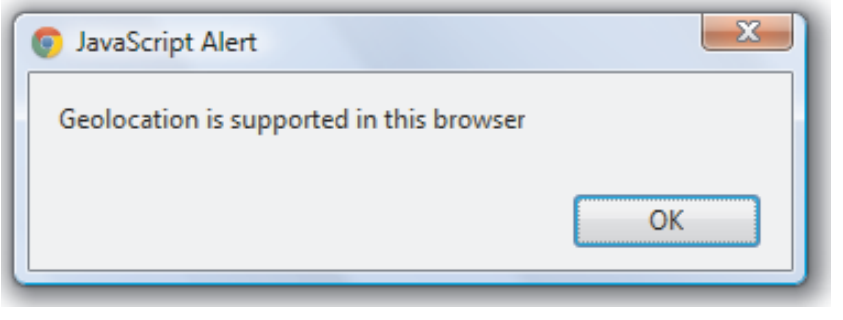



**Figure 21.2: Output – Geolocation Object in Chrome Browser**

## *21.3.2 Geolocation Methods*

The `Geolocation` object provides three methods that can be used to determine the current position of the user.

Table 21.2 lists the methods of the `Geolocation` object.

| Method | Description |
|---|---|
| `getCurrentPosition()` | Retrieves the current geographic location information of the user |
| `watchPosition()` | Retrieves the geographic information of the device at regular intervals |
| `clearWatch()` | Terminates the current watch process |

**Table 21.2: Methods of Geolocation Object**

Also, any changes in the user position is notified through the methods.

## *21.3.3 Retrieve User Position*

The current position of a user is retrieved using the `getCurrentPosition (successCallback, errorCallback, options)` method. This function accepts three parameters, out of which two are optional, `errorCallback` and `options`.

The first parameter, `successCallback` is the name of the function which is invoked after the position of a device is found successfully. The second parameter, `errorCallback` is the name of the function which will be called, if an error occurs in retrieving the position. The last parameter, `options` represents a `PositionOptions` object.

Code Snippet 2 demonstrates the markup that will retrieve the current location of the user.

**Code Snippet 2**:

```
<!DOCTYPE html>
<html >
 <head>
  <title>Geolocation API</title>
  <script>
  function getLocation()
  {
    if (navigator.geolocation) {
       navigator.geolocation.getCurrentPosition(showPosition);
    }
    else{
     alert ("Geolocation is not supported in this browser.");
    }
  }
```

```
function showPosition(position)
{
   alert('Latitude: '+position.coords.latitude+'\n'+'Longitude:
                                '+position.coords.longitude);
}
</script>
</head>
<body>
        <input type="button" value=" Display Location"
                        onClick="getLocation()"/>
</body>
</html>
```

In the code, the `getCurrentPosition()` function obtains the position which is passed as a parameter to the `showPosition()` function. The `showPosition()` function obtains the coordinates of a location through `position` object.

The `position` object is defined in the Geolocation API and holds the current location of the device. It contains attribute named `coords` that retrieves the latitude and longitude of the location. The values retrieved for latitude and longitude are in decimal degrees.

Table 21.3 lists the attributes of the `position` object.

| Attribute | Description |
|---|---|
| coords | An object of type Coordinates that provides different properties, such as latitude, longitude, altitude, accuracy, speed, and so on |
| timestamp | An object of type DOMTimeStamp |

**Table 21.3: Attributes of the position Object**

Figure 21.3 shows the notifications for the Web page containing geolocation code. The browser seeks permission from the user to share their location information with the application.

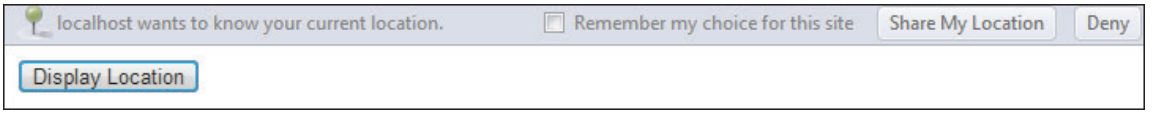



**Figure 21.3: User Permission to Access Geolocation Application**

Figure 21.4 shows a message displaying current location of the user, when the **Share My Location** button is clicked.

Figure 21.4: Output - Message Displayed in Opera

**Note** - The geolocation code works best on the latest Opera browser.

## *21.3.4 Handling Errors*

An application could fail in gathering geographic location information. In that case, the `Geolocation` object calls an `errorCallback()` function. The `errorCallback()` function handles errors by obtaining a `PositionError` object from the API.

➔ **PositionError Object**

The `PositionError` object holds information related to errors occurred while finding the geographic location of the user.

Table 21.4 lists the properties of `PositionError` object.

| Property | Description |
|---|---|
| code | Returns a numeric value for the type of error occurred |
| message | Returns a detailed message describing the error encountered. The message can be used for debugging |

Table 21.4: Properties of the PositionError Object

Table 21.5 lists the different error codes returned by the `code` property of the `PositionError` object.

| Code | Constant | Description |
|---|---|---|
| 1 | PERMISSION_DENIED | Application does not have permission to access Geolocation API |
| 2 | POSITION_UNAVAILABLE | Position of the device could not be obtained |
| 3 | TIMEOUT | Unable to retrieve location information within the specified interval |

Table 21.5: Error Codes

Code Snippet 3 demonstrates the error handling routine for the geolocation code.

**Code Snippet 3**:

```
<!DOCTYPE html>
<html>
<head>
 <title>Handling Error</title>
 <script>
 function getLocation()
 {
function showPosition(position)
 {
 alert('Latitude: '+position.coords.latitude+'\n'+
       'Longitude: '+ position.coords.longitude);
 }
 function errorHandler(error) {
    switch (error.code) {
       case error.PERMISSION_DENIED:
             alert ('You have denied access to your position.');
             break;
       case error.POSITION_UNAVAILABLE:
             alert ('There was a problem getting your position.');
             break;
       case error.TIMEOUT:
             alert ('The application has timed out attempting to
                   get your position.');
            break;
 }
}
 </script>
```

```
</head>
<body>
 <input type="button" value="Display Location"
                    onClick="getLocation()"/>
</body>
</html>
```

In the code, if the application fails to find the current location of the user, then the `errorHandler()` function is invoked. The function is passed as the second parameter in the `getCurrentPosition()` method and is used to handle the errors occurred in the application. It obtains the `error` object which is of type `PositionError` from the API and compares it with the error codes specified in the `switch-case` statement. Depending on the error that has occurred, the appropriate `case` statement is executed and an alert message is displayed to the user.

Figure 21.5 shows the output displaying error message for geolocation application. The reason for displaying error is that the Chrome browser blocks the URL whose file path starts with `file:///`.

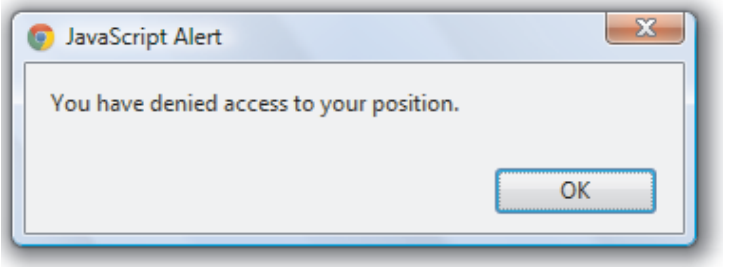



**Figure 21.5: Output – Error Message in Chrome**

## *21.3.5 PositionOptions Object*

`PositionOptions` object is an optional third parameter passed to the `getCurrentPosition()` method. This object defines properties that are optional and are used by an application while retrieving the geolocation information.

Table 21.6 lists the attributes of `PositionOptions` object.

| Attribute | Description |
|---|---|
| enableHighAccuracy | Indicates that the application wants to receive the most accurate results for geolocation. The default value of the attribute is false |
| maximumAge | Obtains the cached position object whose age is less than the specified maximumAge limit (in milliseconds). If age limit is set to 0, then the application must obtain a new position object |

| Attribute | Description |
|---|---|
| timeout | Indicates the maximum time length (in milliseconds) for which the application can wait to obtain the position object |

Table 21.6: Attributes of the PositionOptions Object

Code Snippet 4 demonstrates the script to set the attributes of `PositionOptions` object.

**Code Snippet 4**:

```
<script>
     var options = {
        enableHighAccuracy: true,
        maximumAge: 50000,
        timeout: 60000
     };
function getLocation() {
  if (navigator.geolocation) {
   navigator.geolocation.getCurrentPosition(showPosition,
                                  errorHandler, options);
  }
 else{
     alert ("Geolocation is not supported in this browser.");
 }
}
...
</script>
```

In the code, an object named **`options`** is set with attributes. The attribute `maximumAge` enables the application to use a cached `position` object which is not older than 50 seconds. Also, the `timeout` limit is set to 60 seconds for an application, before notifying an error.

The **`options`** is passed as third parameter to the `getCurrentPosition()` method.

## *21.4 Google Maps API*

The Google Maps API is used to display locations on a map based on the values of their coordinates - latitude and longitude. The Google Maps API must be configured in JavaScript, before it can be referenced

further on the page. It contains a `Map` object which is instantiated and displayed on a Web page.

The syntax shows the configuration of Google Maps API in JavaScript.

**Syntax**:

```
<script src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

where,

`src`: Is the URL of Google Maps API

`sensor`: Parameter sent with the URL. It indicates whether application uses any sensor, such as GPS system to obtain the location of a user. Its value must be explicitly set to `true` or `false`

Code Snippet 5 demonstrates how to load and initialize the Google Maps API in the `<script>` tag. The code will execute after the page is loaded completely and will invoke a function in response to the `onload` event.

**Code Snippet 5**:

```
<!DOCTYPE html>
<html>
 <head>
 <title>Load and Initialize Google Maps</title>
  <style>
      html { height: 100% }
  body { height: 100%; width: 100%; margin: 10% }
  #map_canvas { height: 50%; width: 50% }
  </style>
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script>
```

```
function initialize()
{
 // Loading Google Maps
   var num = new google.maps.LatLng(51.528663,-0.173171);
   var myOptions = {
      zoom: 16,
      center: num,
      mapTypeId: google.maps.MapTypeId.HYBRID
   };
      var mymap = new google.maps.Map(document.getElementById("map_
                          canvas"), myOptions);

  var marker = new google.maps.Marker({
      position: num,
      map: mymap,
      title:"Lord's Cricket Ground, London!"
  });
 }
</script>
 </head>
  <body onload="initialize()">
      <div id="map_canvas"></div>
  </body>
</html>
```

In the code, the URL "`http://maps.google.com/maps/api/js?sensor=false`" defines all symbols and definitions to be loaded for the Google Maps API. Then, the function **`initialize()`** is invoked after the page is loaded completely. This function creates the object of type `Map` and initializes it with the map initialization variables.

In the function, **`var myOptions = {}`**, is an object of type `options` that contains properties, such as `zoom`, `center`, and `mapTypeId`. These properties are used to initialize the map.

Then, the statement `new google.maps.Map (document.getElementById("map_canvas"), myOptions);` creates an instance of `Map` object. The object is displayed in a container on the Web page specified with the `<div>` element.

Finally, to display an icon on the identified location on the Google maps, the `Marker` object is created. The `Marker` object's constructor sets the value for the properties, such as `position`, `map`, and `title`. The `position` property is specified with the location of the marker on the map. The `map` property is specified with the `Map` object to attach the marker with the map. Also, the `title` property sets the title to be displayed as a tooltip on the map.

As mentioned earlier, the **`myOptions`** object has several properties.

Table 21.7 lists some of these properties.

| Property | Description |
|---|---|
| `zoom` | Sets the initial resolution at which map is displayed. A lower zoom value 0 represents a full map of the Earth. Similarly, a higher zoom value displays a map with high resolution. In Code Snippet 5, the zoom level is set to 16. |
| `center` | Centers the map on a specific point by creating an object of type `LatLng` which holds the location coordinates. In Code Snippet 5, the map is centered with the location coordinates `51.528663,-0.173171`. These coordinates display a map centered on Lord's Cricket Ground in London, England. |
| `mapTypeId` | Sets an initial map type. The map types supported are: `ROADMAP` for normal, `SATELLITE` for photographic tiles, `HYBRID` for roads and city names, and `TERRAIN` for water features. In Code Snippet 5, the map type has been set as `google.maps.MapTypeId.HYBRID`. |

**Table 21.7: Properties of the myOptions Object**

Figure 21.6 displays the `Map` object on the Web page that is centered on Lord's Cricket Ground in London.

**Figure 21.6: Map Object**

## *21.4.1 Tracking User's Location on Google Maps*

The `Geolocation` object is used by the Google Maps API to display the geolocation information in the applications.

Code Snippet 6 demonstrates the code that displays current location of a user on the map using `Geolocation` object.

**Code Snippet 6**:

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <style>
   html, body {
   width: 100%;
   height: 100%;
   padding: 10%
  }
  #map_canvas {
   height: 50%;
   width: 50%;
 }
  </style>
  <script src="http://maps.google.com/maps/api/js?sensor=false">
  </script>
  <script>
  // Check support for Geolocation in the browser
  if (navigator.geolocation) {
   // Locate position and invoke function
     navigator.geolocation.getCurrentPosition(displayPosition,
                                              errorFunction);
  } else {
     alert('Geolocation is not enabled in your browser');
 }
```

```
  // Success function
 function displayPosition(position) {
  var my_lat = position.coords.latitude;
  var my_lng = position.coords.longitude;
  var div_info = document.getElementById('user_location');
  div_info.innerHTML = '<h1> Latitude is :' + my_lat + ' and Longitude is
                                                  ' + my_lng + '</h1>';
  // Load Google Maps
   var latlng = new google.maps.LatLng(my_lat, my_lng);
   var myOptions = {
    zoom: 2, //the initial resolution is set at which map is displayed
    center: latlng, //centers the map
    mapTypeId: google.maps.MapTypeId.ROADMAP //sets the map type
   };
  // Creates the Map object
     var map = new google.maps.Map(document.getElementById("map_ canvas"),
myOptions);
// Displays icon on the located position
  var marker = new google.maps.Marker({
      position: latlng,
      map: map,
      title:"User location"
     });
   }
 // Error callback function
 function errorFunction(pos) {
          alert('Error!');
 }
</script>
 </head>
```

```
  <body>
     <div id="map_canvas"></div>
     <div id="user_location"></div>
 </body>
</html>
```

The code uses the `getCurrentPosition()` method and retrieves the current position of the user. Then, it passes the information to **`displayPosition()`** function, which retrieves the coordinates, latitude and longitude. The retrieved coordinates are set into the properties of the `Options` object named **`myOptions`** and initialize the `Map` object. Finally, the `Map` object is displayed along with the current position information in the `<div>` element.

Figure 21.7 shows the output displaying the current location of the user on the Google Maps.

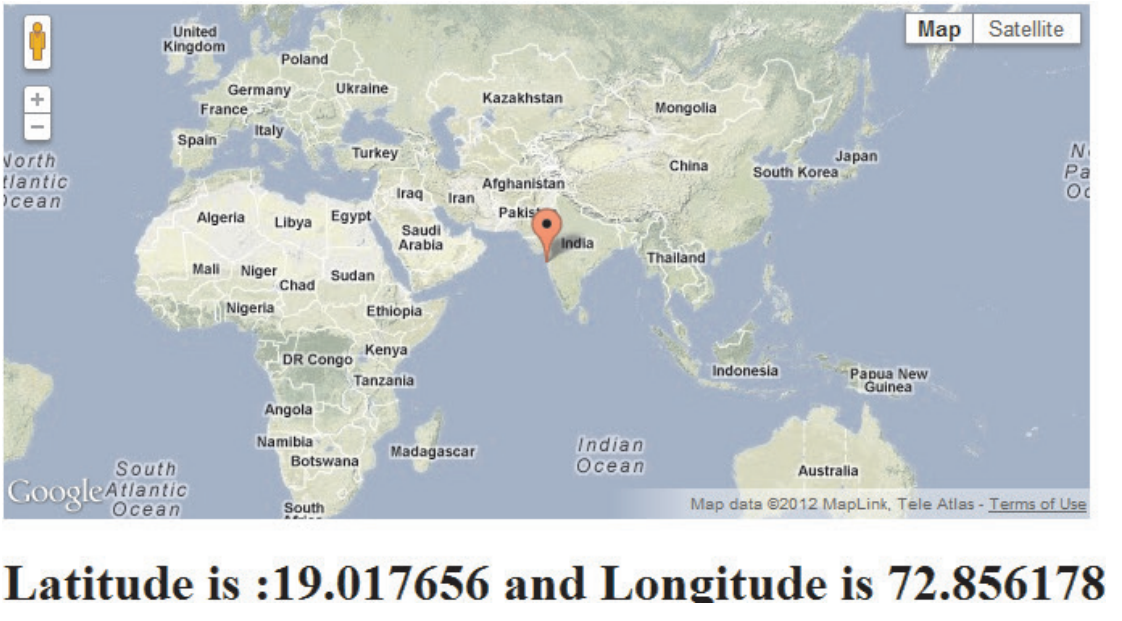



**Figure 21.7: Current User Location on Google Maps**

## *21.5 Drag and Drop*

HTML5 defines drag-and-drop operations that are based on events. The event-based mechanism allow the elements to be copied, reordered, or deleted on a Web page. The drag-and-drop operation involves the use of a pointing device, such as mouse on a visual medium. To perform the drag operation, a mousedown event is triggered followed by multiple mousemove events. Similarly, the drop operation is performed when a user releases the mouse.

The benefit of drag-and-drop mechanism is that it has brought the drag-and-drop operations on the browser level. This makes the programming easier, thus eliminating the need of complex JavaScript code written in earlier HTML versions.

Currently, drag-and-drop operations are supported by all major browsers.

## *21.5.1 Drag Operation*

The steps required to make any element draggable on a Web page are as follows:

- ➔ Set the `draggable` attribute of an element to be dragged
- ➔ Set an `ondragstart` event on the element which stores the data being dragged
- ➔ Store the data into the `DataTransfer` object

Code Snippet 7 shows how to set the `draggable` attribute of an image element.

**Code Snippet 7**:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Drag and Drop API</title>
 </head>
 <body>
      <div id="div" style="border: red 2px solid; height:125px;
                              width:75px; padding: 10px">
      <img src="image.jpg" height="75" width="75" id="image1"
                                       draggable="true"/>
      </div>
 </body>
</html>
```

In the code, the `<img>` element contains `draggable` attribute that is set to `true`. The value `true` indicates that the element is eligible for dragging.

## *21.5.2 Drag Events*

During various stages of the drag-and-drop operation, a number of events are fired. These events are mouse-based events.

Table 21.8 lists the various events triggered during the drag operation.

| Events | Description |
|---|---|
| dragstart | Triggers when an element is started to be dragged by the user |
| drag | Triggers when an element is being dragged using a mouse |
| dragleave | Triggers when the drag and drop operation is completed |

**Table 21.8: Drag Events**

## *21.5.3 DataTransfer Object*

The `dataTransfer` object reveals the **drag data store** that contains the dragged data in the drag-and-drop operation. It allows getting and setting of the data being dragged. In other words, the `dataTransfer` object holds the data during drag-and-drop operation.

The `dataTransfer` object enables to define two types of information. These are as follows:

1. The data type of the draggable element

2. The value of the data being stored in the data store

Code Snippet 8 demonstrates how to associate an element with `dragstart` event to store the data being dragged.

**Code Snippet 8**:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Drag and Drop API</title>
      <script>
       function drag_image(event)
       {
        event.dataTransfer.setData("image", event.target.id);
       }
       </script>
 </head>
```

```
<body>
  <div id="div1" style="border: blue 2px solid; height:125px;
                        width:75px; padding: 10px">
        <img src="image.jpg" height="75" width="75" id="image1"
           draggable="true" ondragstart="drag_image(event)"/>
 </div>
 </body>
</html>
```

In the code, the `<img>` element has been set with an event listener for the `dragstart` event. When the image is dragged, then the `dragstart` event is fired and calls `drag_image()` function. The function uses the `dataTransfer` object to store the data during drag-and-drop operation. The string '**`image`**' represents the data type and `event.target.id` represents the value of `id` attribute of the draggable element.

Figure 21.8 shows the output of the image element to be dragged.

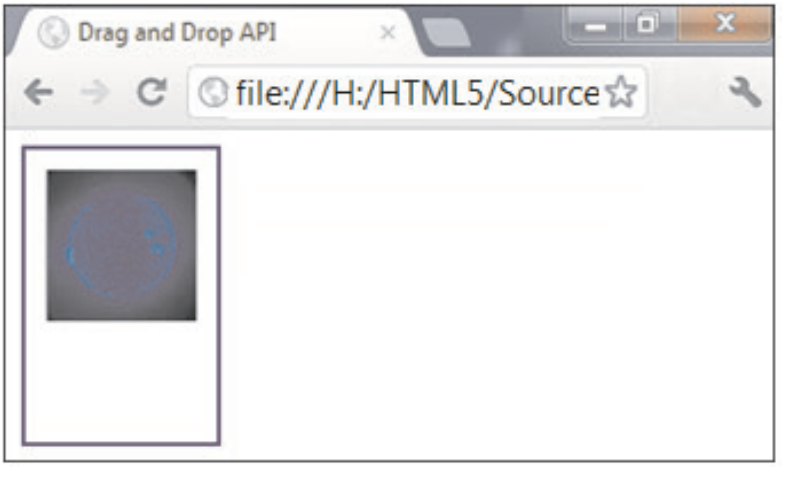



**Figure 21.8: Output – Image to be Dragged**

## *21.5.4 Drop Operation*

After the element has been set up for dragging, it can be dropped on some element on the Web page. By default, elements on the page are not set up to receive dragged elements. Thus, the behavior of element acting as a drop element must be changed. This can be done by creating event listeners for the drop element. The drop element is also referred to as target element.

## *21.5.5 Drop Events*

For any element to receive the drop operation, it must be associated with the drop events.

Table 21.9 lists the events of the drop operation.

| Event | Description |
|---|---|
| dragenter | Triggers when a draggable element is being dragged on the target element for the first time |
| dragleave | Triggers when an element is dragged outside the target element |
| dragover | Triggers when an element is dragged inside the target element |
| drop | Triggers when an element is dropped in the target element |

**Table 21.9: Drop Events**

Code Snippet 9 demonstrates how to set up event listeners to drop the image element on the target element.

**Code Snippet 9**:

```
   <!DOCTYPE html>
<html lang="en">
 <head>
  <title>Drag and Drop API</title>
 <script>
 function drag_image(event)
 {
   event.dataTransfer.setData("image", event.target.id);
 }
 function allow_drop(event)
 {
  event.preventDefault();
 }
 function drop_image(event)
 {
     var data=event.dataTransfer.getData("image");
      event.target.appendChild(document.getElementById(data));
```

```
 }
</script>
</head>
<body>
  <div id="div1" style="border: blue 2px solid; height:125px;
                                width:75px; padding: 10px">
         <img src="image.jpg" height="75" width="75" id="image1"
          draggable="true" ondragstart="drag_image(event)"/>
  </div>
  <br/>
   <div id="div2" style="border: red 2px solid; height:125px;
       width:75px; padding: 10px" ondrop="drop_image(event)"
              ondragover="allow_drop(event)">
    </div>
 </body>
</html>
```

In the code, the `<div>` element with `id` attribute, set as '**`div2`**', is associated with two event listeners namely, `ondragover` and `ondrop`. The `ondropover` calls the `allow_drop()` function which prevents the default behavior of the target element. By default, the browsers do not support dropping of one elements on the other element. To prevent the default behavior, the statement, `event.preventDefault()` is invoked.

Then, the `drop` event is fired on the target element. It calls the function **`drop_image()`** which uses `getData()` method to retrieves image that is set as '**`image`**'. Finally, it appends the dragged image as a element into the target element, `div2`.

Figure 21.9 shows the output of the drop operation, after the image is dragged on the target element.

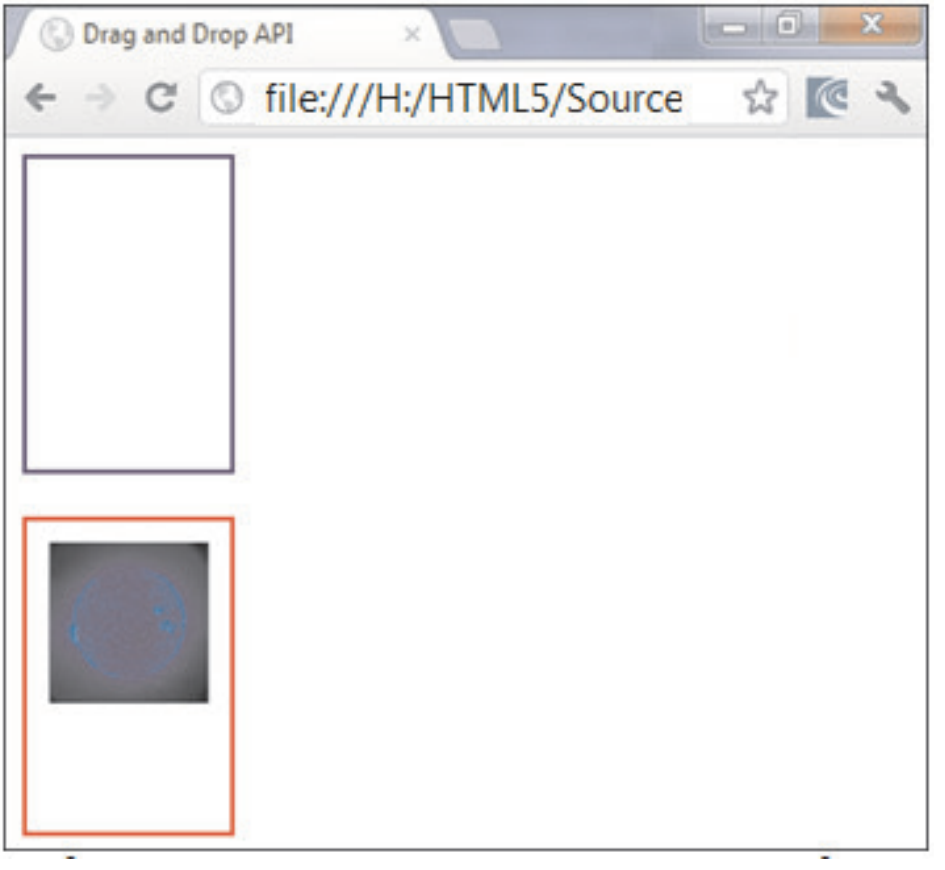



**Figure 21.9: Output – Drop Operation**

## *21.6 Offline Web Applications API*

Consider a situation where a user is travelling outside the coverage area of Internet Service Provider (ISP). In this case, the user will not be able to access Web applications due to the network connection failure.

HTML5 supports offline Web applications that allow a user to work with them without being online. Offline Web applications work by saving all the Web pages locally on the user's system. This feature is known as the **Application Cache**.

The **Application Cache** enables all resources, such as HTML, JavaScript, images, and CSS pages of an Web application to be stored locally on the system.

The following are the steps that can be taken to cache resources locally on the system:

1. Create a manifest file to define the resources that need to be saved.
2. Reference the manifest file in each Web page designed to use cached resources.

## *21.6.1 Creating a Manifest File*

The manifest file is a text file that defines the caching behavior for resources used by the Web page. The file should be saved with the `.manifest` extension.

Code Snippet 10 demonstrates how to create a manifest file.

**Code Snippet 10**:

```
CACHE:
# Defines resources to be cached.
 check.js
 styles.css
 images/figure1.jpg
FALLBACK:
# Defines resources to be used if non-cached resources cannot be #downloaded
 Other_images/ figure2.png
NETWORK:
# Defines resources that will not be cached.
 figure3.png
```

The following are the sections defined in the `.manifest` file:

- `CACHE`: This section defines resources, such as `check.js`, `styles.css`, and `figure1.png` to be stored locally.

- `FALLBACK`: This section defines alternative resource to be used, when the actual resource is not available. For example, `figure2.png` is defined as a fallback image. If a browser cannot access `figure1.jpg` in the `images` folder, then `figure2.png` will replace the unavailable image at the time of rendering the markup on the Web page. The unavailability of the image can be due to network connection or server problem.

- `NETWORK`: This section specifies resources to be accessed when there is a network connection. Resources in this section are not cached.

## *21.6.2 Declaring a Manifest*

To associate a manifest with a Web page, assign `.manifest` file to the attribute named `manifest` specified with the `html` element.

Code Snippet 11 demonstrates how to add the `.manifest` file in an HTML document.

**Code Snippet 11**:

```
<!DOCTYPE html>
<html manifest="appcache.manifest">
 <head>
 <title> Web Page </title>
  <link rel="stylesheet" href="styles.css"/>
  <script type="text/javascript" src="check.js"></script>
 </head>
 <body>
  <input type="button" value="click Here..." onClick="display()"/>
  <img src="images/figure1.jpg" width="100" height="100"/>
 </body>
</html>
```

In the code, the "**`appcache.manifest`**" is specified with the `<html>` tag. The interpretation of the manifest file is similar to any other file reference. The document uses a relative file path, as both the manifest file and HTML document are located in the same directory. By default, a Web page declaring manifest is cached automatically.

The benefit of the Application Cache is that it improves the performance of a Web page by reducing the number of requests made to the Web server. The Web server hosts the Web application to be accessed on the network.

Figure 21.10 shows how to enable the **Work Offline** mode in the Opera browser. This enables to cache the resources of the Web application pages locally.

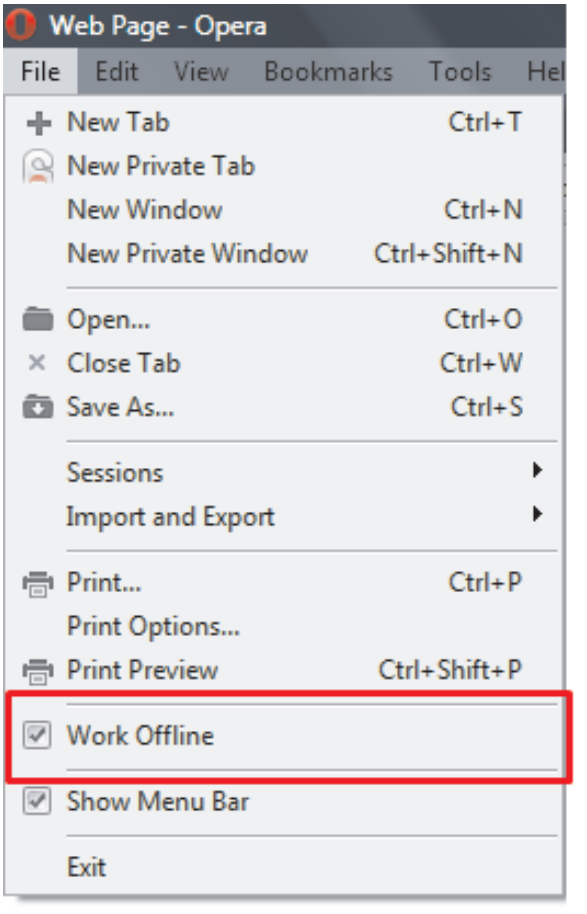



Figure 21.10: Work Offline Mode in Opera

As shown in figure 21.10, **Work Offline** is enabled to cache the resources of the Web page.

Figure 21.11 shows the cached Web page in the Opera browser.

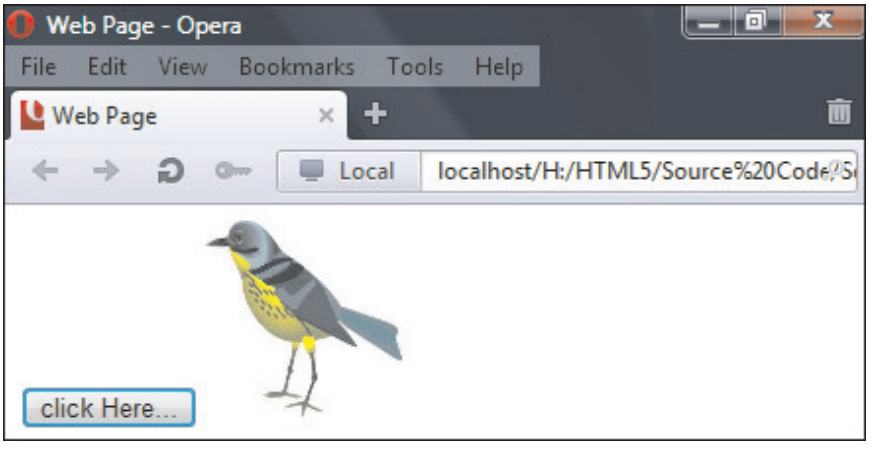



Figure 21.11: Output – Cached Web Page

## *21.7 Check Your Progress*

1. The location of a user is represented as a single _____ on a map.

| | | | |
|---|---|---|---|
| (A) | Point | (C) | Mark |
| (B) | Coordinates | (D) | Symbol |

2. Which one of the following new properties of the `navigator` object is supported by HTML5?

| | | | |
|---|---|---|---|
| (A) | Location | (C) | Geolocation |
| (B) | Cookie | (D) | Script |

3. Which one of the following objects enables to get and set the data of the element being dragged?

| | | | |
|---|---|---|---|
| (A) | dragstart | (C) | drop |
| (B) | dataTransfer | (D) | drag |

4. Identify the steps needed to cache the resources of a Web page locally on the system.

| | | | |
|---|---|---|---|
| (A) | Obtain the resource from online | (C) | Create a manifest file |
| (B) | Reference the manifest file in a Web page | (D) | Create the object of the manifest file in JavaScript |

5. Which of the following is the correct code to load the coordinates on the Google Maps?

| | |
|---|---|
| (A) | `function displayPosition(position) {`<br>`var my_lat = position.coords.latitude;`<br>`var my_lng = position.coords.longitude;`<br>`var latlng = new google.maps.LatLng(my_lat, my_lng);`<br>`}` |
| (B) | `function displayPosition(position) {`<br>`var my_lat = position.coords.latitude;`<br>`var my_lng = position.coords.longitude;`<br>`var latlng = new google.maps.LatitudeLongitude(my_lat, my_lng);`<br>`}` |
| (C) | `function displayPosition(position) {`<br>`var my_lat = coords.latitude;`<br>`var my_lng = coords.longitude;`<br>`var div_info = document.getElementById('user_location');`<br>`var latlng = new google.maps.LatitudeLongitude(my_lat, my_lng);`<br>`}` |
| (D) | `function displayPosition(position) {`<br>`var my_lat = position.coord.latitude;`<br>`var my_lng = position.coord.longitude;`<br>`var latlng = new google.maps.LatLng(my_lat, my_lng);`<br>`}` |

## *21.7.1 Answers*

| | |
|---|---|
| 1. | A |
| 2. | C |
| 3. | B |
| 4. | B, C |
| 5. | A |

# Summary

- Geolocation determines the current location of a user on devices.
- The location is represented as a single point on a map that comprises two components: latitude and longitude.
- The Goelocation API is a specification provided by the W3C which provides a consistent way to develop location-aware Web applications.
- Google Maps API is used to display the user's location on the map.
- The object of type Map is created in JavaScript, before it can be referenced in an HTML document.
- The drag-and-drop operations defines an event-based mechanism using which elements on a Web page can be copied, reordered, or deleted.
- HTML5 supports offline Web applications that allow a user to work with them without being online.

# *Try it Yourself*

1. Develop a mobile Web application to display the current location of a user on Google Maps. The application will make use of jQuery API to handle success and failure conditions, while gathering information about the user location.
2. Create a Web page with a container containing three images and two empty containers. The user can drag the images from the container and drop them in the empty containers back and forth.

Onlinevarsity