New Edition

WEB TECHNOLOGY

Jagdish Bhatta
Basant Chapagain

Ramesh Singh Saud
Sravan Ghimire

1

Chapter

INTRODUCTION

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Web Basics: Internet, Intranet, WWW, Static and Dynamic Web Page
- Web Clients
- Web Servers
- Client Server Architecture
- Single Tier, Two-Tier, Multiple Tier
- HTTP: HTTP Request and Response
- URL, Client Side Scripting Serve Side Scripting, Web 1.0, Web 2.0

Web Basic

Web technology is the establishment and use of mechanisms that make it possible for different computers and devices to communicate and share resources. Web technologies are infrastructural building blocks of any effective computer network: local area network, metropolitan area network or a wide area network, such as the Internet.

Web technologies related to the interface between web servers and their clients. This information includes markup languages, programming interfaces and languages, and standards for document identification and display. In general web technology incorporates tools and techniques for web development.

Web technologies is a general term referring to the many languages and multimedia packages that are used in conjunction with one another, to produce dynamic web sites such as this one. Each separate technology is fairly limited on its own, and tends to require the dual use of at least one other such technology. Therefore we can conclude that all of the components that make up a site are interdependent on one another.

Web Development is a broad term for the work involved in developing a web site for World Wide Web. This can include ***web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.*** However, among web professionals, "web development" usually refers to the main non-design aspects of building web sites: writing markup and coding. Web development can range from developing the simplest static single page of plain text to the most complex web-based internet applications, electronic businesses, or social network services.

Web design is a broad term used to encompass the way that content (usually hypertext or hypermedia) is delivered to an end-user through the World Wide Web, using a web browser or other web-enabled software is displayed. The intent of web design is to create a website—a collection of online content including documents and applications that reside on a web servers. A website may include text, images, sounds and other content, and may be interactive.

For the typical web sites, the basic aspects of design are:

The *content:* The substance and information on the site should be relevant to the site and should target the area of the public that the website is concerned with.

The *usability:* The site should be user-friendly, with the interface and navigation simple and reliable.

The *appearance:* The graphics and text should include a single style that flows throughout, to show consistency. The style should be professional, appealing and relevant.

The *structure:* of the web site as a whole.

Internet

Internet is a short form of the technical term ***internetwork***, the result of interconnecting computer networks with special gateways or routers. The Internet is also often referred to as *the Net*. The Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet.

The internet is a globally connected network system that uses TCP/IP to transmit data via various types of media. The internet is a network of global exchanges – including private, public, business, academic and government networks – connected by guided, wireless and fiber-optic technologies.

Information that travels over the Internet does so via a variety of languages known as protocols. **The Internet is loosely connected compared with the randomized graph.**

The Internet is a globally distributed network comprising many voluntarily interconnected autonomous networks. It operates without a central governing body. However, to maintain interoperability, all technical and policy aspects of the underlying core infrastructure and the principal name spaces are administered by the **Internet Corporation for Assigned Names and Numbers (ICANN).**

The **history of the Internet** starts in the 1950s and 1960s with the development of computers. This began with point-to-point communication between mainframe computers and terminals, expanded to point-to-point connections between computers and then early research into packet switching.

Uses of the Internet

The key to success of Internet is the information. The better the quality, the more usage of Internet operations.

Large volume of Information: Internet can be used to collect information from around the world. This information could relate to education, medicine, literature, software, computers, business, entertainment, friendship, tourism, and leisure.

News and Journals: All the newspapers, magazines and journals of the world are available on the Internet. With the introduction of broadband and advanced mobile telecommunication technologies such as 3G (third generation) and 4G (fourth generation), the speed of internet service has increased tremendously. A person can get the latest news about the world in a matter of few seconds.

Electronic Mode of Communication: Internet has given the most exciting mode of communication to all. We can send an E-mail (the short form of Electronic Mailing System) to all the corners of the world.

Chatting: There are many chatting software that can be used to send and receive real-time messages over the internet. We can chat with our friend and relatives using any one of the chatting software.

Social Networking: People can connect with old friends on social networking sites. They can even chat with them when they are online. Social networking sites also allow us to share pictures with others. We can share pictures with our loved ones, while we are on a vacation. People are even concluding business deals over these social networking sites such as Facebook.

Online Banking (Net-Banking): The use of internet can also be seen in the field of banking transactions. Many banks such as HSBC, SBI, Axis Bank, Hdfc Bank, etc. offers online banking facilities to its customers. They can transfer funds from one account to another using the net-banking facility.

E-commerce: Internet is also used for carrying out business operations and that set of operations is known as Electronic Commerce (E-commerce). Flipkart is the largest e-commerce company in India. The rival, Amazon, is giving stiff competition to Flipkart.

Mobile commerce: Mobile commerce (also M-Commerce) refers to the commercial transaction that takes place over the mobile internet. Using the mobile internet technology, many companies have introduced mobile version of websites and mobile apps, to promote and sell their products. Customers can simply browse several through the products and buy online through mobile internet.

Mobile wallet: Many companies offer the service of mobile wallet to its customers. Users must have a smart-phone and internet connection to use this service. Users can pay an amount into their mobile wallet, which they can use to make online payment such as bill payments, recharges, etc.

Entertainment: Apart from a major source of knowledge and information, the utility of Internet in the field of entertainment cannot be undermined. We can visit various video sites and watch movies and serials at our convenient time.

Technology of the Future: Internet is the technology of future. In the times to come, offices would be managed at distant places through Internet.

Intranet

An intranet is a private enterprise network, designed to support an organization's employees to communicate, collaborate and perform their roles. It serves a broad range of purposes and uses, but at its core, an intranet is there to help employees.

An intranet is a private network contained within an enterprise that is used to securely share company information and computing resources among employees. An intranet can also be used to facilitate working in groups and teleconferences.

Intranets increase communication within an organization by allowing employees to easily access important information, links, applications and forms as well as databases that can provide company records. Security can also be increased within the intranet by establishing a database that maintains all of the usernames of people who are allowed access to the network.

Uses of the intranet

- Streamlining everyday activities by making repeated tasks more feasible.
- Centralizing and managing important information and company data in a single database.
- Making collaboration easier since information can be shared across the entire network.
- Providing personalized content to employees based on their role within the company.
- Improving internal communication by making employee directories, company news and organization charts readily available.
- Providing fast and easy access to information about company policies, benefits and updates.

Main World Wide Web (WWW)

WWW is a system of interlinked hypertext documents accessed via the Internet. The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet.

The term "World Wide Web" (WWW) refers to the collection of public websites connected to the internet worldwide, together with client devices such as computers and cell phones that access its content. For many years it has become known simply as "the web."

The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. Web services, which use HTTP to allow applications to communicate in order to exchange business logic, use the Web to share information. The Web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.

The Web is one of the services that runs on the Internet. It is a collection of textual documents and other resources, linked by hyperlinks and URLs, transmitted by web browsers and web servers. The Web is just one of the ways that information can be disseminated over the Internet, so the Web is just a portion of the Internet. In short, the Web can be thought of as an application "running" on the Internet

Hypertext

Hypertext provides the links between different documents and different document types. In a hypertext document, links from one place in the document to another are included with the text. By selecting a link, you are able to jump immediately to another part of the document or even to a different document. In the WWW, links can go not only from one document to another, but from one computer to another

World Wide Consortium

The **World Wide Web Consortium** (W3C) is the main international standards organization for the World Wide Web. W3C was created to ensure compatibility and agreement among industry members in the adoption of new standards. Prior to its creation, incompatible versions of HTML were offered by different vendors, increasing the potential for inconsistency between web pages. The consortium was created to get all those vendors to agree on a set of core principles and components which would be supported by everyone.

Web Page

A web page is a document or information resource that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a monitor or mobile device. This information is usually in HTML or XHTML format, and may provide navigation to other web pages via hypertext links.

A Web page is a representation of a document that is actually located at a remote site. The information on a Web page is displayed online with the help of a Web browser such as Internet Explorer, Mozilla Firefox or Google Chrome. The Web browser is connected to the Web server, where the website's contents are hosted through HTTP. Every Web page corresponds to various types of information presented to the visitor in a visual and readable manner.

Web pages frequently subsume other resources such as style sheets, scripts and images into their final presentation.

Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).

Web pages may consist of files of static text and other content stored within the web server's file system (static web pages), or may be constructed by server-side software when they are requested (dynamic web pages). Client-side scripting can make web pages more responsive to user input once on the client browser.

Web Site

A **website** or simply **site** is a collection of related web pages containing images, videos or other digital assets. A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a Uniform Resource Locator. All publicly accessible websites collectively constitute the World Wide Web. *Web sites can be static or dynamic.*

Static Website

A static website is one that has web pages stored on the server in the format that is sent to a client web browser. It is primarily coded in Hypertext Markup Language, HTML.

Simple forms or marketing examples of websites, such as classic website, a five-page website or a brochure website are often static websites, because they present pre-defined, static information to the user. This may include information about a company and its products and services via text, photos, animations, audio/video and interactive menus and navigation.

This type of website usually displays the same information to all visitors. Similar to handing out a printed brochure to customers or clients, a static website will generally provide consistent, standard information for an extended period of time. Although the website owner may make updates periodically, it is a manual process to edit the text, photos and other content and may require basic website design skills and software.

In summary, visitors are not able to control what information they receive via a static website, and must instead settle for whatever content the website owner has decided to offer at that time.

Dynamic Website

A dynamic website is one that changes or customizes itself frequently and automatically, based on certain criteria.

Dynamic websites can have two types of dynamic activity: Code and Content. Dynamic code is invisible or behind the scenes and dynamic content is visible or fully displayed.

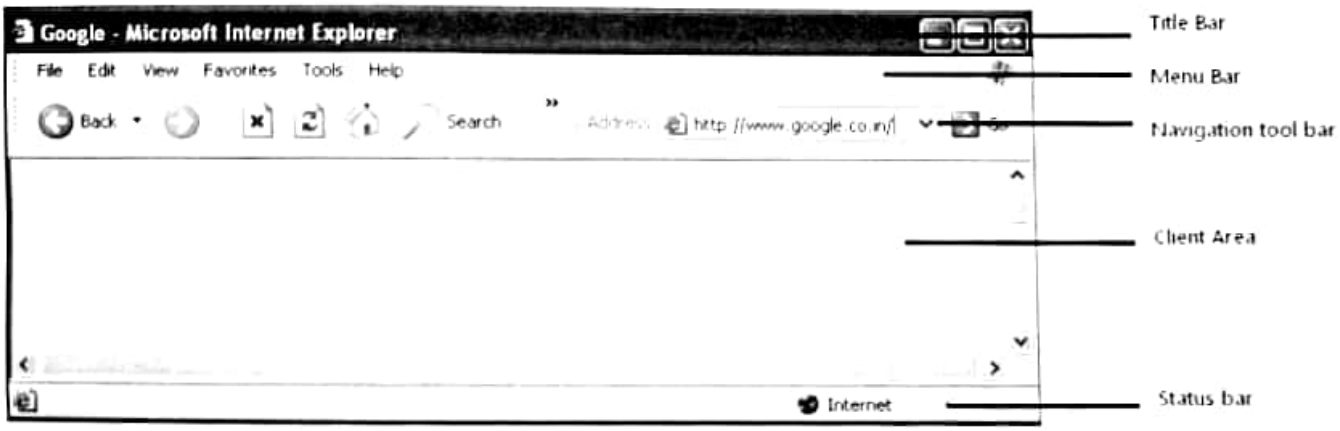
The first type is a web page with dynamic code. The code is constructed dynamically on the fly using active programming language instead of plain, static HTML.

The second type is a website with dynamic content displayed in plain view. Variable content is displayed dynamically on the fly based on certain criteria, usually by retrieving content stored in a database

Web Clients

The Web client is a client-side component within a distributed multi-tiered application model used for building and developing enterprise applications. Client-side components are typically computer applications running on a user's computer and connect to a server. These components perform client-side operations as they might need access to information available only on the client side, like user input, or because the server lacks the processing power necessary in such operations.

Browsers are software programs that allow you to search and view the many different kinds of information that's available on the World Wide Web. The information could be web sites, video or audio information.



- **Status Bar:** You will find the status bar at the very bottom of your browser window. It basically tells you what you are doing at the moment. Mainly, it shows you load speed and the URL address of whatever your mouse is hovering over.
- **Title Bar:** You will find this bar at the absolute top of your browser and in will be the color blue for the major browsers. The purpose of the Title bar is to display the title of the web page that you are currently viewing.
- **Menu Bar:** The menu bar contains a set of dropdown menus
- **Navigational Tool:** A bar contains standard push button controls that allow the user to return to a previously viewed page, to reverse and refresh the page, to display the home page and to print the page etc.
- **Toolbar Icons:** You will find the Toolbar directly under the Title Bar. The Toolbar is where you will find the back button, home button and the refresh button etc.
- **Client Area:** It is a display window which is the space in which you view the website.
- **Scroll Bars:** The Scroll bars, usually located to the right of the Display Window, allows you to "scroll" (move down or up the web page) so you can view information that is below or above what is currently in the Display Window.

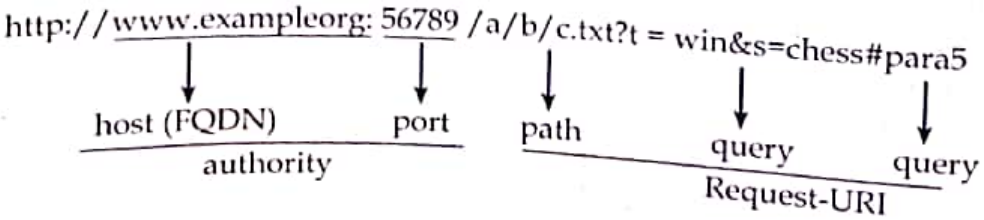
Web Servers

Basic functionality:

- It receives HTTP request via TCP
- It maps Host header to specific virtual host (one of many host names sharing an IP address)
- It maps Request-URI to specific resource associated with the virtual host
  - File: Return file in HTTP response
  - Program: Run program and return output in HTTP response
- It maps type of resource to appropriate MIME type and use to set Content-Type header in HTTP response
- It Logs information about the request and response
- All e-commerce site require basic Web server software to answer requests from customers like ;
  - Apache
    - ✓ Leading Web server software (47% of market)
    - ✓ Works with UNIX, Linux , Windows OSs
  - Microsoft's Internet Information Server (IIS)
    - ✓ Second major Web server software (25% of market)
    - ✓ Windows-based

Domain Names, DNS, and URLs

- IP addresses are not convenient for users to remember easily. So an IP address can be represented by a natural language convention called a **domain name**
- **Domain name system (DNS)** translates domain names into IP addresses. DNS is the "phone book" for the Internet, it maps between host names and IP addresses.
- **A uniform resource locator (URL)**, which is the address used by a Web browser to identify the location of content on the Web, also uses a domain name as part of the URL.
- **Syntax: scheme: scheme-depend-part.** Example: In http://www.example.com/, the scheme is http.



HTTP

- HTTP is based on the request-response communication model:
  - Client sends a request
  - Server sends a response
  - HTTP is a stateless protocol: where the protocol does not require the server to remember anything about the client between requests.
- Normally implemented over a TCP connection (80 is standard port number for HTTP)
- The following is the typical browser-server interaction using HTTP:
  - User enters Web address in browser
  - Browser uses DNS to locate IP address
  - Browser opens TCP connection to server
  - Browser sends HTTP request over connection
  - Server sends HTTP response to browser over connection
  - Browser displays body of response in the client area of the browser window

Client/Server Computing:

- A model of computing in which powerful personal computers are connected in a network together with one or more servers
- **Client** is a powerful personal computer that is part of a network; service requester
- **Server** is a networked computer dedicated to common functions that the client computers on the network need; service provider
- Web is based on client/server technology. Web servers are included as part of a larger package of internet and intranet related programs for serving e-mail, downloading requests for FTP files and building and publishing web pages. Typically the e-commerce customer is the client and the business is the server. In the client/ server model single machine can be both client and the server The client/ server model utilises a database server in which RDBMS user queries can be answered directly by the server.
- The client/ server architecture reduces network traffic by providing a query response to the user rather than transferring total files. The client/ server model improves multi-user updating through a graphical user interface (GUI) front end to the shared database. In client/ server architectures client and server typically communicate through statements made in structured query language (SQL).

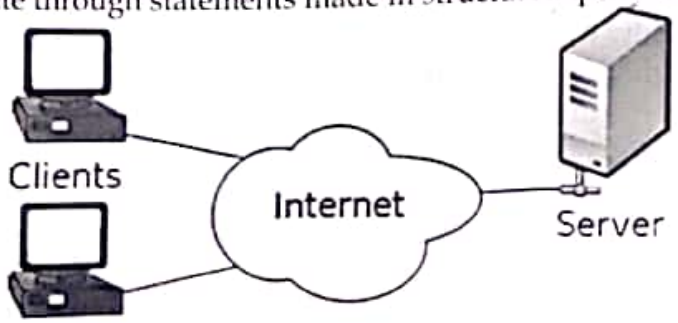


Figure: Client/ Server Model

Single Tier, Multiple tier, Two tier From BCA

Web Clients

It typically refers to the Web browser in the user's machine. It is a software application for retrieving, presenting, and traversing information resources on the web server. It is used to create a HTTP request message and for processing the HTTP response message.

User agent: Any web client is designed to directly support user access to web servers is known as user agent. Web browsers can run on desktop or laptop computers. Some of the browsers are: Internet Explorer, Mozilla, FireFox, Chrome, Safari, Opera, Netscape Navigator.

Client-Side Scripting

- Client-side scripting generally refers to writing the class of computer programs (scripts) on the web that are executed at client-side, by the user's web browser, instead of server-side (on the web server). Usually scripts are embedded in the HTML page itself.
- JavaScript, VBScript, Jscript, Java Applets etc. are the examples of client side scripting technologies. JavaScript is probably the most widely used client-side scripting language.
- Client-side scripts have greater access to the information and functions available on the user's browser, whereas server-side scripts have greater access to the information and functions available on the server. Upon request, the necessary files are sent to the user's computer by the web server (or servers) on which they reside. The user's web browser executes the script, then displays the document, including any visible output from the script.
- Client-side scripts may also contain instructions for the browser to follow in response to certain user actions, (e.g., clicking a button). Often, these instructions can be followed without further communication with the server.

Server-Side Scripting:

- Includes writing the applications executed by the server at run-time to process client input or generate document in response to client request. So server side script consists the directives embedded in Web page for *server* to process before passing page to requestor.
- It is usually used to provide interactive web sites that interface to databases or other data stores.
- This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.
- PHP, JSP, ASP.... etc, are the server side scripting technologies.

Web 2.0:

The term **Web 2.0** is associated with web applications that facilitate participatory *information sharing, interoperability, user-centered design, and collaboration* on the World Wide Web. Web 2.0, a second advanced generation of WWW, is about revolutionizing the way of creating, editing and sharing user generated content online. Web 2.0 is one of the series of improved technology

rather than a specific version of web. It is characterized specifically as a transition from Static web pages to highly Dynamic web pages or user generated content. A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community, in contrast to websites where users are limited to the passive viewing of content that was created for them. Examples of Web 2.0 include *social networking sites, blogs, wikis, video sharing sites, hosted services, web applications.*

Features of Web 2.0

- Web 2.0 uses the approach of "guide on the Side" rather than implementing "top-down" approach i.e dynamically change or edit the content rather then simply reading.
- It changed the concept of "mostly read only web" to "widely read and write" over web.
- Web 2.0 provides a perfect platform base for effective user interaction that was not available before.
- It changed the idea from passive consumption and delivery of content, to actively participating in creation, sharing, and collaboration.
- It is subjected to be a powerful lure for an Enterprise; that fetch more employees into accounts at a lower cost for greater participation in projects and idea sharing.

Advantages of Web 2.0:

- Available at any time, any place.
- Variety of media.
- Ease of usage.
- Learners can actively be involved in knowledge building.
- Can create dynamic learning communities.
- Everybody is the author and the editor, every edit that has been made can be tracked.
- User friendly.
- Updates in wiki are immediate and it offers more sources for researchers.
- Provides real-time discussion.

2
Chapter

HYPER TEXT MARKUP LANGUAGE

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Introduction to HTML
- Elements of HTML Document
- HTML Elements and HTML Attributes, Headings, Paragraph, Division, Formatting: bi, i, small, sup, sub
- Spacing: pre, Br
- Formatting ext Phrases: Span, strong, tt
- Image elements
- Anchors
- Lists: ordered and Unordered and Definition
- Tables, Frames, Forms: Form Elements, ID attributes, Class Attributes of HTML Elements
- Meta Tag, Audio, Video, Canvas, Main, Section, Articles Header, Footer, Aside, Nav, Figure Tags
- HTML Events: Windows Events, form Element Events, Keyboard Events, Mouse Events

Introduction to HTML

HTML stands for **hypertext markup language**. It is not a programming language. A markup language specifies the *layout and style* of a document. A markup language consists of a set of **markup tags**. HTML uses markup tags to describe web pages. An HTML tag is commonly defined as a set of characters constituting a formatted command for a Web page. At the core of HTML, tags provide the directions or recipes for the visual content that one sees on the Web.HTML tags are keywords surrounded by **angle brackets** like <html>. Most HTML tags normally come in pairs like and . The first tag is called the **start tag** (or **opening tag**) and the second tag is called the **end tag** (or **closing tag**). HTML documents describe Web pages. HTML documents contain HTML tags and plain text. HTML documents are also called Web pages. A web browser read HTML documents and displays them as Web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. A simple HTML document is given below:

```
<html>
        <head>
                <title>This is my first web page</title>
        </head>
        <body>
                <h1>My first heading</h1>
                <p>My first paragraph</p>
        </body>
</html>
```

Save this page with **.html** or **.htm** extension. However, it is good practice to use **.htm** extension.

HTML Elements

An HTML element is an individual component of an HTML document or web page. HTML is composed of a tree of HTML nodes, such as text nodes. Each node can have HTML attributes specified. Nodes can also have content, including other nodes and text. Many HTML nodes represent semantics, or meaning. For example, the <title> node represents the title of the document. An HTML element is everything from the start tag to the end tag. For example, My first paragraph . An HTML element consists of start tag, end tag, and element content. The element content is everything between the start tag and end tag. Empty elements are closed in the start tag. Most HTML elements can have attributes. For example, src attribute of img tag.

HTML Attributes

HTML attributes are special words used inside the opening tag to control the element's behavior. HTML attributes are a modifier of an HTML element type attributes provide additional information about HTML elements. Attributes are always specified in the start tag. Attributes come in name/value pair like name = "value". For example, HTML links are defined with **<a>** tag and the link address is provided as an attribute **href** like **cdcsit**.

Note: Always quote attribute values and use lowercase attributes.

HTML Headings

HTML headings are defined with the <h1> to <h6> tags. <h1> displays largest text and <h6> smallest.

For example,

```
<html>
    <h1>This is heading with h1 tag</h1>
    <h2>This is heading with h2 tag</h2>
    <h3>This is heading with h3 tag</h3>
    <h4>This is heading with h4 tag</h4>
    <h5>This is heading with h5 tag</h5>
    <h6>This is heading with h6 tag</h6>
</html>
```

Output:

This is heading with h1 tag

This heading with h2 tag

This is heading with h3 tag

This is heading with h4 tag

This is heading with h5 tag

This is heading with h4 tag

HTML Paragraphs

HTML paragraphs are defined with <p> tag. Paragraphs are separated by new line.

For example:

```
<html>
    <body>
        <p>This is a paragraph.</p>
        <p>This is a paragraph.</p>
        <p>This is a paragraph.</p>
    </body>
</html>
```

Output:
This is a paragraph.
This is a paragraph.
This is a paragraph.

HTML Comments

We use comments to make our HTML code more readable and understandable. Comments are ignored by the browser and are not displayed. Comments are written between <!-- and - ->.

For example

```
<html>
    <body>
        <p>Comments are not displayed in browser</p>
        <!--comments are used to make code understandable and readable-->
    </body>
</html>
```

Output:

Comments are not displayed in browser

HTML Line Breaks

If you want a new line (line break) without starting a new paragraph, use <br /> tag.

HTML Formatting Tags

We use different tags for formatting output, like<b>,<i>,<small>,<sup>,<sub> illustrated in example.

For example:

```
<html>
    <body>
        <p><b>This tag is used to bold the text</b></p>
        <p><i>This tag is used to italic the text</i></p>
        <p><small>This tag defines smaller text other than other tag</small></
p>
        <p>This text contains <sup>superscript</sup> text.</p>
        <p>This text contains <sub>subscript</sub> text.</p>
    </body>
</html>
```

Output:

This tag is used to bold the text

This tag is used to italic the text

This tag defines smaller text other than other tag

This text contains superscript text

This text contains subscript text

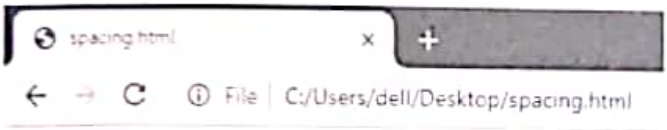
HTML Spacing

A common character entity used in HTML is the non-breaking space. i.e: & **nbsp;** Two words separated by a non-breaking space will stick together (not break into a new line)

For example:

```
<html>
    <body>
        Spacing between 10 and gm is given as 10   gm
    </body>
</html>
```

Output:



Spacing between 10 and gm is given as 10 gm

HTML Formatting Text

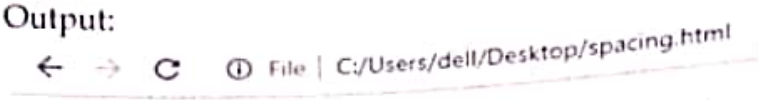
The following are the tags used in formatting the text like: <pre>, <br>, tags , , <tt>, <a> and <img>.The <pre> tag is used to defines preformatted text.Text in a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks. The
tag inserts a single line break as illustrated in example 1

The tag is used to group inline-elements in a document. The tag provides no visual change by itself. The tag provides a way to add a hook to a part of a text or a part of a document. The tag gives text a strong emphasis which traditionally means that the text is displayed as bold by the browser. The <tt> tag defines teletype text which is not supported in HTML 5 and onwards. The <a> tag defines hyperlink, which is used to link from one page to another. The <img> tag is the image element tag defines an image in an HTML page, which requires two attributes such as src and alt. The src attributes specifies the URL of an image and the alt attribute specifies an alternate text for an image.

Example

```
<html>
<body>
      <pre>
      Text in a pre element
      is displayed in a fixed-width
      font, and it preserves
      both      spaces and
      line breaks
      </pre>
      This line has br <br> tag.
      </body>
</html>
```

Output:

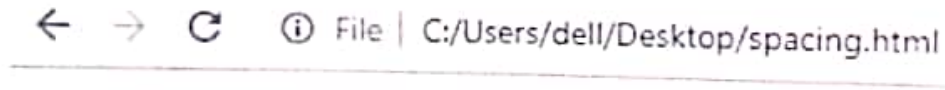



```
Text in a pre element
is displayed in a fixed-width
font, and it preserves
both space and line breaks
this line has for
tax
```

Example 2:

```
<html>
<body>
    <p>My mother has eyes and my father has <span>dark green</span> eyes.</p>
    <p>My mother has eyes and my father has <strong>dark green</strong> eyes.<
/p>
    <p>This is deprecated tag <tt>deprecated</tt></p>
    <p><a href="#">Click Here to go Google</a></p>
    <img src="sankheswor-mahadev-
temple.jpg" alt="Smiley face" width="42" height="42">
</body>
</html>
```

Output:



My mother has eyes and my father has dark green eyes.

My mother has eyes and my father has dark green eys.

This deprecated tag deprecated

Click Here to go Google

HTML List

List is the collection of items or elements. In HTML list are categorized as:

a. Unordered List
b. Ordered List
c. Description List

Unordered List

The list which are not ordered by numbers are known as unordered list. An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. The list items will be marked with bullets by defaults however it can be changed. To change the bullets an attribute type is added. The value of type are disc,circle,square,none. Like <ul type="cirlce"> The example following shows the unordered list

```
<html>
<body>
<h2>An unordered HTML list </h2>
<ul>
      <li> Coffee </li>
      <li> Tea <li>
      <li> Milk </li>
</ul>
</body>
<lhtml>
```

Output:

An unordered HTML list

- Coffee
- Tea
- Milk

Ordered List

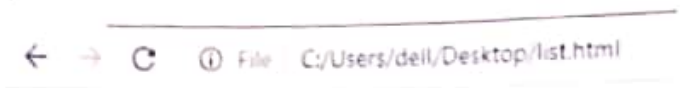
The list which are ordered by numbers are known as ordered list. An ordered list starts with the <ol> tag. Each list item starts with the <li> tag. The list items will be marked with numbers by default. The ordered list has the type attribute of the <ol> tag, defines the type of the list item marker. The following table shows the type attribute with the descriptions

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="i"	The list items will be numbered with lowercase roman numbers
type="I"	The list items will be numbered with uppercase roman numbers

Example: List starting with numbers

```
<html>
<body>
    <ol type="1">
        <li>Nepal</li>
        <li>India</li>
        <li>China</li>
    </ol>
</body>
</html>
```

Output:



1. Nepal
2. India
3. China

Description List

Definition List displays elements in definition form like in dictionary. The <dl>, <dt> and <dd> tags are used to define description list. The 3 HTML description list tags are given below:

1. **<dl> tag** defines the description list.
2. **<dt> tag** defines data term.
3. **<dd> tag** defines data definition (description).

Example description list

```
<html>
<body>
    <dl>
        <dt>HTML</dt>
        <dd>is a markup language</dd>
    </dl>
</body>
</html>
```

Output:

HTML
is a markup language

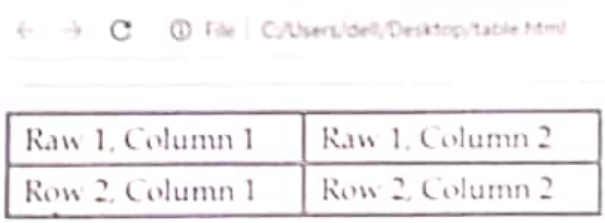
HTML Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.The HTML tables are created using the <table> tag in which the <tr> tag is used to create table rows and <td> tag is used to create data cells. The elements under <td> are regular and left aligned by default

Example:

```
<html>
   <head>
      <title>HTML Tables</title>
   </head>
   <body>
      <table border = "1">
         <tr>
            <td>Row 1, Column 1</td>
            <td>Row 1, Column 2</td>
         </tr>
         <tr>
            <td>Row 2, Column 1</td>
            <td>Row 2, Column 2</td>
         </tr>
      </table>
   </body>
</html>
```

Output:



Here, the border is an attribute of <table> tag and it is used to put a border across all the cells. If you do not need a border, then you can use border = "0".

Table Heading

Table heading can be defined using <th> tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally top row is considered as table heading as shown below, otherwise <th> element is used in any row. Headings, which are defined in <th> tag are centered and bold by default.

Example:

```
<html>
<head>
    <title>HTML Table Header</title>
</head>
<body>
    <table border = "1">
        <tr>
        <th>Name</th>
        <th>Salary</th>
        </tr>
        <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
        </tr>
        <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
        </tr>
    </table>
</body>
</html>
```

Output:

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

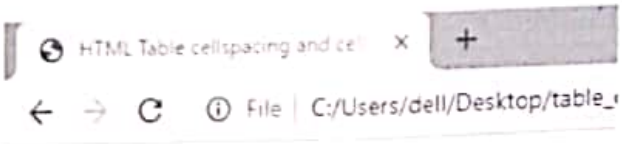
Cellpadding and Cellspacing Attributes

There are two attributes called cellpadding and cellspacing which is used to adjust the white space in the table cells. The cellspacing attribute defines space between table cells, while cellpadding represents the distance between cell borders and the content within a cell.

Example:

```
<html>
<head>
    <title>HTML Table cellspacing and cell padding</title>
</head>
<body>
        <table border = "1" cellpadding = "5" cellspacing = "5">
        <tr>
        <th>Name</th>
        <th>Salary</th>
        </tr>
        <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
        </tr>
        <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
        </tr>
    </table>
</body>
</html>
```

Output:



Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

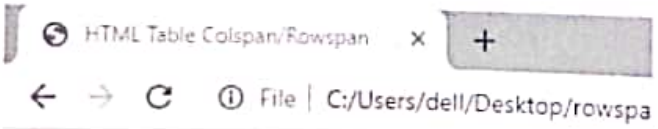
Colspan and Rowspan Attributes

The colspan attribute is used to 7merge two or more columns into a single column. Similar way the rowspan is used to merge two or more rows.

Example:

```
<html>
   <head>
      <title>HTML Table Colspan/Rowspan</title>
   </head>
   <body>
      <table border = "1">
         <tr>
            <th>Column 1</th>
            <th>Column 2</th>
            <th>Column 3</th>
         </tr>
         <tr>
            <td rowspan = "2">Row 1 Cell 1</td>
            <td>Row 1 Cell 2</td>
            <td>Row 1 Cell 3</td>
         </tr>
         <tr>
            <td>Row 2 Cell 2</td>
            <td>Row 2 Cell 3</td>
         </tr>
         <tr>
            <td colspan = "3">Row 3 Cell 1</td>
         </tr>
      </table>
   </body>
</html>
```

Output:



Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Tables Backgrounds

Table background can be set using one of the following two ways –

- bgcolor attribute – Used to set background color for whole table or just for one cell.
- background attribute – Used to set background image for whole table or just for one cell.

To set border color also using bordercolor attribute. Like <table border = "1" bordercolor = "green" bgcolor = "yellow">

Table Height and Width

Table width and height can be adjusted using width and height attributes. The table width or height in terms of pixels or in terms of percentage of available screen area can be specified by height and width attributes.

Like:<table border = "1" width = "400" height = "150">

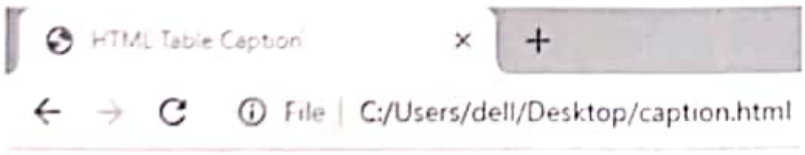
Table Caption

The caption tag will serve as a title or explanation for the table and it shows up at the top of the table.

Example:

```
<html>
   <head>
      <title>HTML Table Caption</title>
   </head>
   <body>
      <table border = "1">
         <caption>This is the caption</caption>
         <tr>
            <td>row 1, column 1</td><td>row 1, columnn 2</td>
         </tr>
         <tr>
            <td>row 2, column 1</td><td>row 2, columnn 2</td>
         </tr>
      </table>
   </body>
</html>
```

Output:



This is the caption

row 1, column 1	row 1, column 1
row 1, column 1	row 1, column 1

Table Header, Body, and Footer

Tables can be divided into three portions – a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are:

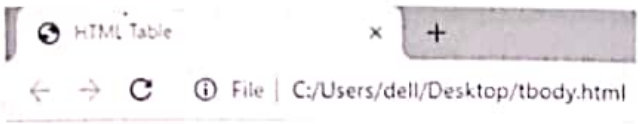
- <thead> – to create a separate table header.
- <tbody> – to indicate the main body of the table.
- <tfoot> – to create a separate table footer.

A table may contain several <tbody> elements to indicate different pages or groups of data. But it is notable that <thead> and <tfoot> tags should appear before <tbody>

Example:

```
<html>
   <head>
      <title>HTML Table</title>
   </head>
   <body>
      <table border = "1">
         <thead>
            <tr>
               <th colspan = "4">This is the head of the table</th>
            </tr>
         </thead>
         <tbody>
            <tr>
               <td>Cell 1</td>
               <td>Cell 2</td>
               <td>Cell 3</td>
               <td>Cell 4</td>
            </tr>
         </tbody>
         <tfoot>
            <tr>
               <th colspan = "4">This is the foot of the table</th>
            </tr>
         </tfoot>
      </table>
   </body>
</html>
```

Output:



This is the head of the table			
Cell 1	Cell 2	Cell 3	Cell 4
This is the foot of the table			

HTML Frames

HTML frames are used to divide the browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns. The disadvantages of using frames are:

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back* button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page <frameset> tag instead of <body> tag is used. The <frameset> tag defines, how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Important: You cannot use the <body></body> tags together with the<frameset></frameset> tags. However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags.

Example:

```
<html>
   <head>
      <title>HTML Frames</title>
   </head>
   <frameset rows = "10%, 80%, 10%">
      <frame name = "top" src = "/html/top_frame.htm" />
      <frame name = "main" src = "/html/main_frame.htm" />
      <frame name = "bottom" src = "/html/bottom_frame.htm" />
      <noframes>
         <body>Your browser does not support frames.</body>
      </noframes>
   </frameset>
</html>
```

HTML Forms

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML <form> tag is used to create an HTML form and it has following syntax:

```
<form>
    //input elements
</form>
```

Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Sr.No.	Attribute & Description
1	**action** Backend script ready to process your passed data.
2	**method** Method to be used to upload data. The most frequently used are GET and POST methods.
3	**target** Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
4	**enctype** You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are – application/x-www-form-urlencoded – This is the standard method most forms use in simple scenarios. mutlipart/form-data – This is used when you want to upload binary data in the form of files like image, word file etc.

Syntax:

```
<form action="index.html" method= "post">
</form>
```

Difference between GET and POST in method attribute of form

GET	POST
Parameters remain in browser history because they are part of the URL	Parameters are not saved in browser history.
Can be bookmarked.	Can not be bookmarked.
GET requests are re-executed but may not be re-submitted to server if the HTML is stored in the browser cache.	The browser usually alerts the user that data will need to be re-submitted.
Application/x-www-form-urlencoded	Multipart/form-data or application/x-www-form-urlencoded Use multipart encoding for binary data.
Can send but the parameter data is limited to what we can stuff into the request line (URL). Safest to use less than 2K of parameters, some servers handle up to 64K	Can send parameters, including uploading files, to the server.
Easier to hack for script kiddies	More difficult to hack
Yes, only ASCII characters allowed.	No restrictions. Binary data is also allowed.
GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history and server logs in plaintext.	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs.
Yes, since form data is in the URL and URL length is restricted. A safe URL length limit is often 2048 characters but varies by browser and web server.	No restrictions
GET method should not be used when sending passwords or other sensitive information.	POST method used when sending passwords or other sensitive information.
GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.	POST method variables are not displayed in the URL.
Data can be cached	Data cannot be cached

HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Text Input Controls

There are three types of text input used on forms –

- Single-line text input controls – This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.
- Password input controls – This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML<input> tag.
- Multi-line text input controls – This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

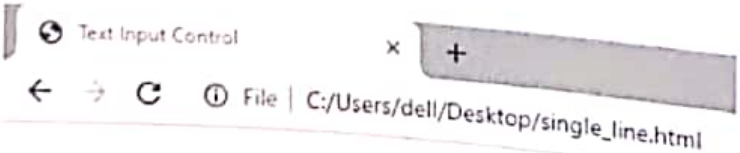
Single-line text input controls

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Example:

```
<html>
   <head>
      <title>Text Input Control</title>
   </head>
   <body>
      <form >
         First name: <input type = "text" name = "first_name" />
         <br>
         Last name: <input type = "text" name = "last_name" />
      </form>
   </body>
</html>
```

Output:



First name:

Last name:

Attributes

Following is the list of attributes for <input> tag for creating text field.

S.No	Attribute & Description
1	type Indicates the type of input control and for text input control it will be set to text.
2	name Used to give a name to the control which is sent to the server to be recognized and get the value.
3	value This can be used to provide an initial value inside the control.
4	size Allows to specify the width of the text-input control in terms of characters.
5	maxlength Allows to specify the maximum number of characters a user can enter into the text box.

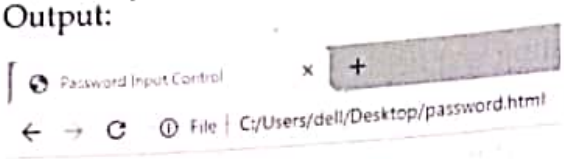
Password input controls

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input>tag but type attribute is set to password.

Example:

```
<html>
   <head>
      <title>Password Input Control</title>
   </head>
   <body>
      <form >
         User ID : <input type = "text" name = "user_id" />
         <br>
         Password: <input type = "password" name = "password" />
      </form>
   </body>
</html>
```

Output:



User ID: 12

Password: •••••••••

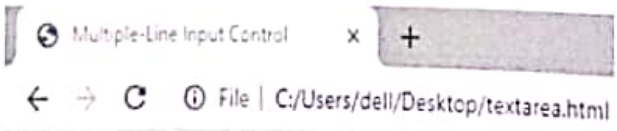
Multiple-Line Text Input Controls

The most commonly used form tag is <input> tag. The type of input is specified with the type attribute within the <input> tag.

Example:

```
<html>
    <head>
        <title> Multiple-Line Input control </title>
    <head>
    <body>
        description : <br/>
        <textarea rows / = "S" cols = "50" name = description>
           Entra description here . . .
           </textarea>
    </form>
</body>
</html>
```

Output:



Description:

Enter description here . . .

Attributes used in multiple line

Sr.No	Attribute & Descriptiona
1	**name:** It is Used to give a name to the control which is sent to the server to be recognized and get the value.
2	**rows:** It is Indicates the number of rows of text area box.
3	**cols:** It Indicates the number of columns of text area box

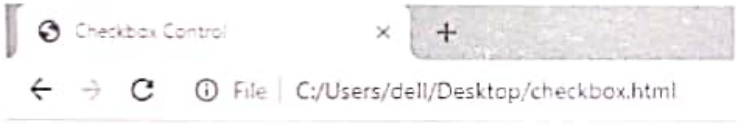
Checkbox Control

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to checkbox.

Example:

```
<html>
   <head>
      <title>Checkbox Control</title>
   </head>
   <body>
      <form>
         <input type = "checkbox" name = "maths" value = "M"> Maths
         <input type = "checkbox" name = "physics" value = "P"> Physics
      </form>
   </body>
</html>
```

Output:



☐ Maths ☐ Physic

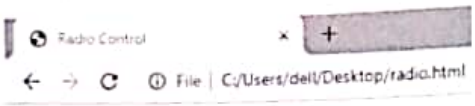
Radio Button Control

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to radio. The name attribute in radio button control must be same in every input tag which is shown in example below

Example:

```
<html>
   <head>
      <title>Radio Control</title>
   </head>
   <body>
      <form>
         <input type = "radio" name = "gender" value = "male"> Male
         <input type = "radio" name = "gender" value = "female"> Female
      </form>
   </body>
</html>
```

Output:



o Male o Female

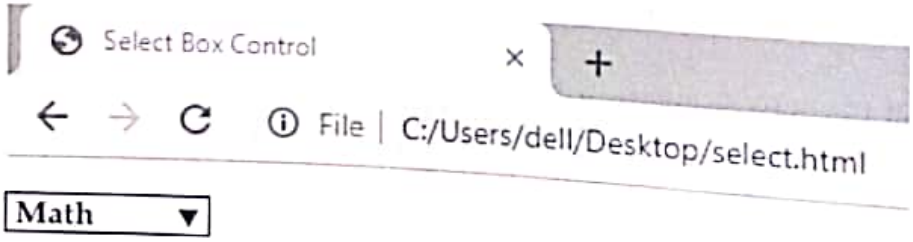
Select Box Control

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Example:

```
<html>
   <head>
      <title>Select Box Control</title>
   </head>
   <body>
      <form>
         <select name = "dropdown">
            <option value = "Maths" selected>Maths</option>
            <option value = "Physics">Physics</option>
         </select>
      </form>
   </body>
</html>
```

Output:



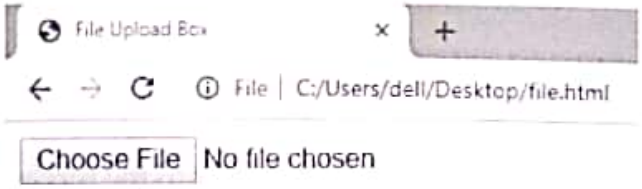
File Upload Box

To upload a file to in the web site file upload box is needed, also known as a file select box. This is also created using the <input> element but type attribute is set to file.

Example:

```
<html>
   <head>
      <title>File Upload Box</title>
   </head>
   <body>
      <form>
         <input type = "file" name = "fileupload" accept = "image/*" />
      </form>
   </body>
</html>
```

Output:



Button Controls

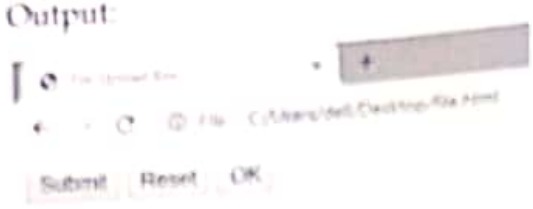
There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input>tag by setting its type attribute to button. The type attribute can take the following values:

Sr.No	Type & Description
1	**submit** This creates a button that automatically submits a form.
2	**reset** This creates a button that automatically resets form controls to their initial values.
3	**button** This creates a button that is used to trigger a client-side script when the user clicks that button.
4	**image** This creates a clickable button but we can use an image as background of the button.

Example

```
<html>
   <head>
      <title>File Upload Box</title>
   </head>
   <body>
      <form>
         <input type = "submit" name = "submit" value = "Submit" />
         <input type = "reset" name = "reset"  value = "Reset" />
         <input type = "button" name = "ok" value = "OK" />
      </form>
   </body>
</html>
```

Output:



When the user clicks on the "Submit or Ok" button, the content of the form is sent to the ser... The form's **action attribute** defines the name of the file to send the content to. The file defined... the action attribute usually does something with the received input. For example,

```
<form name="input" action=" submit.php" method="get">
 Username:
 <input type="text" name="user" />
 <input type="submit" value="Submit" />
</form>
```

If you type some characters in the text field above, and click the "Submit" button, the brow... will send your input to a page called "submit.php". The page will show you the received inp...

The Reset button resets the input field.

The **method** attribute of <form> tag specifies how to send form-data (the form-data is sen... the page specified in the action attribute). We can use **"get"** and **"post"** as values of meth... attribute. When we use get, form-data can be sent as URL variables and when we use po... form-data are sent as HTTP post.

Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- There is a limit to how much data you can place in a URL (varies between browsers... therefore, you cannot be sure that all of the form-data will be correctly transferred
- Never use the "get" method to pass sensitive information! (password or other sensitiv... information will be visible in the browser's address bar)

Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

ID attributes of HTML elements

the ID attribute in HTML is a unique identifier for the element. It provides a way to identify an area of a web page for CSS styles, anchor links, and targets for scripts.

What is the ID attribute used for?

The ID attribute can perform several actions for web pages:

- **A style sheet selector**: This is the function most people use the ID attribute for. Becaus... they are unique, you can be sure you'll be styling just the one item on your web page... when you style using an ID property. The downside to using an ID for styling purposes... is that it has a very high level of specificity, which can make it very challenging if yo...

need to override a style for some reason later in a stylesheet. Because of this, current web practices lean toward using classes and class selectors in place of IDs and ID selectors for general styling purposes.

- **Named anchors for linking to**: web browsers allow you to target precise locations in your web documents by pointing to the ID at the end of the URL. You simply add the id to the end of the page URL, preceded by a pound-sign (#). You can also link to these anchors with the page itself by adding the pound-sign (#) and the ID name in the href attribute for the element. For instance, if you have a division with an ID of contact, you could link to it on that page with #contact.
- **A reference for scripts**: If you write any Javascript functions, you will want to use the ID attribute so that you can make changes to the precise element on the page with your scripts.
- **Other processing**: The id allows you to process your web documents in whatever way you need to. For example, you might extract the HTML into a database, and the ID attribute identifies fields.

Rules for using the ID attribute

There are a few rules you must follow to have a valid document that uses the id attribute anywhere in the document:

- The ID must start with a letter (a-z or A-Z)
- All subsequent characters can be letters, numbers (0-9), hyphens (-), underscores (_), colons (:), and periods (.).
- Each ID must be unique within the document.

Syntax:

```
<element id="id_name">
```

The element may be of any types of HTML elements discussed previous

Class Attributes of HTML Elements

The class attribute specifies one or more classnames for an element.The class attribute is mostly used to point to a class in a style sheet. However, it can also be used by a JavaScript (via the HTML DOM) to make changes to HTML elements with a specified class.

Syntax:

```
<element class="class_name">
```

The element may be of any types of HTML elements discussed previous

HTML Color

HTML colors are displayed using RED, GREEN, and BLUE light. Colors are defined using hexadecimal (hex) notation for combination of red, green, and blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00) and the highest values is 255 (hex FF). We can use HEX (e.g. **#2000FF**) as well as RGB (e.g. **rgb(32, 0, 255)**) values to define different colors.

The combination of Red, Green and Blue values from 0 to 255 gives a total of more than 16 million different colors to play with (256 x 256 x 256).

We can also use color names instead of hex and rgb values. The World Wide Web Consortium (W3C) has listed 16 valid color names for HTML and CSS: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. Some examples are given below:

```
<body style = "background:rgb(12, 32, 255)">
<body style = "background:#0008FF>
<body style = "background:red">
```

HTML Fonts

The **<font>** tag in HTML is deprecated. It is supposed to be removed in a future version of HTML. For example,

```
<p>
 <font size="2" face="Verdana" color = "red">
  This is a paragraph.
 </font>
</p>
```

HTML Character Entities

Character entities are replaced with reserved characters. A character entity looks *&entity_name* OR*&#entity_number*. Some commonly used character entities are:

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	Ampersand	&amp;	&#38;
¢	Cent	&cent;	&#162;
£	Pound	&pound;	&#163;
¥	Yen	&yen;	&#165;
€	Euro	&euro;	&#8364;
©	Copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

HTML Head

The head element contains general information, also called meta-information, about a document. The elements inside the head element should not be displayed by a browser. According to the HTML standard, only a few tags are legal inside the head section. These are: <base>, <link>, <meta>, <title>, <style>, and <script>.You must use this element and it should be used just once. It must start immediately after the opening <html> tag and end directly before the opening <body> tag.

HTML Meta

HTML includes a meta element that goes inside the head element. The purpose of the meta element is to provide meta-information about the document. Meta elements are purely used for search engine's use and to provide some additional information about your pages. We use three attributes (**name, content**, and **http-equiv**) with **<meta>** tag.

We use **name = "keywords"** to provide information for a search engine. If the keywords you have chosen are the same as the ones they have put in, you come up in the search engine's result pages. For example,

<meta name="keywords" content="HTML, DHTML, CSS, XML, XHTML, JavaScript" />

We use **name = "description"** to define a description of your page. It is sort summary of the content of the page. Depending on the search engine, this will be displayed along with the title of your page in an index. For example,

<meta name="description" content="Free Web tutorials on HTML, CSS, XML, and XHTML" />

We use **name = "generator"** to define a description for the program you used to write your pages. For example,

<meta name="generator" content="Homesite 4.5" />

We use **name = "author"** and **name = "copyright"** for author and copyright details. For example,

<meta name="author" content="W3schools" />

<meta name="copyright" content="W3schools 2005" />

We use **name = "expires"** to give the browsers a data, after which the page is deleted from the browsers cache, and must be downloaded again. This is useful if you want to make sure your visitors are reading the most current version of a page. For example,

<meta name="expires" content="13 July 2008" />

We use **http-equiv= "expires"** to refresh itself to the most current version or change to another location (page) entirely after some time. This is useful if you've moved a page to a new url and want any visitors to the old address to be quietly sent to the new location. For example,

<meta http-equiv = "refresh" content="5;url=http://www.tu.edu.np" />

Here, the number is the number of seconds to wait before changing to the new page.

Setting it to 0 results in an instant redirect.

HTML Audio

Since the release of HTML5, audios can be added to webpages using the "audio" tag. Previously audios could be only played on webpages using web plugins like Flash. The "audio" tag is an inline element which is used to embed sound files into a web page. It is a very useful tag if you want to add audio such as songs, interviews, etc on your webpage.

Syntax:

```
<audio>
<source src="sample.mp3" type="audio/mpeg">
</audio>
```

Attributes

The various attributes that can be used with the "audio" tag are listed below :

- Controls : Designates what controls to display with the audio player.
- Autoplay : Designates that the audio file will play immediately after it loads controls.
- Loop : Designates that the audio file should continuously repeat.
- src : Designates the URL of the audio file.
- muted : Designates that the audio file should be muted.

Supported Formats

Three formats mp3, ogg, wav are supported by HTML5

Example

```
<html>
<body>
<audio controls>

  <source src="national.ogg" type="audio/ogg">
  <source src="song.mp3" type="audio/mpeg">

Your browser does not support the audio element.
</audio>
</body>
</html>
```

Output:

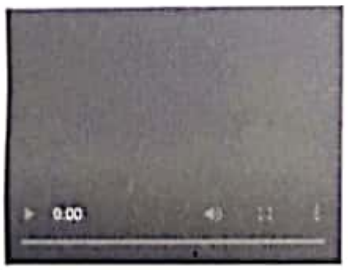
HTML Video

The HTML5 <video> element specifies a standard way to embed a video in a web page.

Example:

```
<html>
<body>
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
</body>
</html>
```

Output:



HTML Canvas

The HTML <canvas> element is used to draw graphics on a web page. The graphic to the left is created with <canvas>. It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

What is HTML Canvas?

The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript. The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics. Canvas has several methods for drawing paths, boxes, circles, text, and adding images. A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

Syntax:

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

Example:

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

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
Your browser does not support the HTML5 canvas tag.
</canvas>

</body>
</html>
```

Output:

Example: Drawing a circle using canvas

```
<html>
<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>

</body>
</html>
```

Output:

HTML Main

The <main> tag specifies the main content of a document. The content inside the <main> element should be unique to the document. It should not contain any content that is repeated across documents such as sidebars, navigation links, copyright information, site logos, and search forms.

Syntax:

<main>
//statements
</main>

Example:

```
<html>
<body>

<main>
  <h1>Web Browsers</h1>
  <p>Google Chrome, Firefox, and Internet Explorer are the most used browsers today.</p>
  <article>
    <h1>Google Chrome</h1>
    <p>Google Chrome is a free, open-
source web browser developed by Google, released in 2008.</p>
  </article>
  <article>
    <h1>Internet Explorer</h1>
    <p>Internet Explorer is a free web browser from Microsoft, released in 1995.</p>
  </article>
  <article>
    <h1>Mozilla Firefox</h1>
    <p>Firefox is a free, open-source web browser from Mozilla, released in 2004.</p>
  </article>
</main>
<p><strong>Note:</strong> The main tag is not supported in Internet Explorer 11 and earl
ier versions.</p>
</body>
</html>
```

Output:

Web Browser

Google Chrome, Firefox, and Internet explorer are the most used browser today.

Google Chrome

Google Chrome is a free, open-source web browser developed by Google, released in 2008.

Internet Explore

Internet Explorer is a free web browser from Microsoft, released in 1995

Mozilla Firefiox

Firefox is a free, open-source web browser from Mozilla, released in 2004

Note: The main tag is not supported in Internet Explorer 11 and earlier versions.

HTML section

The <section> tag defines sections in a document, such as chapters, headers, footers, or any other sections of the document.

Syntax:

```
<section>
//statements
</section>
```

Example:

```
<html>
<body>
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is an international organization
working on issues regarding the conservation, research and restoration of the
environment, formerly named the World Wildlife Fund. WWF was founded in 1961.<
/p>
</section>
<p><strong>Note:</strong> The section tag is not supported in Internet Explore
r 8 and earlier versions.</p>
</body>
</html>
```

Output:

WWF

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

Note: The section tag is not supported in Internet Explore 8 and earlier versions.

HTML Article

The <article> tag specifies independent, self-contained content.An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.

Syntax:

```
<article>
//statements
</article>
```

Example:

```
<html>
<body>
<article>
  <h1>Google Chrome</h1>
  <p>Google Chrome is a free, open-
source web browser developed by Google, released in 2008.</p>
</article>
<p><strong>Note:</strong> The article tag is not supported in Internet Explore
r 8 and earlier versions.</p>
</body>
</html>Z
```

Output:

Google Chrome

Google Chrome is a free, open-source web browser develop by Google, released in 2008

Note: the article tag is not supported an internet Explore 8 and earlier versions.

HTML Header

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

A <header> element typically contains:

- one or more heading elements (<h1> - <h6>)
- logo or icon
- authorship information

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

Example:

```
<html>
<body>
<article>
  <header>
    <h1>Most important heading here</h1>
    <h3>Less important heading here</h3>
    <p>Some additional information here.</p>
  </header>
  <p>This is the heading paragraph</p>
</article>
</body>
</html>
```

Output:

Most important heading here

Less important heading here

Some additional information here

This is the heading paragraph

HTML Footer

The <footer> tag defines a footer for a document or section.A <footer> element should contain information about its containing element.

A <footer> element typically contains:

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

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

Example:

```
<html>
<body>
<footer>
  <p>Posted by: Web Tech</p>
  <p>Contact information: <a href="mailto:someone@example.com">web@web.com</a>
.</p>
</footer>
<p><strong>Note:</strong> The footer tag is not supported in Internet Explorer
 8 and earlier versions.</p>
</body>
</html>
```

Output:

Posted by : Web Tech

Contact information: Web@web.com

Note: The footer tag is not supported in Internet Explorer 8 and earlier versions.

HTML Aside

The <aside> tag defines some content aside from the content it is placed in.The aside content should be related to the surrounding content.

Example:

```
<html>
<body>
<p>My family and I visited The Epcot center this summer.</p>
<aside>
  <h4>Epcot Center</h4>
  <p>The Epcot Center is a theme park in Disney World, Florida.</p>
</aside>
<p><strong>Note:</strong> The aside tag is not supported in Internet
Explorer 8 and earlier versions.</p>
</body>
</html>
```

Output:

My family and I visited The Epcot center this summer.

Epcot Center

The Epcot Center is a theme park in Disney World, Florida.

Note: The aside tag is not supported in Internet Explorer\ 8 and earlier versions.

HTML Nav

The <nav> tag defines a set of navigation links.

Example:

```
<html>
<body>
<nav>
<a href="/web/">Web</a> |
<a href="/C++/">C++</a> |
<a href="/Microprocesspor/">Microprocesspor</a>
</nav>
<p><strong>Note:</strong> The nav tag is not supported in Internet Explorer 8
and earlier versions.</p>

</body>
</html>
```

Output:

Web | C++ | Microprocesspor

Note: The nav tag is not supported in Internet Explorer 8 and earlier vesions.

HTML Figure Tags

The <figure> tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.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 should not affect the flow of the document.

Example:

```
<html>
<body>
<p>Nepal has a history of nine decades of science teaching/learning at the hig
her level. Several schemes, plans, policies and implementation strategies have
 been experimented in science education. This institute has come to the presen
t form after facing many challenges and preparing scientifically oriented tech
nical manpower of different levels to meet the requirements of the nation. It
is capable to compete with universities at regional and global levels in terms
 of curriculum and pedagogic standards.
</p>
<figure>
  <img src="tu.gif" alt="The Pulpit Rock" width="304" height="228">
</figure>
<p><strong>Note:</strong> The figure tag is not supported in Internet Explorer
 8 and earlier versions.</p>

</body>
</html>
```

Output:

Nepal has a history of nine decades of science teaching learning at the higher level. Several schemes, plans, policies and implementation strategies have been experimented in science education. This institute has come to the present from after facing many challenges and preparing scientifically oriented technical manpower of different level to meet the requirements of the nation. It is capable to compete with universities at regional and global levels in terms of curriculum and pedagogies standards

Note: this figure tag is not supported in Internet Explore 8 and earlier versions.

HTML Div

The ** ** element defines logical divisions within the document. When you use a element, you are indicating that the enclosed content is specific section of the page and you can format the section with CSS (Cascading Style Sheet). For example,

```
<div                    style="background-color:orange;text-
 align:center"><p>Navigation section</p>

</div>

<div style="border:1px solid black">

 <p>Content section</p>

</div>
```

HTML Events

Events trigger actions in the browser, like starting a JavaScript when a user clicks on an HTML element. Below is a list of attributes that can be inserted to HTML tags to define event actions. These HTML events are given below

Window Events (Only valid in body and frameset elements)

Attribute	Value	Description
Onload	*Script*	Script to be run when a document loads
Onunload	*Script*	Script to be run when a document unloads

Form Element Events (Only valid in form elements)

Attribute	Value	Description
Onchange	*Script*	Script to be run when the element changes
Onsubmit	*Script*	Script to be run when the form is submitted
Onreset	*Script*	Script to be run when the form is reset
Onselect	*script*	Script to be run when the element is selected
Onblur	*script*	Script to be run when the element loses focus
Onfocus	*script*	Script to be run when the element gets focus

Keyboard Events (Not valid in *base, bdo, br, frame, frameset, head, html, iframe, meta,param, script, style,* and *title* elements)

Attribute	Value	Description
Onkeydown	*script*	What to do when key is pressed
Onkeypress	*script*	What to do when key is pressed and released
Onkeyup	*script*	What to do when key is released

Mouse Events (Not valid in *base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, title* elements)

Attribute	Value	Description
Onclick	*script*	What to do on a mouse click
Ondblclick	*script*	What to do on a mouse double-click
Onmousedown	*script*	What to do when mouse button is pressed
Onmousemove	*script*	What to do when mouse pointer moves
Onmouseout	*Script*	What to do when mouse pointer moves out of an element
Onmouseover	*Script*	What to do when mouse pointer moves over an element
Onmouseup	*script*	What to do when mouse button is released

WAP to create a form containing text field for name, email, password, roll number, faculty, gender, hobbies, description and a submit button and clear button

Discussion Exercise

1. Difference between HTML and XHTML.
2. Write History of HTML?
3. What is HTML attribute? Describe HTML elements with its types.
4. What is Tag in HTML? Describe different types of tags in HTML.
5. Describe the use of hyperlink tag
6. Create a basic HTML table with cell padding, cellspacing, rowspan, colspan and border attribute
7. Create a basic form that includes text field for name, number field for age, a field for salary, a select dropdown, radio button for gender, checkbox for hobbies and a text area for description.
8. Discuss HTML events with examples

□□□

3

Chapter

CASCADING STYLE SHEETS

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Introduction
- Cascading Style Sheets (CSS)
- CSS Syntax
- Interesting CSS: Inline, Internal, External, ID and Class Sectors; Colors, Background, Border, Text, Text, Font, List, Table, CSS Box Model
- Normal Flow Box Layout: Basic Box Layout, Display Property, Padding
- Margin Positioning: Relative, float, Absolute: CSS3 Borders, Box Shadows, Text Effects and Shadow
- Basics of Respective Web Designs, Media Queries, Introduction to Bootstrap

CSS (Cascading Style Sheets)

CSS stands for cascading style sheets. It was first developed in 1997, as a way for Web developers to define the look and feel of their Web pages. It was intended to allow developers to separate content from design and layout so that HTML could perform more of the function without worry about the design and layout. It is used to separate style from content. It is used to control the style of a web document in a simple and easy way.

Importance of CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. The following are some of the key advantages of learning CSS:

- Create Stunning Web site - CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.
- Become a web designer - If you want to start a carrier as a professional web designer, HTML and CSS designing is a must skill.
- Control web - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
- Learn other languages - Once you understand the basic of HTML and CSS then other related technologies like JavaScript, php, or angular are become easier to understand.

Applications of CSS

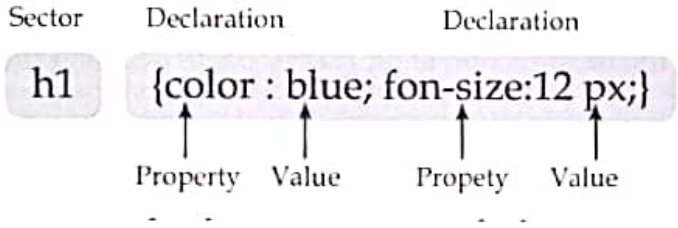
As mentioned before, CSS is one of the most widely used style language over the web. The following are the list few of them here:

- **CSS saves time:** We can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
- **Pages load faster:** If we are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- Easy maintenance - To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- Superior styles to HTML - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.

- Multiple Device Compatibility - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- Global web standards - Now HTML attributes are being deprecated and it is being recommended to use CSS. So it's a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

CSS Syntax

A CSS rule has two main parts: a *selector* and one or more *declarations*. **Selector** is normally the HTML element you want to style and each **declaration** consists of a *property* and *value*. The **property** is the style attribute we want to use and each property has a **value** associated with it.



Example:

```
p {color:red;text-align:center;}
```

Here in this example p refers to the paragraph and the color red is assign to the paragraph with the text align center

Inserting CSS or Types of CSS

We can use style sheets in three different ways in out HTML document. They are **external style sheet, internal style sheet** and **inline style**.

External Style Sheet

If we want to apply the same style to many pages, we use external style sheet. With an external style sheet, you can change the look of an entire Web site by changing one style sheet file. Each page must link to the style sheet using the **<link>** tag. The <link> tag goes inside the head section.

Example:

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

An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension. An example of a style sheet file is shown below:

```
hr {
      color:sienna;
      }
p {
     margin-left:20px;
     }
/*Note: Do not leave space between property value and units*/
body {
      background-image:url("images/back40.gif");
```

Internal Style Sheet

If you want a unique style to a single document, an internal style sheet should be used. You define internal styles in the head section of an HTML page, by using the **<style>** tag.

Example:

```
<head>
   <style type="text/css">
      hr {
         color:red;
         }
      p {
         margin-left:20px;
         }
      body {
         background-image:url("images/back40.gif");
         }
   </style>
</head>
```

Inline Styles

If you want a unique style to a single element, an inline style sheet should be used. An inline style loses many of the advantages of style sheets by mixing content with presentation. To use inline styles you use the **style attribute** in the relevant tag. The style attribute can contain any CSS property.

Example:

```
<p style="color:yellow;margin-left:20px">This is a paragraph.</p>
```

Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers. A CSS comment begins with "/*", and ends with "*/".

Example:

```
<style>
   /* body {
      background-image:url("images/back40.gif");
      } */
</style>
```

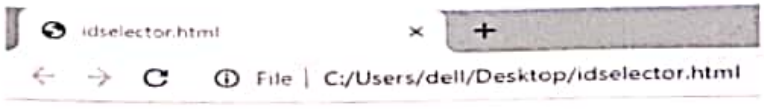
Id and Class Selectors

The id selector is used to specify a style for a single, unique element. The id selector uses id attribute of the HTML element and is defined with "#".

Example:

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

Output:



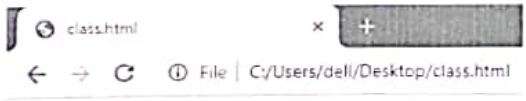
Hello World!
This paragraph is not affected by style

The class selector is used to specify a style for a group of elements. Unlike the id selector, the class selector is most often used on several elements. This allows you to set a particular style for any HTML elements with the same class. The class selector uses the HTML class attribute, and is defined with (dot symbol) ".".

Example:

```
<html>
<head>
   <style type="text/css">
   .para
   {
      text-align:left;
      color:red;
   }
   </style>
</head>
   <body>
      <p class="para">Hello World!</p>
      <p>This paragraph is not affected by the style.</p>
   </body>
</html>
```

Output:

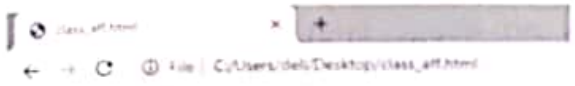


Hello World!
This paragraph is not affected by style
You can also specify that only specific HTML elements should be affected by a class.

Example:

```
<html>
    <head>
        <style type="text/css">
            p.center
            {
            color:red;
            }
        </style>
    </head>
    <body>
        <h1 class="center">This heading will not be affected</h1>
        <p class="center">The font color is red.</p>
    </body>
</html>
```

Output:



This heading will not be affected

The font colour is red

Multiple Styles Will Cascade into One

Styles can be specified:

- inside an HTML element
- inside the head section of an HTML page
- in an external CSS file

Cascading order

What style will be used when there is more than one style specified for an HTML element?

Generally speaking, we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

- Browser default
- External style sheet
- Internal style sheet (in the head section)
- Inline style (inside an HTML element)

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

Note: If the link to the external style sheet is placed after the internal style sheet in HTML<head>, the external style sheet will override the internal style sheet!

CSS Background

Background properties are used to define the background effects of an HTML element. CSS properties used to define background effects are: **background-color**, **background-image**, **background-repeat**, **background-attachment**, and **background-position**.

Background Image

The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.

The background image for a page can be set like this:

```
body
{
    background-image:url('paper.gif');
}
```

Background Image - Repeat Horizontally or Vertically

By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example:

```
body
{
    background-image:url('gradient2.png');
}
```

If the image is repeated only horizontally (repeat-x), the background will look better:

Example:

```
body
{
    background-image:url('gradient2.png');
    background-repeat:repeat-x;
}
```

Background Image - Set position and no-repeat

When using a background image, use an image that does not disturb the text. Showing the image only once is specified by the background-repeat property:

Example

```
body
{
    background-image:url('img_tree.png');
    background-repeat:no-repeat;
}
```

In the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much.

The position of the image is specified by the background-position property:

Example:

```
body
{
    background-image:url('img_tree.png');
    background-repeat:no-repeat;
    background-position:right top;
}
```

Shorthand Property

To shorten the code, it is also possible to specify all the properties in one single property. This is called a shorthand property. The shorthand property for background is simply "background". When using the shorthand property,the order of the property valuesare: background-color, background-image, background-repeat, background-attachment, and background-position.

Example:

```
body
{
    background:#ffffff url('img_tree.png') no-repeat right top;
}
```

Grouping Selectors

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

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

To minimize the code, you can group selectors. Separate each selector with a comma. In the example below we have grouped the selectors from the code above:

Example

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

CSS Borders

The CSS border properties allows to specify how the border of the box representing an element should look. There are three properties of a border that can be changed:

- border-color: specifies the color of a border.
- border-style: specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
- border-width: specifies the width of a border.

The border-color Property

The border-color property allows you to change the color of the border surrounding an element. You can individually change the color of the bottom, left, top and right sides of an element's border using the properties –

- border-bottom-color: changes the color of bottom border.
- border-top-color: changes the color of top border.
- border-left-color: changes the color of left border.
- border-right-color: changes the color of right border.

Example:

```
<html>
   <head>
      <style type = "text/css">
         p.example1 {
            border:1px solid;
            border-bottom-color:#009900; /* Green */
            border-top-color:#FF0000;    /* Red */
            border-left-color:#330000;   /* Black */
            border-right-color:#0000CC;  /* Blue */
         }
         p.example2 {
            border:1px solid;
            border-color:#009900;        /* Green */
         }
      </style>
   </head>
   <body>
      <p class = "example1">
         This example is showing all borders in different colors.
      </p>
      <p class = "example2">
         This example is showing all borders in green color only.
      </p>
   </body>
</html>
```

Output:

This example is showing all boarders in different colors.

This example is showing all boarders in green color only

The border-style Property

The border-style property allows you to select one of the following styles of border:

- none: No border. (Equivalent of border-width:0;)
- solid: Border is a single solid line.
- dotted: Border is a series of dots.
- dashed: Border is a series of short lines.
- double: Border is two solid lines.
- groove: Border looks as though it is carved into the page.
- ridge: Border looks the opposite of groove.
- inset: Border makes the box look like it is embedded in the page.
- outset: Border makes the box look like it is coming out of the canvas.
- hidden: Same as none, except in terms of border-conflict resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using the following properties:

- border-bottom-style: changes the style of bottom border.
- border-top-style: changes the style of top border.
- border-left-style: changes the style of left border.
- border-right-style: changes the style of right border.

Example:

```
<html>
   <head>
       <title>Border Style</title>
   </head>
   <body>
      <p style = "border-width:1px; border-style:none;">
         I have no border
      </p>
      <p style = "border-width:1px; border-style:solid;">
         I have a solid border
      </p>
      <p style = "border-width:1px; border-style:dashed;">
         I have a dashed border.
      </p>
      <p style = "border-width:1px; border-style:double;">
         I have a double border.
      </p>
      <p style = "border-width:1px; border-style:groove;">
         I have a groove border.
      </p>
      <p style = "border-width:1px; border-style:ridge">
         I have a ridge  border.
      </p>
```

```
      <p style = "border-width:1px; border-style:inset;">
         I have a inset border.
      </p>
      <p style = "border-width:1px; border-style:outset;">
         I have a outset border.
      </p>
      <p style = "border-width:1px; border-style:hidden;">
         I have a hidden border.
      </p>
      <p style = "border-width:4px;
         border-top-style:solid;
         border-bottom-style:dashed;
         border-left-style:groove;
         border-right-style:double;">
         I have four different styles of borders.
      </p>
   </body>
</html>
```

Output:

I have no boarder

I have a solid boarder

I have a dashed boarder

I have double boarder.

I have groove boarder

I have a ride boarder

I have outset border

I have a hidden boarder

The border-width Property

The border-width property allows to set the width of an element borders. The value of this property could be either a length in px, pt or cm or it should be set to thin, medium or thick.

You can individually change the width of the bottom, top, left, and right borders of an element using the following properties:

- border-bottom-width changes the width of bottom border.
- border-top-width changes the width of top border.
- border-left-width changes the width of left border.
- border-right-width changes the width of right border.

Example:

```
<html>
   <head>
      <title>Border Width</title>
   </head>
   <body>
      <p style = "border-width:4px; border-style:solid;">
         I have a border whose width is 4px.
      </p>
      <p style = "border-width:4pt; border-style:solid;">
         I have a border whose width is 4pt.
      </p>
      <p style = "border-width:thin; border-style:solid;">
         I have a border whose width is thin.
      </p>
      <p style = "border-width:medium; border-style:solid;">
         I have a border whose width is medium;
      </p>
      <p style = "border-width:thick; border-style:solid;">
         I have a border whose width is thick.
      </p>
      <p style = "border-bottom-width:4px;border-top-width:10px;
         border-left-width: 2px;border-right-width:15px;border-style:solid;">
         I have a border with four different width.
      </p>
   </body>
</html>
```

Output:

I have a boarder whose width is 4px

I have a boarder whose width is 4px

I have a boarder whose width is thin

I have a boarder whose width is medium

I have a boarder whose width is thick

I have a border with four different width.

CSS Text

Text can be manipulate using CSS properties. The following are the text properties of an element:

- The **color** property is used to set the color of a text.
- The **direction** property is used to set the text direction.
- The **letter-spacing** property is used to add or subtract space between the letters that make up a word.
- The **word-spacing** property is used to add or subtract space between the words of a sentence.
- The **text-indent** property is used to indent the text of a paragraph.
- The **text-align** property is used to align the text of a document.
- The **text-decoration** property is used to underline, overline, and strikethrough text.
- The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.
- The **white-space** property is used to control the flow and formatting of text.
- The **text-shadow** property is used to set the text shadow around a text.

Some of the properties are listed in the example below:

```
<html>
   <head>
   </head>
   <body>
        <p style = "color:red;">
            My color will be red
        </p>
        <p style = "text-align:right;">
            I will be right aligned.
        </p>
        <p style = "text-align:center;">
            I will be center aligned.
        </p>
        <p style = "text-align:left;">
            I will be left aligned.
        </p>
   </body>
</html>
```

Output:

My color will be red

I will beleft aligned

I will be center aligned

I will be right aligned

CSS Fonts

To set the fonts of a contents in HTML using CSS, the following font properties are used:

- The **font-family** property is used to change the face of a font.
- The **font-style** property is used to make a font italic or oblique.
- The **font-variant** property is used to create a small-caps effect.
- The **font-weight** property is used to increase or decrease how bold or light a font appears.
- The **font-size** property is used to increase or decrease the size of a font.
- The **font** property is used as shorthand to specify a number of other font properties.

Font-family

The font family of a text is set with the font-family property. The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on. More than one font family is specified in a comma-separated list.

Example

```
p {
  font-family: "Times New Roman", Times, serif;
 }
```

Font-style

The font-style property is mostly used to specify italic text. This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

Example

```
<html>
    <head>
        <style>
2        p.normal {
        font-style: normal;
        }
        p.italic {
        font-style: italic;
        }
        p.oblique {
        font-style: oblique;
        }
    </style>
    </head>
    <body>
    <p class="normal">I am in normal style.</p>
    <p class="italic">I am in italic style.</p>
    <p class="oblique">I am in oblique style.</p>
    </body>
</html>
```

Output:

I am in normal style.

I am in intalic style

I am in oblique style

Font-variant

The font-variant property specifies whether or not a text should be displayed in a small-caps font. In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appear in a smaller font size than the original uppercase letters in the text.

Example:

```
<html>
<head>
    <style>
        p.normal {
        font-variant: normal;
        }
        p.small {
        font-variant: small-caps;
        }
    </style>
</head>
<body>
    <p class='normal'>My name is John. </p>
    <p class='small'>My name is John. </p>
</body>
</html>
```

Output

My name is John

MY NAME IS JOHN

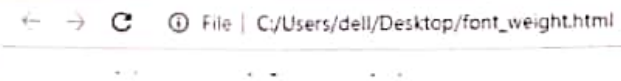
Font-weight

The font-weight property specifies the weight of a font.

Example:

```
<html>
<head>
    <style>
        p.normal {
        font-weight: normal;
        }
        p.light {
        font-weight: lighter;
        }
        p.thick {
        font-weight: bold;
        }
        p.thicker {
        font-weight: 900;
        }
    </style>
</head>
<body>
    <p class="normal">I am with normal font weight</p>
    <p class="light">I am with light font weight</p>
    <p class="thick">I am with thick font weight</p>
    <p class="thicker">I am with thicker font weight</p>
</body>
</html>
```

Output:



I ma with normal font weight

I am with light font weight

I am with thick font weight

I am with thicker font weight

Font-size

The font-size property sets the size of the text. Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs. Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs. The font-size value can be an absolute, or relative size.

Absolute Size

- Sets the text to a specified size
- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size

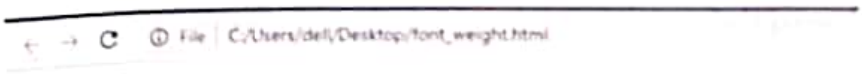
- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers

Font size can be set with pixels also that gives the full control over the text size

Example:

```
<html>
<head>
    <style>
        h1 {
        font-size: 40px;
        }
        h2 {
        font-size: 30px;
        }
        p {
        font-size: 14px;
        }
    </style>
</head>
<body>
    <h1>I am with font size 40px</h1>
    <h2>I am with font size 30px</h2>
    <p>I am with font size 14px</p>
    <p>I am with font size 14px</p>
</body>
</html>
```

Output:



I am with font size 40 px
I am with font size 30 px
I am with font size 14 px
I am with font size 14 px

CSS List

Lists are very helpful in conveying a set of either numbered or bullet points. This chapter teaches you how to control list type, position, style, etc., using CSS.The CSS list properties allows to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

We have the following five CSS properties, which can be used to control lists:

- The **list-style-type** allows to control the shape or appearance of the marker.
- The **list-style-position** specifies whether a long point that wraps to a second line should align with the first line or start underneath the start of the marker.
- The **list-style-image** specifies an image for the marker rather than a bullet point or number.
- The **list-style** serves as shorthand for the preceding properties.
- The **marker-offset** specifies the distance between a marker and the text in the list.

The list-style-type property

The list-style-type property allows to control the shape or style of bullet point (also known as a marker) in the case of unordered lists and the style of numbering characters in ordered lists.

The following are the values used for unordered list:

S.No	Value and Description
1	None
2	Disc(default): A filled in circle
3	Circle: An empty circle
4	Square: A filled in square

The following are some of the values used for ordered list:

Value	Description	Example
decimal	Number	1,2,3,4,5
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V

Example:

```
<html>
   <head>
   </head>
   <body>
      <ul style = "list-style-type:circle;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ul style = "list-style-type:square;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ol style = "list-style-type:decimal;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ol>
      <ol style = "list-style-type:lower-alpha;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ol>
      <ol style = "list-style-type:lower-roman;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ol>
   </body>
</html>
```

Output:

- Maths
- Social Science
- Physics
- Maths
- Social Science
- Physics

1. Maths
2. Social Science
3. Physics

a. Maths
b. Social science
c. Physics

i. Maths
ii. social Science
iii. Physics

The list-style-position Property

The list-style-position property indicates whether the marker should appear inside or outside of the box containing the bullet points. It can have one the two values:

Sr.No.	Value & Description
1	none
2	inside
	If the text goes onto a second line, the text will wrap underneath the marker. It will also appear indented to where the text would have started if the list had a value of outside.
3	outside
	If the text goes onto a second line, the text will be aligned with the start of the first line (to the right of the bullet).

Example:

```
<html>
   <head>
   </head>
   <body>
      <ul style = "list-style-type:circle; list-stlye-position:outside;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ul style = "list-style-type:square;list-style-position:inside;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ol style = "list-style-type:decimal;list-stlye-position:outside;">
```

```
            <li>Maths</li>
            <li>Social Science</li>
            <li>Physics</li>
        </ol>
        <ol style = "list-style-type:lower-alpha;list-style-position:inside;">
            <li>Maths</li>
            <li>Social Science</li>
            <li>Physics</li>
        </ol>
    </body>
</html>
```

Output:

- Maths
- Social Science
- Physics

  - Maths
  - Social Science
  - Physics

1. Maths
2. Social Science
3. Physics
   a. Maths
   b. Social science
   c. Physics

The list-style-image Property

The list-style-image allows you to specify an image so that you can use your own bullet style. The syntax is similar to the background-image property with the letters url starting the value of the property followed by the URL in brackets. If it does not find the given image, then default bullets are used.

Example:

```
<html>
    <head>
    </head>
    <body>
        <ul>
            <li style = "list-style-image: url(/images/bullet.gif);">Maths</li>
            <li>Social Science</li>
            <li>Physics</li>
        </ul>
        <ol>
            <li style = "list-style-image: url(/images/bullet.gif);">Maths</li>
            <li>Social Science</li>
            <li>Physics</li>
        </ol>
    </body>
</html>
```

Output:

- Maths
- Social Science
- Physics
  - Maths

2. Social Science
3. Physics

The list-style Property

The list-style allows to specify all the list properties into a single expression. These properties can appear in any order.

Example:

```
<html>
   <head>
   </head>
   <body>
      <ul style = "list-style: inside square;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ol style = "list-style: outside upper-alpha;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ol>
   </body>
</html>
```

Output:

- Maths
- Social Science
- Physics

A. Maths
B. Social Science
C. Physics

The marker-offset Property

The marker-offset property allows to specify the distance between the marker and the text relating to that marker. Its value should be a length as shown in the following example

Example:

```
<html>
   <head>
   </head>
   <body>
      <ul style = "list-style: inside square; marker-offset:2em;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ul>
      <ol style = "list-style: outside upper-alpha; marker-offset:2cm;">
         <li>Maths</li>
         <li>Social Science</li>
         <li>Physics</li>
      </ol>
   </body>
</html>
```

Output:

- Maths
- Social Science
- Physics

A. Maths
B. Social Science
C. Physics

CSS Table

The look of HTML table can be improved with CSS. The following are the properties of table used in css:

- The **border-collapse** specifies whether the browser should control the appearance of the adjacent borders that touch each other or whether each cell should maintain its style.
- The **border-spacing** specifies the width that should appear between table cells.
- The **caption-side** captions are presented in the <caption> element. By default, these are rendered above the table in the document. You use the caption-side property to control the placement of the table caption.
- The **empty-cells** specifies whether the border should be shown if a cell is empty.
- The **table-layout** allows browsers to speed up layout of a table by using the first width properties it comes across for the rest of a column rather than having to load the whole table before rendering it.

The border-collapse

This property can have two values collapse and separate

Example:

```
<html>
   <head>
      <style type = "text/css">
         table.one {border-collapse:collapse;}
         table.two {border-collapse:separate;}
         td.a {
            border-style:dotted;
            border-width: 3px;
            border-color: #000000;
            padding: 10px;
         }
         td.b {
            border-style: solid;
            border-width: 3px;
            border-color: #333333;
            padding: 10px;
         }
      </style>
   </head>
   <body>
      <table class = "one">
         <caption>Collapse Border Example</caption>
         <tr><td class = "a">Cell A Collapse Example</td></tr>
         <tr><td class = "b">Cell B Collapse Example</td></tr>
      </table>
      <br />
      <table class = "two">
         <caption>Separate Border Example</caption>
         <tr><td class = "a">Cell A Separate Example</td></tr>
         <tr><td class = "b">Cell B Separate Example</td></tr>
      </table>
   </body>
</html>
```

Output

Collapse Border Example

Cell A Collapse Example

Cell A Collapse Example

Separate Border Example

Cell A Separate example

Cell B Separate example

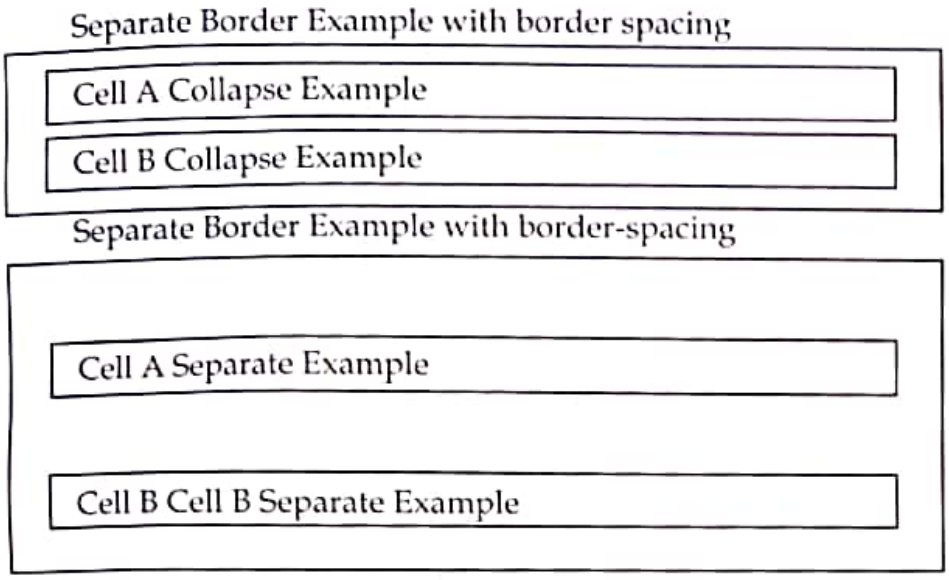
The Border-spacing Property

The border-spacing property specifies the distance that separates adjacent cells'. borders. It can take either one or two values; these should be units of length. If there is only one value, it will apply to both vertical and horizontal borders. And if there are two values, in this case, the first refers to the horizontal spacing and the second to the vertical spacing:

Example:

```
<html>
   <head>
      <style type = "text/css">
         table.one {
            border-collapse:separate;
            width:400px;
            border-spacing:10px;
         }
         table.two {
            border-collapse:separate;
            width:400px;
            border-spacing:10px 50px;
         }
      </style>
   </head>
   <body>
      <table class = "one" border = "1">
         <caption>Separate Border Example with border-spacing</caption>
         <tr><td> Cell A Collapse Example</td></tr>
         <tr><td> Cell B Collapse Example</td></tr>
      </table>
      <br />
      <table class = "two" border = "1">
         <caption>Separate Border Example with border-spacing</caption>
         <tr><td> Cell A Separate Example</td></tr>
         <tr><td> Cell B Separate Example</td></tr>
      </table>
   </body>
</html>
```

Output:



The Caption-side Property

The caption-side property allows you to specify where the content of a <caption> element should be placed in relationship to the table. The table that follows lists the possible values. This property can have one of the four values top, bottom, left or right. The following example uses each value.

The Empty-cells Property

The empty-cells property indicates whether a cell without any content should have a border displayed. This property can have one of the three values - show, hide or inherit.

The Table-layout Property

The table-layout property is supposed to help you control how a browser should render or lay out a table. This property can have one of the three values: fixed, auto or inherit. The following example shows the difference between these properties.

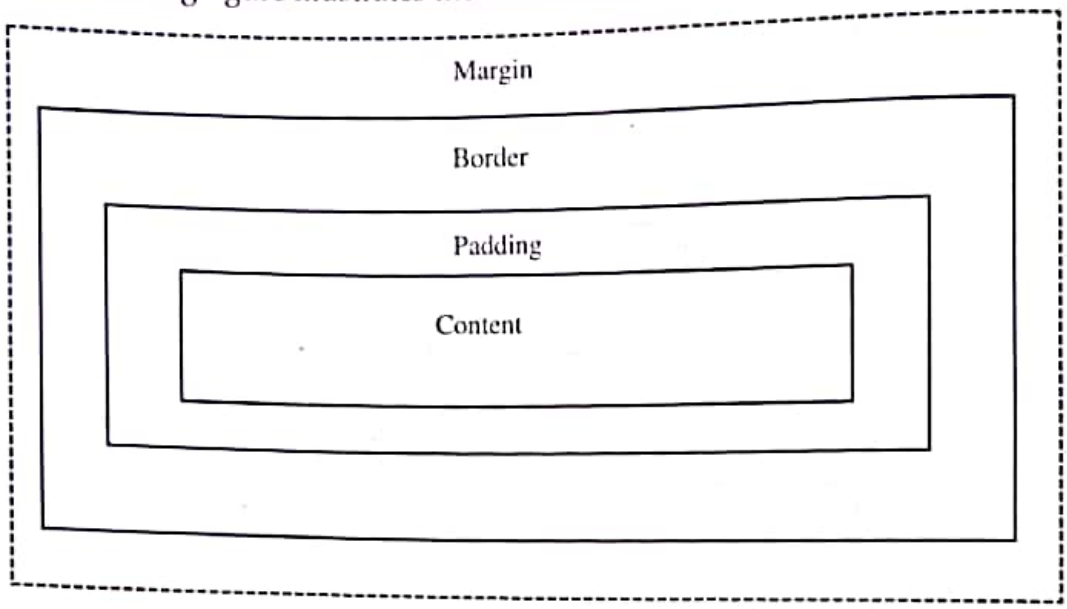
CSS Box Model

CSS box model is a container which contains multiple properties including borders, margin, padding and the content itself. It is used to create the design and layout of web pages. It can be used as a toolkit for customizing the layout of different elements. The web browser renders every element as a rectangular box according to the CSS box model.

Box-Model has multiple properties in CSS. Some of them are given below:

- borders
- margins
- padding
- Content

The following figure illustrates the box model.



- **Border Area**: It is the area between the box's padding and margin. Its dimensions are given by the width and height of border.
- **Margin Area**: This area consists of space **between** border and margin. The dimensions of Margin area are the margin-box width and the margin-box height. It is useful to separate the element from its neighbors.
- **Padding Area**: It includes the element's padding. This area is actually the space around the content area and within the border box. Its dimensions are given by the width of the padding-box and the height of the padding-box.
- **Content Area**: This area consists of content like text, image, or other media content. It is bounded by the content edge and its dimensions are given by content box width and height.

Box Layout

The width and height attribute are used to create the box layout in HTML using CSS which are illustrated in the example below

Example:

```
<html>
<head>
    <style>
    div {
    background-color: none;
    width: 300px;
    border: 1px solid ;
    }
    </style>
</head>
<body>
    <div>This is normal layout</div>
</body>
</html>
```

Output:

This is normal layout

Example of box using padding :

```
<html>
   <head>
      <style>
         .div1 {
            width: 300px;
            height: 100px;
            border: 1px solid blue;
         }
         .div2 {
            width: 300px;
            height: 100px;
            padding: 50px;
            border: 1px solid red;
         }
      </style>
   </head>

   <body>
      <div class = "div1">I am box 1</div><br />
      <div class = "div2">I am box 2</div>
   </body>
</html>
```

Output:

I am box 1

I am box 2

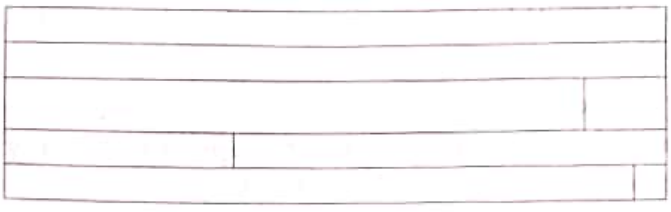
Example

Box using margin, padding, float, width

```
<html>
<head><title>example</title>
<style type="text/css">
.top{
        width:100%;
        border-top:1px solid;
        border-right:1px solid;
        border-left:1px solid;
        border-bottom:1px solid;
}
.box1{
        padding:20px;
}
.mid{
         width:100%;
         border-top:1px solid;
}
.box2{
    padding:20px;
}
.middle{
           width:100%;
           border-top:1px solid;
}
.box3{
            width:80%;
            float:left;
            border-right:1px solid;
            padding:30px;
}
.box4{
           padding:30px;
}
.slastbox{
           width:100%;
           border-top:1px solid;
}
.box5{
        width:30%;
        float:left;
        border-right:1px solid;
        padding:20px;
}
.box6{
```

```
        padding:20px;
}
.lastbox{
        width:100%;
        border-top:1px solid;
}.box7{
      width:90%;
      float:left;
      border-right:1px solid;
      padding:20px;
}
.box8{
        padding:20px;
}
</style>
</head>
<body>
<div class="top">
 <div class="box1"></div>
<div class="mid">
 <div class="box2"></div>
 <div class="middle">
  <div class="box3"></div>
  <div class="box4"></div>
<div class="slastbox">
 <div class="box5"></div>
 <div class="box6"></div>
<div class="lastbox">
 <div class="box7"></div>
 <div class="box8"></div>
 </body>
 </html>
```

Output:



CSS Display Properties

The display property affects the most basic presentation of an element, effectively classing the element as a certain type of element. The rendering of the element may depend heavily on its display type, and certain properties will only work on elements that have specific display values.

The following are the possible values for display properties

- **inline**: This value causes an element to generate an inline-level box; for example, the HTML elements STRONG, CODE, or EM (among others). The element will generate one or more inline boxes when it is displayed.
- **block**: This value causes an element to generate a block-level box; for example, the HTML elements P, H1, or PRE (among others). The element will generate a block box when it is displayed.
- **list-item**: This value causes an element to generate both a block box and a list-item inline box. In HTML, the LI element is the only example of such an element.
- **run-in**: Under certain conditions, this value will cause the element to be insertedinto the beginning of the following element. If an element A is set to display: run-in and is followed by a block-level element B, then A becomes the first inline-level box of B. If the element following A is not block-level, then A becomes a block-level box.
- **compact**: Under certain conditions, this value will cause the element to be placed to one side of the following element.
- **marker:** This value will set generated content to be a marker; thus, it should be used only in conjunction with the :before and :after pseudo- elements when they are set on block-level elements.
- **table:** This value causes an element to generate a block-level table box. This is analogous to the HTML element TABLE.
- inline-table: This value causes an element to generate an inline-level table box. While there is no analogue in HTML, it can be envisioned as a traditional HTML table which can appear in the middle of a line of text.
- **table-cell**: This value declares the element to be a table cell. This is analogous to the HTML element TD.
- **table-row:** This value declares the element to be a row of table cells. This is analogous to the HTML element TR.
- **table-row-group**: This value declares the element to be a group of table rows. This is analogous to the HTML element TBODY.
- **table-column**: This value declares the element to be a column of table cells. This is analogous to the HTML element COL.
- **table-column-group**: This value declares the element to be a group of table columns. This is analogous to the HTML element COLGROUP.
- **table-header-group**: This value declares the element to be a group of cells which is always visible at the top of the table, placed before any row or row-groups but after any top-aligned table captions. This is analogous to the HTML element THEAD.
- **table-footer-group**: This value declares the element to be a group of cells which is always visible at the bottom of the table, placed after any row or row-groups but before any bottom-aligned table captions. This is analogous to the HTML element TFOOT.
- **table-caption**: This value declares the element to be a caption for a table. This is analogous to the HTML element CAPTION.
- **none**: The element will generate no boxes at all, and thus will neither be displayed nor impact the layout of the document.

Example:

```
<html>
<head><title>display</title>
</head>
<body>
    <p style = "display:inline;">
        This paragraph will inline with the next paragraph
    </p>
    <p style = "display:inline;">
        and will make a single line.
    </p>
    <br />
    <br />
    <div style = "display:block;">
        This paragraph will be separate from the next paragraph
    </div>
    <div style = "display:block;">
        and this is second paragraph.
    </div>
 </body>
 </html>
```

Output:

This paragraph will inline with the next paragraph
and will make a single line.
This paragraph will be separate from the next paragraph
and this is section paragraph.

CSS Padding

The padding property allows to specify how much space should appear between the content of an element and its border. The value of this attribute should be either a length, a percentage, or the word inherit. If the value is inherit, it will have the same padding as its parent element. If a percentage is used, the percentage is of the containing box.

The following CSS properties can be used to control lists. The different values for the padding on each side of the box can be set using the following properties:

- The **padding-bottom** specifies the bottom padding of an element.
- The **padding-top** specifies the top padding of an element.
- The **padding-left** specifies the left padding of an element.
- The **padding-right** specifies the right padding of an element.
- The **padding** serves as shorthand for the preceding properties.

Example:

```
<html>
   <head><title>Css paddidng</title>
   </head>
   <body>
    <p style = "padding-bottom: 15px; border:1px solid black;">
         This is a paragraph with a specified bottom padding
    </p>
    <p style = "padding-left: 15px; border:1px solid black;">
           This is a paragraph with a specified left padding
    </p>
    <p style = "padding-right: 15px; border:1px solid black;">
    This is a paragraph with a specified right padding
    </p>
    <p style = "padding: 15px; border:1px solid black;">
    all four padding will be 15px
    </p>
    <p style = "padding:10px 2%; border:1px solid black;">
    top and bottom padding will be 10px, left and right
    padding will be 2% of the total width of the document.
    </p>
    <p style = "padding: 10px 2% 10px; border:1px solid black;">
    top padding will be 10px, left and right padding will
    be 2% of the total width of the document, bottom padding will be 10px
    </p>
    <p style = "padding: 10px 2% 10px 10px; border:1px solid black;">
    top padding will be 10px, right padding will be 2% of
    the total width of the document, bottom padding and top padding will be 1
px
    </p>
   </body>
</html>
```

Output:

This is a paragraph with a specified bottom padding

This is a paragraph with a specified left padding

This is a paragraph with a specified right padding

all four padding will be 15 px

top and bottom padding will be 10 px, left and right padding will be 2% of the total width of the document

top padding will be 10 px, left and right padding will be 2% of the total width the document, bottom padding will be 10 px

top padding will be 10 px, right padding will be 2% of the total width of the document, bottom padding and to padding will be 10 px

CSS Margin

The margin property defines the space around an HTML element. It is possible to use negative values to overlap content.

The values of the margin property are not inherited by the child elements. Remember that the adjacent vertical margins (top and bottom margins) will collapse into each other so that the distance between the blocks is not the sum of the margins, but only the greater of the two margins or the same size as one margin if both are equal.

The following are the properties to set an element margin.

- The **margin** specifies a shorthand property for setting the margin properties in one declaration.
- The **margin-bottom** specifies the bottom margin of an element.
- The **margin-top** specifies the top margin of an element.
- The **margin-left** specifies the left margin of an element.
- The **margin-right** specifies the right margin of an element.

Example

```
<html>
<head>
<style>
p {
    background-color: green;
    color:white;
}
p.ex {
    margin-top: 50px;
    margin-bottom: 50px;
    margin-right: 100px;
    margin-left: 100px;
}
</style>
</head>
<body>
<p>This paragraph is not displayed with specified margin. </p>
<p class="ex">This paragraph is displayed with specified margin.</p>
</body>
</html>
```

Output:

This Paragraph is not displayed with specified margin.

This paragraph is displayed with specified margin.

CSS Float

With CSS float, an element can be pushed to the left or right, allowing other elements to wrap around it. Float is very often used for images, but it is also useful when working with layouts.

How Elements Float

Elements are floated horizontally; this means that an element can only be floated left or right, not up or down. A floated element will move as far to the left or right as it can. Usually this means all the way to the left or right of the containing element. The elements after the floating element will flow around it. The elements before the floating element will not be affected. If an image is floated to the right, a following text flows around it, to the left.

Example

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

Floating Elements Next to Each Other

If you place several floating elements after each other, they will float next to each other if there is room. Here we have made an image gallery using the float property:

Example

```
.thumbnail
{
float:left;
width:110px;
height:90px;
margin:5px;
}
```

Turning off Float - Using Clear

Elements after the floating element will flow around it. To avoid this, use the clear property. The clear property specifies which sides of an element other floating elements are not allowed. Add a text line into the image gallery, using the clear property:

Example

```
.text_line
{
clear:both;
}
```

CSS Positioning

CSS helps to position the HTML element. The HTML element can be put at whatever location. You can specify whether you want the element positioned relative to its natural position in the page or absolute based on its parent element.
The following are the positioning related properties:

Relative Positioning

Relative positioning changes the position of the HTML element relative to where it normally appears. So "left:20" adds 20 pixels to the element's LEFT position.
You can use two values top and left along with the position property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top.

Example:

```
<html>
   <head>
   </head>
   <body>
      <div style = "position:relative; left:80px; top:2px; background-
color:yellow;">
         I have relative positioning.
      </div>
   </body>
</html>
```

Output:

I have relative positioning

Absolute Positioning

An element with position: absolute is positioned at the specified coordinates relative to your screen top-left corner. You can use two values top and left along with the position property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top.

Example

```
<html>
   <head>
   </head>
   <body>
      <div style = "position:absolute; left:80px; top:20px; background-
color:yellow;">
         This div has absolute positioning.
      </div>
   </body>
</html>
```

Output

This div has absolute positioning.

Fixed Positioning

Fixed positioning allows you to fix the position of an element to a particular spot on the page, regardless of scrolling. Specified coordinates will be relative to the browser window. You can use two values top and left along with the position property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for left.
- Move Right - Use a positive value for left.
- Move Up - Use a negative value for top.
- Move Down - Use a positive value for top

Example

```
<html>
   <head>
   </head>
   <body>
      <div style = "position:fixed; left:80px; top:20px; background-
color:yellow;">
         This div has fixed positioning.
      </div>
   </body>
</html>
```

Output:

This div has fixed positioning

CSS Box Shadows

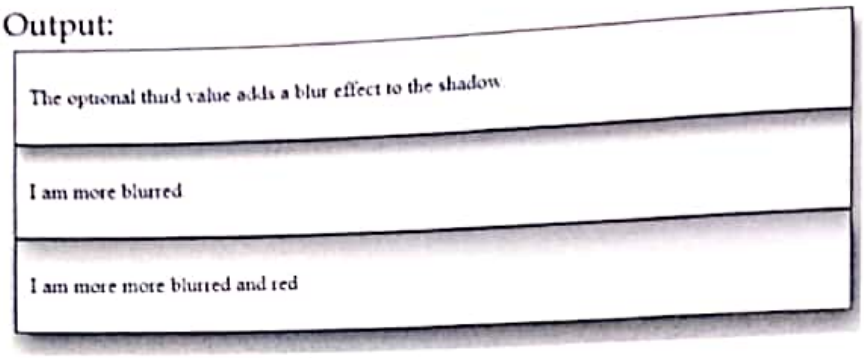
The box-shadow property attaches one or more shadows to an element. To attach more than one shadow to an element, add a comma-separated list of shadow

Syntax:
box-shadow: none | h-offset v-offset blur spread color | inset | initial | inherit;

Example:

```
<html>
<head>
<style>
#example1 {
  border: 1px solid;
  padding: 10px;
  box-shadow: 5px 10px 8px #888888;
}
#example2 {
  border: 1px solid;
  padding: 10px;
  box-shadow: 5px 10px 18px #888888;
}
#example3 {
  border: 1px solid;
  padding: 10px;
  box-shadow: 5px 10px 18px red;
}
</style>
</head>
<body>
<div id="example1">
  <p>The optional third value adds a blur effect to the shadow.</p>
</div>
<div id="example2">
  <p>I am more blurred.</p>
</div>
<div id="example3">
  <p>I am more more blurred and red.</p>
</div>
</body>
</html>
```

Output:



CSS3 Text Effects and Shadows

CSS3 supported to add shadow effects to text.

Example:

```
<html>
   <head>
      <style>
         h1 {
            text-shadow: 2px 2px;
         }
         h2 {
            text-shadow: 2px 2px red;
         }
         h3 {
            text-shadow: 2px 2px 5px red;
         }
         h4 {
            color: white;
            text-shadow: 2px 2px 4px #000000;
         }
         h5 {
            text-shadow: 0 0 3px #FF0000;
         }
         h6 {
            text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
         }
         p {
            color: white;
            text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
         }
      </style>
   </head>
   <body>
      <h1>CSS is fun</h1>
      <h2>CSS is fun</h2>
      <h3>CSS is fun</h3>
      <h4>CSS is fun</h4>
      <h5>CSS is fun</h5>
      <h6>CSS is fun</h6>
      <p>CSS is fun</p>
   </body>
</html>
```

Output

CCS is fun

CSS is fun

CSS is fun

CSS is fun

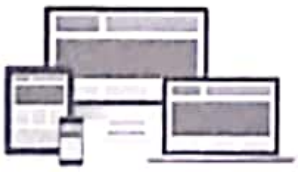
CSS is fun

CSS is fun

Basics of Responsive Web Designs

Responsive web design provides an optimal experience, easy reading and easy navigation with a minimum of resizing on different devices such as desktops, mobiles and tabs).

Responsive Structure

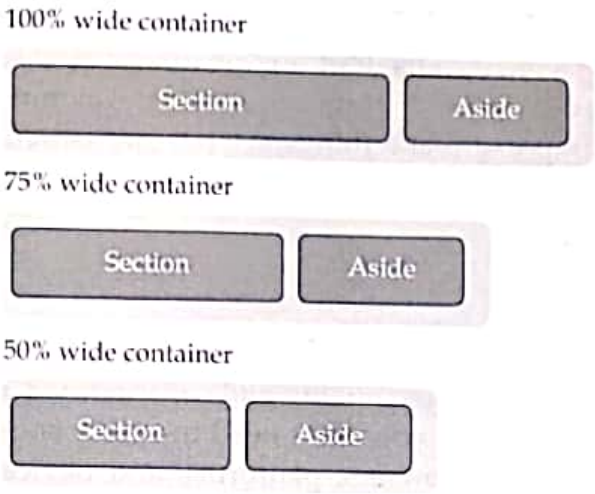


Example

```
<html>
   <head>
      <style>
         body {
            font: 600 14px/24px "Open Sans",
               "HelveticaNeue-Light",
               "Helvetica Neue Light",
               "Helvetica Neue",
               Helvetica, Arial,
               "Lucida Grande",
               Sans-Serif;
         }
         h1 {
            color: #9799a7;
            font-size: 14px;
            font-weight: bold;
            margin-bottom: 6px;
         }
         .container:before, .container:after {
            content: "";
            display: table;
         }
         .container:after {
            clear: both;
         }
```

```
        .container {
            background: #eaeaed;
            margin-bottom: 24px;
            *zoom: 1;
        }
        .container-75 {
            width: 75%;
        }
        .container-50 {
            margin-bottom: 0;
            width: 50%;
        }
        .container, section, aside {
            border-radius: 6px;
        }
        section, aside {
            background: #2db34a;
            color: #fff;
            margin: 1.858736059%;
            padding: 20px 0;
            text-align: center;
        }
        section {
            float: left;
            width: 63.197026%;
        }
        aside {
            float: right;
            width: 29.3680297%;
        }
    </style>
</head>
<body>
    <h1>100% Wide Container</h1>
    <div class = "container">
        <section>Section</section>
        <aside>Aside</aside>
    </div>
    <h1>75% Wide Container</h1>
    <div class = "container container-75">
        <section>Section</section>
        <aside>Aside</aside>
    </div>
    <h1>50% Wide Container</h1>
    <div class = "container container-50">
        <section>Section</section>
        <aside>Aside</aside>
    </div>
</body>
</html>
```

Output:



Media Queries

Media queries is for different style rules for different size devices such as mobiles, desktops, etc. For media queries we use @media screen in css

Example

```
<html>
   <head>
      <style>
         body {
            background-color: lightpink;
         }
         @media screen and (max-width: 420px) {
            body {
               background-color: lightblue;
            }
         }
      </style>
   </head>
   <body>
      <p>
         If screen size is less than 420px, then it will show lightblue
  color, or else it will show light pink color
      </p>
   </body>
</html>
```

Output

If screen size is less than 420 px, then it will show lightblue color, or else it will show light pink color

Introduction to Bootstrap

Bootstrap is a free and open-source framework for creating websites and web applications. It's the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Bootstrap can make things a whole lot easier. Bootstrap enables you to create responsive websites without you needing to do the "responsive" bit. Bootstrap takes care of that.

Advantages of Bootstrap

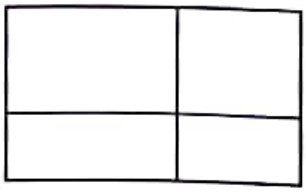
One of the main benefits of development frameworks like Bootstrap is that they can help speed up development times, while maintaining quality and consistency across the site. You no longer need to re-design every element. And you don't need to spend hours trying to get everything looking and working right across browsers, platforms, and devices. By using Bootstrap, all (most) of the hard work is done for you.

Given Bootstrap is the most popular frontend development framework on the web, this skillset could be a useful one to learn. Adding Bootstrap to your bag of tricks could help you in many ways from building websites faster, to landing your dream job.

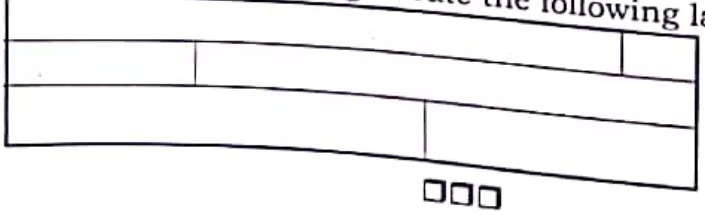
Also, although Bootstrap comes with its own set of styles, these are easy to override. You're not locked into the "Bootstrap design". You are free to use whichever Bootstrap components you choose, while adding your own on top. There are thousands of websites out there that are built on Bootstrap, but with their own design.

Discussion Exercise

1. What is CSS? Describe different types of CSS with example.
2. What is CSS Selector? Describe different types of CSS selector with example.
3. Explain CSS Model with examples.
4. List the types of CSS with examples
5. Without using table tag write



6. Write HTML and CSS code to generate the following layout (with using divs)



4

Chapter

CLIENT SIDE SCRIPTING WITH JAVASC RIPT

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Structure of Java Script Program
- Variable and Data Types
- Statements: Expression Keyword, Block, Operators
- Flow, Controls, Looping, Function, Popup Boxes: Alert, Confirm, Prompt, Object and Properties, Constructors
- Array, Built-in Object, Event handling and Form Validation, Error Handling Cookies, JQuery Syntax
- JQuery Selectors, Events and Effect
- Introduction to JSON

Introduction

Javascript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled. JavaScript renders web pages in an interactive and dynamic fashion. This allowing the pages to react to events, exhibit special effects, accept variable text, validate data, create cookies, detect a user's browser, etc.

There are two ways to use JavaScript in an HTML file. The first one involves embedding all the JavaScript code in the HTML code, while the second method makes use of a separate JavaScript file that's called from within a Script element, i.e., enclosed by Script tags. JavaScript files are identified by the .js extension. Although JavaScript is mostly used to interact with HTML objects, it can also be made to interact with other non-HTML objects such as browser plugins, CSS (Cascading Style Sheets) properties, the current date, or the browser itself.

Structure of Javascript Program

JavaScript is written in plain text. All the Javascript program are written inside the <script type="text/javascript"> tag. The <script type="text/javascript"> tag tells the browser that the code inside this tag are Javascript program and browser render accordingly.

The <script> tag is written inside the <body> tag of the HTML page. JavaScript supports both C-style and C++-style comments, thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.-->
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Variable and Data Types

Variables are referred as named containers for storing information. We can place data into these containers and then refer to the data simply by naming the container. Here are the important rules that must be followed while declaring a variable in JavaScript.

No particular character identifies a variable in JavaScript as the dollar sign does in PHP. Instead, variables use the following naming rules:

- A variable may include only the letters a-z, A-Z, 0-9, the $ symbol, and the underscore (_).
- No other characters, such as spaces or punctuation, are allowed in a variable name.
- The first character of a variable name can be only a-z, A-Z, $, or _ (no numbers).
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.

However, var keyword is used to define variable as follows:

Example

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

String Variables

JavaScript string variables should be enclosed in either single or double quotation marks, like this:

Example:

```
<scripttype="text/javascript">>
    sub= 'microprocessor';
    program= "csit";
</script>
```

Numeric Variables

Creating a numeric variable is as simple as assigning a value

Example:

```
<scripttype="text/javascript">>
    mark = 42;
    percentage = 42;
</script>
```

The following functions are used in numbers in javascript

- **parseInt()**:The parseInt() function parses a string and returns an integer.

  Syntax:

  parseInt(string)

- **parseFloat()**: The parseFloat() function parses a string and returns a floating point number.

  Syntax:

  parseFloat(string)

Example:

```
<html>
<head>
<html>
<head>
<script type="text/javascript">
   var ten = parseInt("10");
   document.write(ten + " is a integer <br>");
   var a = parseFloat("10.25");
   document.write(a + " is a float");
</script>
</head>
</body>
</html>
```

Output:

10 is integer

10.25 is a float

- **eval():** This function evaluates or executes an arguments

  Syntax:

  eval(string)

Example:

```
<html>
<head>
    <script>
          var x = 10;
          var y = 20;
          var a = eval("x * y") + "<br>";
          var b = eval("2 + 2") + "<br>";
          var c = eval("x + 17") + "<br>";

          var res = a + b + c;
          document.write(res);
        </script>
</head>
</body>
</html>
```

Output:

200

4

27

Variables can be initialized at time of declaration or after declaration as follows

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

Data Types:

The following are the data types of Javascript

- String: Can contain groups of character as single value. It is represented in double quotes. E.g. var x= "Hello".
- Numbers: Contains the numbers with or without decimal. E.g. var x=44, y=44.56;
- Booleans: Contain only two values either true or false. E.g. var x=true, y= false.
- Undefined: Variable with no value is called Undefined. E.g. var x;
- Null: If we assign null to a variable, it becomes empty. E.g. var x=null;
- Array: Can contain groups of values of same type. E.g. var x={1,2,3,55};
- Objects: Objects are stored in property and value pair. E.g. var rectangle = { length: 5, breadth: 3};

Statements

In a computer language, a group of words, numbers, and operators that performs a specific task is a statement. But in JavaScript, a statement might look as follows:

var a = var b * 2;

The characters a and b are called **variables** which are like simple boxes you can store any of your stuff in. In programs, variables hold values (like the number 42) to be used by the program. Think of them as symbolic placeholders for the values themselves. By contrast, the 2 is just a value itself, called a **literal value**, because it stands alone without being stored in a variable. The = and * characters are operators, they perform actions with the values and variables such as assignment and mathematic multiplication. Most statements in JavaScript conclude with a semicolon (;) at the end. The statement a = b * 2; tells the computer, roughly, to get the current value stored in the variable b, multiply that value by 2, then store the result back into another variable we call a.

Expressions

Statements are made up of one or more expressions. An expression is any reference to a variable or value, or a set of variable(s) and value(s) combined with operators.

For example:

```
a = b * 2;
```

This statement has four expressions in it:

- 2 is a literal value expression.
- b is a variable expression, which means to retrieve its current value.
- b * 2 is an arithmetic expression, which means to do the multi-plication.
- a = b * 2 is an assignment expression, which means to assign the result of the b * 2 expression to the variable a (more on assignments later).

A general expression that stands alone is also called an expression statement, such as the following:

```
b * 2;
```

This flavor of expression statement is not very common or useful, as generally it wouldn't have any effect on the running of the program it would retrieve the value of b and multiply it by 2, but then wouldn't do anything with that result.

Keywords

Keywords are reserved and cannot be used as variable or function names. Here is the complete list of JavaScript keywords:

break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instance of, new, return switch, this, throw, try, type of, var, void, while, with

Block Statement in JavaScript

A block statement groups zero or more statements. In languages other than JavaScript, it is known as a compound statement.

Here's the syntax:

```
{
        //List of Statements
}
```

Variables with a block get scoped to the containing function. Block statement never introduce scope and using var to declare variables don't have block scope.

Example:

```
var a= 20; {
        var b= 40;
}
```

Now, when you will print the value of a, it will print 40, not 20. This is because variable declared with a var within the block has the same scope like var before the block.

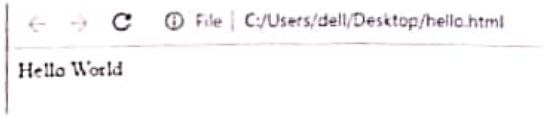
Printing Statements or Variables in JavaScript

To write a strings of text to a document use, **document.write()**. This function can also be used to write the variables directly in the HTML. The **write()** method writes HTML expressions or JavaScript code to a document.

Example: Printing string using document.write()

```
<script type="text/javascript">
    document.write("Hello World");
</script>
```

Output:



Example : Printing variable using document.write()

```
<script type="text/javascript">
    var world = "Hello World"
    document.write(world);
</script>
```

In the above two example the output is same the only difference is inside the document.write() function, in the example 1 the Hello World is a string which is kept inside the double quoted but in example 2 the Hello World is stored in variable named as world and the world variable is printed without using any quoted marks. i.e. double quote or single quote.

Operators

Operators in JavaScript, as in PHP, can involve mathematics, changes to strings, and comparison and logical operations (and, or, etc.). JavaScript mathematical operators look a lot like plain arithmetic for instance, the following statement outputs 7:

Example

```
<script type="text/javascript">
    document.write(2+5);
</script>
```

Arithmetic Operators

Arithmetic operators are used to perform mathematics. It can be used for the main four operations (addition, subtraction, multiplication, and division) as well as to find the modulus (the remainder after a division) and to increment or decrement a value.

The following table shows the arithmetic operators list

Operator	Description	Example
+	Addition	j + 12
-	Subtraction	j - 22
*	Multiplication	j * 7
/	Division	j / 3.13
%	Modulus (division remainder)	j % 6
++	Increment	++j
--	Decrement	

Assignment Operators

The assignment operators are used to assign values to variables. They start with the very simple =, and move on to +=, -=, and so on. The operator += adds the value on the right side to the variable on the left, instead of totally replacing the value on the left. The following table lists the various assignment operator variable

Operator	Example	Equivalent to
=	j = 99	j = 99
+=	j += 2	j = j + 2
+=	j += 'string'	j = j + 'string'
-=	j -= 12	j = j - 12
*=	j *= 2	j = j * 2
/=	j/= 6	j = j / 6
%=	j %= 7	j = j % 7

Comparison Operators

Comparison operators are generally used inside a construct such as an if statement, where you need to compare two items. For example, you may wish to know whether a variable you have

been incrementing has reached a specific value, or whether another variable is less than a set value, and so on. The following table shows the list of comparison operators used in JavaScript.

`	Description	Example
==	Is *equal* to	j == 42
!=	Is *not equal* to	j != 17
>	Is *greater than*	j > 0
<	Is *less than*	j < 100
>=	Is *greater than or equal* to	j >= 23
<=	Is *less than or equal* to	j <= 13
===	Is *equal* to (and of the same type)	j === 56
!==	Is *not equal* to (and of the same type)	j !== '1'

Logical Operators

Unlike PHP, JavaScript's logical operators do not include and and or equivalents to && and ||, and there is no xor operator. The following table shows the list of logical operator used in JavaScript

Operator	Description	Example
&&	And	j == 1 && k == 2
||	Or	j < 100 || j > 0
!	Not	! (j == k)

Incrementing, Decrementing, and Shorthand Assignment

The following forms of post- and pre-incrementing and decrementing you learned to use in PHP are also supported by JavaScript, as are shorthand assignment operators:

++x

--y

x += 22

y -= 3

String Concatenation

JavaScript handles string concatenation slightly differently from PHP. Instead of the . (period) operator, it uses the plus sign (+), like this:

document. write("You have " + messages + " messages.")

Example:

```
<script type="text/javascript">
    var name = "John";
   document.write("Hello " + name + ".");
</script>
```

Output:

| Hello John

Escape Characters

Escape characters, which you've seen used to insert quotation marks in strings, can also be used to insert various special characters such as tabs, newlines, and carriage returns. Here is an example using tabs to lay out a heading—it is included here merely to illustrate escapes, because in web pages, there are better ways to do layout.

Example

```
<script type="text/javascript">
    var title = "Name\tAge\tLocation"
   document.write(title);
</script>
```

Output

Name Age Location

Here the backslash(\) is escaped. The following table shows the JavaScript's escape characters

Character	Meaning
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\'	Single quote (or apostrophe)
\"	Double quote
\\	Backslash
\XXX	An octal number between 000 and 377 that represents the Latin-1 character equivalent (such as \251 for the © symbol)
\xXX	A hexadecimal number between 00 and FF that represents the Latin-1 character equivalent (such as \xA9 for the © symbol)
\uXXXX	A hexadecimal number between 0000 and FFFF that represents the Unicode character equivalent (such as \u00A9 for the © symbol)

Flow Controls

Conditionals alter program flow. They enable you to ask questions about certain things and respond to the answers you get in different ways. There are three types of non-looping conditionals: The if statement, the switch statement, and the? operator.

The if Statement

Several examples in this chapter have already made use of if statements. The code within such a statement is executed only if the given expression evaluates to true.

Syntax:

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

The else Statement

When a condition has not been met, you can execute an alternative by using an **else** statement

Syntax:

```
if(condition){
//statement
}
else{
//statement
}
```

JavaScript has no **elseif** statement, but that's not a problem because you can use an **else** followed by another **if** to form the equivalent of an **elseif** statement

Syntax:

```
if(condition){
//statement
}
else{
//statement
}
```

Example:

```
<script type = "text/javaScript">
if (a > 100)
{
  document.write("a is greater than 100")
}
else if(a < 100)
{
  document.write("a is less than 100")
}
else
{
  document.write("a is equal to 100")
}
</script>
```

The switch Statement

The **switch** statement is useful when one variable or the result of an expression can have multiple values, and you want to perform a different function for each value.

It works by passing a single string to the main menu code according to what the user requests. Let's say the options are Home, About, News, Login, and Links, and we set the variable page to one of these according to the user's input.

Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Example:

```
<script type = "text/javaScript">
switch (page)
  {
    case "Home":
      document.write("You selected Home")
      break
    case "About":
      document.write("You selected About")
      break
    case "News":
      document.write("You selected News")
      break
    case "Login":
      document.write("You selected Login")
      break
    case "Links":
      document.write("You selected Links")
      break
  }
</script>
```

The variable page is mentioned only once at the start of the switch statement. Thereafter, the case command checks for matches. When one occurs, the matching conditional statement is executed. Of course, a real program would have code here to display or jump to a page, rather than simply telling the user what was selected.

Breaking out

The break command allows your code to break out of the switch statement once a condition has been satisfied. Remember to include the break unless you want to continue executing the statements under the next case.

Default Action

When no condition is satisfied, you can specify a default action for a switch statement by using the default keyword.

The ? Operator

The ternary operator (?), combined with the : character, provides a quick way of doing if...else tests. With it you can write an expression to evaluate, and then follow it with a ? symbol and the code to execute if the expression is true. After that, place a : and the code to execute if the expression evaluates to false.

Example:

```
<script type = "text/javaScript">
<script>
  document.write(
    a <= 5 ?
    "a is less than or equal to 5" :
    "a is greater than 5"
  )
</script>
</script>
```

Looping in Javascript

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

There are mainly two types of loops:

1. **Entry Controlled loops:** In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops:** In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.
3. JavaScript mainly provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time

while loop

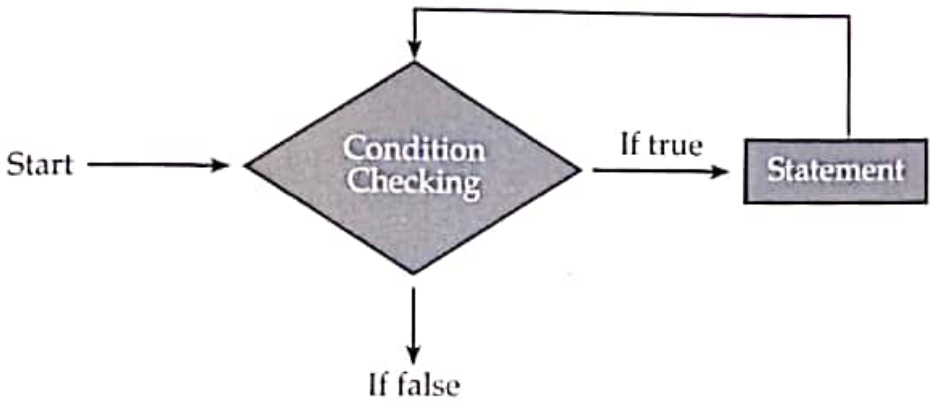
A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax:

while(boolean condition)

```
{
      Loop satements…
}
```

Flow Chart



Explanation:

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
<script type = "text/javaScript">
    // JavaScript program to illustrate while loop
    var x = 1;
    // Exit when x becomes greater than 4
    while (x <= 4)
    {
        document.write("Value of x:" + x + "<br />");
        // increment the value of x for
        // next iteration
        x++;
    }
</script>
```

Output:

Value of x: 1

Value of x: 2

Value of x : 3

Value of x : 4

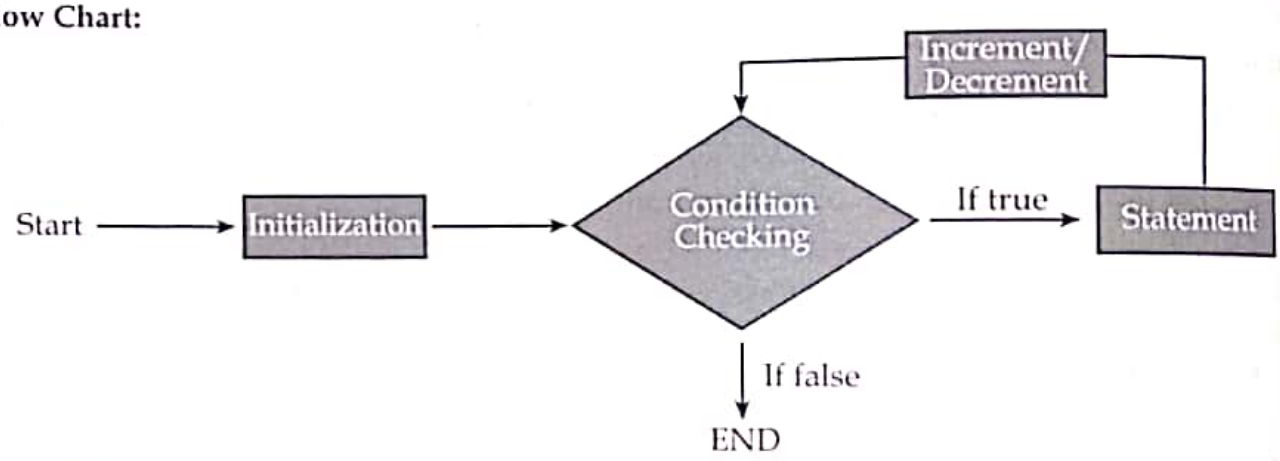
for loop

For loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition; increment/decrement)
{
  statement(s)
}
```

Flow Chart:



Explanation:

- Initialization condition: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- Testing Condition: It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- Statement execution: Once the condition is evaluated to true, the statements in the loop body are executed.
- Increment/ Decrement: It is used for updating the variable for next iteration.
- Loop termination: When the condition becomes false, the loop terminates marking the end of its life cycle.

Example

```
<script type = "text/javaScript">
// JavaScript program to illustrate for loop
    var x;
    // for loop begins when x=2
    // and runs till x <=4
    for (x = 2; x <= 4; x++)
    {
        document.write("Value of x:" + x + "<br />");
    }
</script>
```

Output:

Value of x:2

Value of x:3

Value of x:4

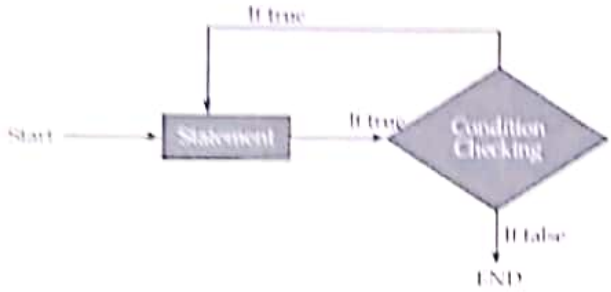
do while

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax:

```
do
{
statements..
}
while (condition);
```

Flow Chart



Explanation

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

Example:

```
<script type = "text/javaScript">
// JavaScript program to illustrate do-while loop
    var x = 21;
    do
    {
        // The line while be printer even
        // if the condition is false
        document.write("Value of x:" + x + "<br />");
        x++;
    } while (x < 20);
</script>
```

Output:

Value of x:21

Functions in JavaScript

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

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax:

```
<script type = "text/javascript">
      function functionname(parameterlist) {
         statements
      }
</script>
```

To create a function in JavaScript, we have to first use the keyword *function*, separated by name of function and parameters within parenthesis. The part of function inside the curly braces {} is the body of the function. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword *function* followed by,
- A user defined function name which should be unique,
- A list of parameters enclosed within parenthesis and separated by commas,
- A list of statement composing the body of the function enclosed within curly braces {}.

Example

```
<script type = "text/javaScript">
function printHello()
{
   document.write("Hello");
}
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

Example:

```
<script type = "text/javaScript">
function printHello()
{
   document.write("Hello");
}
printHello();
</script>
```

Output:

Hello

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example:

```
<script type = "text/javascript">
   function printParameters(name, age) {
      document.write (name + " is " + age + " years old.");
   }
   printParameters('John', 37);
</script>
```

Output:

John is 37 years old

The return Statement

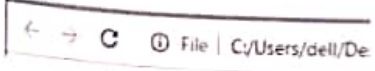
A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example:

```
<script type = "text/javascript">
   function concatenateName(first, last) {
      var full;
      full = first + last;
      return full;
   }
   function printsName() {
      var result;
      result = concatenateName('Zara', ' Ali');
      document.write (result );
   }
   printsName();
</script>
```

Output:



Zara Ali

The following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

Events in Javascript

Popup Boxes

In JavaScript, popup boxes are used to display the message or notification to the user. There are three types of pop up boxes in JavaScript namely **Alert Box, Confirm Box** and **Prompt Box.**

Alert Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed

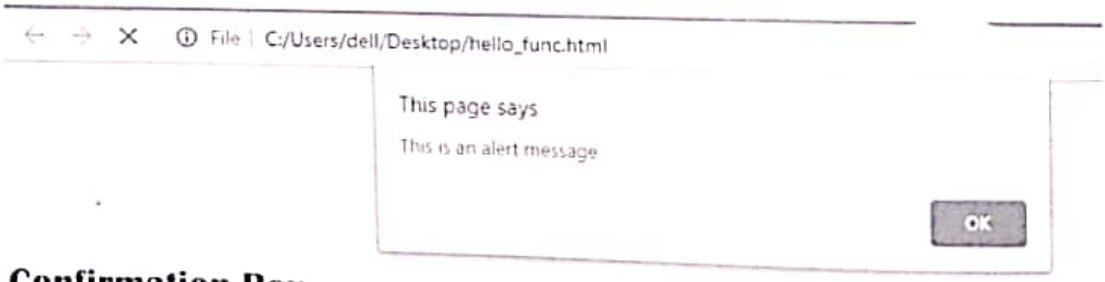
Syntax:

alert("Alert messsage)

Example:

```
<script type = "text/javascript">
    alert("This is an alert message");
</script>
```

Output:



Confirmation Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

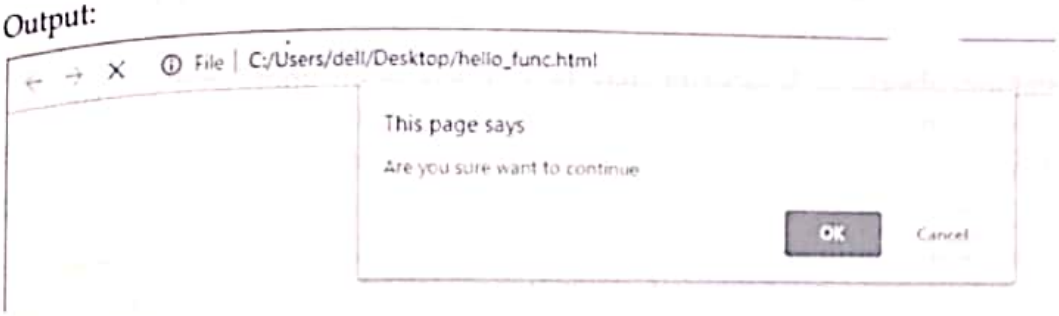
Syntax:

confirm("Message")

Example:

```
<script type = "text/javascript">
    confirm("Are you sure want to continue");
</script>
```

Output:



Prompt Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

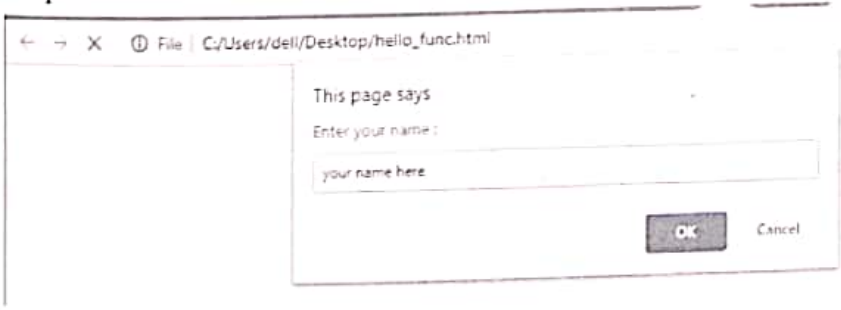
This dialog box is displayed using a method called **prompt()** which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: **OK** and **Cancel**. If the user clicks the OK button, the window method **prompt()** will return the entered value from the text box. If the user clicks the Cancel button, the window method **prompt()** returns null.

Example

```
<script type = "text/javascript">
   var name = prompt("Enter your name : ", "your name here");
       document.write("You have entered : " + name);
</script>
```

Output:



Object and Properties

Objects, in JavaScript, is its most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types (Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

- Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
- An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the

location in memory where the object is stored. The variables don't actually store the value.

- Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

An object can be created with figure brackets {…} with an optional list of properties. A property is a "key: value" pair, where a key is a string (also called a "property name"), and value can be anything.

Example:

```
<script type = "text/javascript">
    // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
 </script>
```

In the above example the college is an object and the faculty:"Bsc.Csit", year:"2017", subject:"Web" is called property and property value of the object.

ccessing Object Properties

The object property can be accessed by following ways:

objectName.propertyName

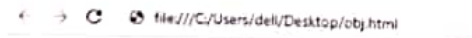
or

objectName["propertyName"]

Example: Using objectName.propertyName

```
<script type = "text/javascript">
    // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
// Display some data from the object:
document.write("The college has " + college.faculty);
 </script>
```

Output:



The college has Bsc.Csit

Example: Using objectName["propertyName"]

```
<script type = "text/javascript">
    // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
// Display some data from the object:
document.write("The college has " + college['faculty']);
 </script>
```

Output:

The college has Bsc. Csit

Adding New Properties

New properties to an existing object can be added by simply giving a value

Example:

```
<script type = "text/javascript">
   // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
college.location ="Lalitpur";
// Display some data from the object:
document.write("The college is located at " + college['location']);
</script>
```

Output:

The college is located at Lalitpur

Deleting Properties

The delete keyword deletes a property from an object.

Example

```
<script type = "text/javascript">
   // Create an object:
var college = {faculty:"Bsc.Csit", year:"2017", subject:"Web"};
college.location ="Lalitpur";
delete college.faculty;
// Display some data from the object:
document.write("The college is faculty is deleted" + college.faculty);
</script>
```

Output

The college is faculty is deleted undefined

In this example we delete the faculty properties so when we tried to get the property value we get undefined message

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called Object.

The new Operator

The new operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example

```
<script type = "text/javascript">
        var book = new Object();    // Create the object
        book.subject = "WEB";      // Assign properties to the object
        book.author  = "Mohtashim";
        document.write("Book name is : " + book.subject + "<br>");
        document.write("Book author is : " + book.author + "<br>");
</script>
```

Output

Book name is : WEB

Book author is: Mohtashim

JavaScript Arrays

The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Array Declaration

An array can be declared using any one of the following techniques

Syntax:

var arrayname = new Array();

Or

var arrayname = [];

Array in JavaScript are indexed from 0.

Example: Creating an array and accessing array elements

```
<script type = "text/javascript">
    // Create an array:
var college = ["NCIT", "NCC", "PMC"];
document.write(college[0] + "<br>");//accessing the array elements
document.write(college[1] + "<br>");
document.write(college[2] + "<br>");
</script>
```

Output

NCIT
NCC
PMC

Array Methods

- **concat():** Returns a new array comprised of this array joined with other arrays or values
  Syntax:
  array1.concat(array2)

Example:

```
<script type = "text/javascript">
    var alpha = ["a", "b", "c"];
    var numeric = [1, 2, 3];
    var alphaNumeric = alpha.concat(numeric);
    document.write("alphaNumeric : " + alphaNumeric );
</script>
```

Output:

alphaNumeric : a,b,c,1,2,3

- **length:** Returns the length of the array
  Syntax:
  arrayname.length

Example

```
<script type = "text/javascript">
    var subject = new Array("Web", "DBMS", "C Programming");
    document.write("The length of the array is : " + subject.length );
</script>
```

Output:
The length of the array is : 3

- **join():** Joins all elements of an array into a string.
  Syntax:
  arrayname.join(separator)
  Seperator: Specifies a string to separate each element of the arrayname. If omitted, the array elements are separated with a comma.

Example:

```
<script type = "text/javascript">
    var arr = new Array("First", "Second", "Third");
    var str = arr.join();
    document.write("str : " + str );
    var str = arr.join(", ");
    document.write("<br />str : " + str );
    var str = arr.join(" + ");
    document.write("<br />str : " + str );
</script>
```

Output:

str: First, Second, Third

str: First, Second, Third

str: First + Second + Third

- **indexOf():**Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

  Syntax:

  arrayname.indexOf(item, start)

  The first parameter is required and the second parameter is optional, it is used for start point of the search.Negative values will start at the given position counting from the end, and search to the end.

Example:

```
<script type = "text/javascript">
    var subject = new Array("Web", "DBMS", "Web", "MicroProcessor", 'C++');
    var  a = subject.indexOf("Web"); //starts the searching for begining
    var  b = subject.indexOf("Web", 2);
    document.write("The first search is in " +a);
    document.write("The second search is in " +b);
</script>
```

Output:

The first search is in 0The second search is in 2

- **lastIndexOf():** The lastIndexOf() method searches the array for the specified item, and returns its position.The search will start at the specified position, or at the end if no start position is specified, and end the search at the beginning of the arrayname.

  Syntax:

  arrayname.lastIndexOf(item,start)

Example:

```
<script type = "text/javascript">
    var str = "This is string one and again string.";
    var index = str.lastIndexOf("string");
document.write("lastIndexOf found String" +index);
document.write("<br/>");
    var index = str.lastIndexOf("one", 5);
    document.write("The lastIndexOf found string" +index);
 </script>
```

Output:

last Index of found string: 29

last Index of found String: - 1

The first parameter is required and the second parameter is optional, it is used for start point of the search. Negative values will start at the given position counting from the end, and search to the end.

- **pop():** Removes the last element from an array and returns that element

Syntax:

arrayname.pop()

Example:

```
<script type = "text/javascript">
    var num = new array(6, 2, 9, 10);
    var element = num.pop();
document.write("Element is" +num);
document.write("<br/>");
    var element = num.pop();
    document.write("Element is " +num);
 </script>
```

Output:

element is : 10

element is : 89

- **push():** Adds one or more elements to the end of an array and returns the new length of the array.

Syntax:

arrayname.push(element1,.....,elementN)

Example:

```
<script type = "text/javascript">
    var numbers = new array(6, 2, 9, 10);
    var element = numbers.push(15);
```

```
document.write("new array is" +numbers);
document.write("<br/>");
    var element = numbers.push(20);
    document.write("new array is " +numbers);
</script>
```

Output:
new array is: 6, 2, 9, 10, 15
new array is : 6, 2, 9, 10, 15, 20

- **reverse() :** Reverses the order of the elements of an array. The first becomes the last, and the last becomes the first.

  Syntax:
  arrayname.reverse()

Example:

```
<script type = "text/javascript">
    var numbers = new array(6, 2, 9, 10);
    var arr = numbers.reverse();
document.write("Reverse array is: " +numbers);
</script>
```

Output:
Reserve array is: 10, 9, 2, 6

- **shift():** Removes the first element from an array and returns that element.

  Syntax:
  arrayname.shift()

Example:

```
<script type = "text/javascript">
    var numbers = new array(6, 2, 9, 10);
    var arr = numbers.shift();
document.write("Removed element is: " +numbers);
</script>
```

Output:
Removed element is: 2, 9, 10

- **unshift() :** Adds one or more elements to the front of an array and returns the new length of the array.

  Syntax:
  arrayname.unshift(element1,.....,elementN)

Example:

```
<script type = "text/javascript">
    var numbers = new array(6, 2, 9, 10);
```

```
    var arr = numbers.unshift(15);
document.write("Added element is: " +numbers);
</script>
```

Output:

Added element is : 15, 6, 2, 9, 10

- **sort():** Sorts the elements of an array

Syntax:

arrayname.sort()

Example:

```
<script type = "text/javascript">
    var arr = new array(6, 2, 9, 10);
    var sorted = arr.sort();
document.write("Sorted array is: " +numbers);
</script>
```

Output:

Sorted array is: banana, mango, orange, sugar

- **slice():** Extracts a section of an array and returns a new array.

Syntax:

arrayname.slice(begin,end)

Example:

```
<script type = "text/javascript">
    var arr = new array("orange", "mango", "banana", "sugar", "tea");
    document.write("arr.slice(1, 2): " + arr.slice( 1, 2));
document.write("<br/>arr.slice(1, 2): " + arr.slice( 1, 3));
</script>
```

Output:

arr.slice(1,2): mango

arr.slice(1,3: mango, banana

- **toString():** Returns a string representing the array and its elements.

Syntax

arrayname.toString()

Example:

```
<script type = "text/javascript">
    var arr = new array("orange", "mango", "banana", "sugar");
    var str = arr.toString();
document.write("Returned string is: " + str);
</script>
```

Output:

Returned string is: orange, mango, banana, sugar

String In JavaScript:

The String object lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods.JavaScript automatically converts between string primitives and String objects

Syntax:

var name = new String(string_name)

String Methods and Properties

- **length :**The length property returns the length of a string

  Syntax:
  string.length

Example

```
<script type = "text/javascript">
    var text = "HTML is not a programming language";
    var length = text.length;
    document.write("The length of the string is " + length);
</script>
```

Output:
The length of the string is 34

- **charAt():**Returns the character at the specified index

  Syntax:
  string.charAt(index)
  The index of the first character is 0, the second character is 1, and so on.

Example:

```
<script type = "text/javascript">
    var text = "This is a string.";
    document.write("text.charAt(5) is: " + text.charAt(5));
</script>
```

Output:
str.charAt(5) is: i

- **concat():** Combines the text of two strings and returns a new string. This method does not change the existing strings, but returns a new string containing the text of the joined strings.

  Syntax:
  string.concat(string2,.....string3,.....,stringN)

Example:

```
<script type = "text/javascript">
    var str1 = "Hello";
var str2 = "World";
    document.write("Combined string is: " + str1.concat(str2));
</script>
```

Output:
Combined string is Hello World

- **indexOf():** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.

  Syntax:

  string.indexOf(*searchvalue, start*)

Example:

```
<script type = "text/javascript">
    var str = ("This is a String.");
    var index = str.indexOf("a");
    document.write("indexOf(a) is: " +index);
 </script>
```

Output:

indexOf(a) is: 8

- **lastIndexOf():** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.

  Syntax:

  string.lastIndexOf(*searchvalue, start*)

Example:

```
<script type = "text/javascript">
    var str = ("This is a String and also a text.");
    var index = str.lastIndexOf("a");
    document.write("lastIndexOf(a) is: " +index);
 </script>
```

Output:

lastIndexOf(a) is : 26

- **localeCompare():** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

  Syntax:

  string.localeCompare(*compareString*)

Example:

```
<script type = "text/javascript">
          var str1 = new String( "This is a string" );
          var index1 = str1.localeCompare( "is a" );
document.write("localeCompare first :" + index1 );
          var index = str1.localeCompare( "What" );
document.write("<br/>localeCompare second :" + index );
</script>
```

Output:

locale Compare first: 1

locale Compare second: – 1

- **match():** Used to match a regular expression against a string.

  Syntax:

  string.match(*regexp*)

Example:

```
<script type = "text/javascript">
var string = "This is a string";
var result = string.match(/trin/i);
document.write("Match() : " + result);
</script>
```

Output:
Match():train

- **replace():** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Syntax:
string.replace(searchvalue, newvalue)

Example:

```
<scripttype = "text/javascript">
          var str = 'Hello This is Web Technology';
          var newstring = str.replace('Technology', 'Programmming');
document.write(newstring);
</script>
```

Output:
Hello This is Web Programming

- **search():** Executes the search for a match between a regular expression and a specified string

Syntax:
string.search(searchvalue)

Example:

```
<script type = "text/javascript">
var str = "This is a string.";
var res = str.search("a");
document.write(res);
</script>
```

Output:
8

- **slice():** Extracts a section of a string and returns a new string.

Syntax:

string.slice(start, end)

Example:

```
      <scripttype = "text/javascript">
          var str = "Hello world!";
          var res = str.slice(0, 5);
document.write(res);
</script>
```

Output:
Hello

- **split():** Splits a String object into an array of strings by separating the string into substrings.

  Syntax:
  string.split(separator, limit)

Example:

```
<scripttype = "text/javascript">
          var str = "How are you doing today?";
          var res = str.split(" ");
document.write(res);
</script>
```

Output:
How, are, you, doing, today?

- **substr():** Returns the characters in a string beginning at the specified location through the specified number of characters.

  Syntax:
  string.substr(start, length)

Example:

```
<scripttype = "text/javascript">
          var str = "Hello world!";
          var res = str.substr(0, 5);
document.write(res);
</script>
```

Output:
Hello

- **substring():** Returns the characters in a string between two indexes into the string.

  Syntax:
  string.substring(start, end)

Example:

```
<scripttype = "text/javascript">
          var str = "Hello world!";
          var res = str.substring(1, 4);
document.write(res);
</script>
```

Output:
ell

- **to Locale Lower Case():** The characters within a string are converted to lower case while respecting the current locale.

  Syntax:
  string.toLocaleLowerCase()

Example:

```
<scripttype = "text/javascript">
          var str = "Hello World!";
          var res = str.toLocaleLowerCase();
document.write(res);
</script>
```

Output:
hello world!

- **to Locale Upper Case():** The characters within a string are converted to upper case while respecting the current locale.

  Syntax:
  string.toLocaleUpperCase ()

Example:

```
<scripttype = "text/javascript">
          var str = "Hello World!";
          var res = str.toLocaleUpperCase();
document.write(res);
</script>
Output:
```

HELLO WORLD!

- **toLowerCase():** Returns the calling string value converted to lower case.

  Syntax:
  string.toLowerCase()

Example:

```
<scripttype = "text/javascript">
          var str = "Hello World!";
          var res = str.toLowerCase();
document.write(res);
</script>
```

Output:
hello world!

- **toUpperCase():**Returns the calling string value converted to uppercase.

  Syntax:
  string.toUpperCase()

Example:

```
<scripttype = "text/javascript">
          var str = "Hello World!";
          var res = str.toUpperCase();
document.write(res);
</script>
```

Output:
Hello World!

- **toString():** converts a number to a string.

  Syntax:
  number.toString()

Example:

```
<scripttype = "text/javascript">
          var num = 321;
          var res = num.toString(8);
document.write(res);
</script>
```

Output:
501

- **valueOf()**: Returns the primitive value of the specified object.

  Syntax:
  string.valueOf()

Example:

```
<script type="text/javascript">
var str = "Hello World!";
var res = str.valueOf();
document.write(res);
</script>
```

Output:

Hello World!

Number in JavaScript

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

Syntax:

var n = new Number(number)

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Number Methods in JavaScript

- **toExponential()**:Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.

  Syntax:
  number.toExponential(*x*)
  where x is optional and an integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value

Example:

```
<script type="text/javascript">
    var num = 5.56789;
      var res = num.toExponential();
document.write(res);
</script>
```

Output:
5.56789e+0

- **toFixed()** : Formats a number with a specific number of digits to the right of the decimal.

  Syntax:
  number.toFixed(*x*)
  where x is optional and the number of digits after the decimal point. Default is 0 (no digits after the decimal point)

Example:

```
<script type="text/javascript">
var num = 5.56789;
var res = num.toFixed(2);
document.write(res);
</script>
```

Output:

5.57

- **toLocaleString()** : Returns a string value version of the current number in a format that may vary according to a browser's local settings.

Syntax

number.toLocaleString(*locales, options*)

Example:

```
<script type="text/javascript">
        var num = 3325;
        var result = num.toLocaleString('en-US');
document.write(result);
</script>
```

Output:

3,325

- **toPrecision()** : Defines how many total digits (including digits to the left and right of the decimal) to display of a number.

Syntax:

number.toPrecision(x)

where x isOptional. The number of digits. If omitted, it returns the entire number (without any formatting)

Example:

```
<script type="text/javascript">
    var num=213.45689;
document.write(num.toPrecision(4));
</script>
```

Output:

213.5

- **toString()**:Returns the string representation of the number's value.

Example:

```
<script>
        var num = 321;
        var res = num.toString(8);
document.write(res);
</script>
```

Output:

501

- **valueOf():** Returns the number's value.

  Syntax:
  number.valueOf()

Example:

```
<script type="text/javascript">
        var num=0/0;
document.write("Output : " + num.valueOf());
</script>
```

Output:
Output: NaN

Boolean in JavaScript

The Boolean object represents two values, either "true" or "false". If value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Syntax:

var name= new Boolean(value)

Boolean Methods

- **toSource():** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.

  Syntax:
  boolean.toSource()

- **toString():**Returns a string of either "true" or "false" depending upon the value of the object.

  Syntax:

  boolean.toString()

- valueOf():Returns the primitive value of the Boolean object.

Date In JavaScript

- **getyear():** returns the year in the specified date according to local time. Use getFullYear instead.

  Syntax:

  date.getFullYear();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 01:15:00");
  var n = d.getFullYear();
document.write("Full Year: "+n);
}
```

Output:
Full Year: 1999

- **setDate() :** Sets the day of the month for a specified date according to local time.

  Syntax:
  date.setDate(day_number);

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 03, 1999 01:15:00");
  d.setDate(29);
document.write("demo").innerHTML = d;
}
```

Output:
New Date: Mon Mar 29 1999 01:15:00 GMT + 0545 (Nepal Time)

- **setFullYear() :** Sets the full year for a specified date according to local time.

  Syntax:
  date.setFullYear(year_number);

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 03, 1999");
  d.setFullYear(2019);
document.write("demo").innerHTML = d;
}
</script>
```

Output:
New Full Year: Sun Mar 03 2019 00:00:00 GMT + 0545 (Nepal Time)

- **setHours():** Sets the hours for a specified date according to local time.

  Syntax:

  date.setHours(Hrs, minutes,seconds);

Example:

```
<script type="text/javascript">
function myFunction() {
 var d = new Date();
d.setHours(15, 35, 1);
document.write("demo").innerHTML = d;
}
</script>
```

Output:

Sun Oct 20 2019 15:35:01: GMT+0545(Nepal Time)

- **setMilliseconds():** Sets the milliseconds for a specified date according to local time.

  Syntax:
  date.setMilliseconds

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 03, 1999");
  d.setFullYear(2019);
document.write("demo").innerHTML = d;
}
</script>
```

Output:
New Set Milliseconds: 192

- **setMinutes() :** Sets the minutes for a specified date according to local time.

  Syntax:
  date.setMinutes();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  d.setMinutes(17);
document.write("demo").innerHTML = d;
}
</script>
```

Output:
New Set Minutes: Mon Mar 29 1999 08:17:00 GMT + 0545 (Nepal Time)

- setMonth() : Sets the month for a specified date according to local time.

  Syntax:
  date.setMonth();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  d.setMonth(01);
document.write("demo").innerHTML = d;
}
</script>
```

Output:
New Set Months: Mon Mar 01 1999 08:44:00 GMT + 0545 (Nepal Time)

- **setSeconds() :** Sets the seconds for a specified date according to local time.

  Syntax:
  date.setSeconds();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  d.setSeconds(44);
document.write("demo").innerHTML = d;
}
</script>
```

Output:
New Set Second: Mon Mar 29 1999 08:44:44 GMT + 0545 (Nepal Time)

- **setTime() :** Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.

  Syntax:
  date.setTime();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  d.setTime(151214171819);
document.write= d;
}
</script>
```

Output:
New Set Time : Thu Oct 17 1974 09: 26:11 GMT+ 0530 (Nepal Time)

- **toDateString:** Returns the "date" portion of the Date as a human-readable string.

  Syntax:

  Date.toDateString();

Example:

```
<script>
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  var n = d.toDateString();
document.write("Date String: "+n);
}
</script>
```

Output:

Date String: Mon Mar 29 1999 to Locale Date String():Returns the "date" portion of the Date as a string, using the current locale's conventions.

Syntax:

date.toLocaleDateString();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date();
  var n = d.toLocaleDateString();
document.write("Locale Date String: "+n);
}
</script>
```

Output:
Locale Date String: 10/20/2019

- **toLocaleFormat():** Converts a date to a string, using a format string.

  Syntax:

  date.toLocaleFormat();

Example:

```
<script type = "text/javascript">
       function format(){
          var dt = new Date(1993, 6, 28, 14, 39, 7);
document.write( "Formated Date : " + dt.toLocaleFormat( "%A, %B %e, %Y"
) );
     }
</script>
```

Output:

Formated Date: Web Jul 28 1993 14:39:07

- **toLocaleString():** Converts a date to a string, using the current locale's conventions.

  Syntax:

  date.toLocaleString();

Example:

```
<script>
function myFunction() {
  var d = new Date("March 29, 1999 8:44:00");
  var n = d.toLocaleString();
document.write("Local String: "+n);
}
</script>
Output:
```

Local String: 3/29/1999, 8:44:00 AM

- **to Locale Time String():**Returns the "time" portion of the Date as a string, using the current locale's conventions.

  Syntax:

  date. to Locale Time String();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date();
  var n = d.toLocaleTimeString();
document.write("Local Time String: "+n);
}
</script>
```

Output:

Local Time String: 4:06:05 PM

- **toSource():** Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.

  Syntax:

  date.toSource

- **toString():** Returns a string representing the specified Date object.

  Syntax:

  date.toString();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date();
   var n = d.toString();
document.write("To String: "+n);
}
</script>
```

Output:

To String: Sun Oct 20 32019 16:10:23 GMT + 0545 (Nepal Time)

- **TimeString():** Returns the "time" portion of the Date as a human-readable string.

  Syntax:

  date.toTimeString();

Example:

```
<script type="text/javascript">
function myFunction() {
  var d = new Date();
   var n = d.toTimeString();
document.write("To Time String: "+n);
}
</script>
```

Output:

To Time String: 16: 12: 35 GMT +0545(Nepal Time)

- **valueOf():**Returns the primitive value of a Date object.

  Syntax:

  date.valueOf();

Example:

```
<script type="text/javascript">
      function myFunction() {
        var d = new Date();
         var n = d.valueOf();
      document.write("Value Of: "+n);
      }
      </script>
```

Output:

Value of 1571567375060

Math in JavaScript

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of Math are static and can be called by using Math as an object without creating it.

Math Methods in JavaScript

1. abs(): Returns the absolute value of a number
2. ceil(): Returns the smallest integer greater than or equal to a number
3. exp():Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
4. floor(): Returns the largest integer less than or equal to a number
5. log(): Returns the natural logarithm of a number
6. max(): Returns the largest of zero or more numbers
7. min(): Returns the smallest of zero or more numbers
8. pow(): Returns base to the exponent power, that is base exponetn
9. random(): Returns a pseudo-random number between 0 and 1
10. round():Returns the value of a number rounded to the nearest integer
11. sqrt(): Returns the square root of a number

Regular Expressions and RegExp Object

A regular expression is an object that describes a pattern of characters.

The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax

A regular expression could be defined with the RegExp () constructor, as follows

```
var pattern = new RegExp(pattern, attributes);
```

or simply

```
var pattern = /pattern/attributes;
```

JavaScript - Document Object Model or DOM

Every web page resides inside a browser window which can be considered as an object.

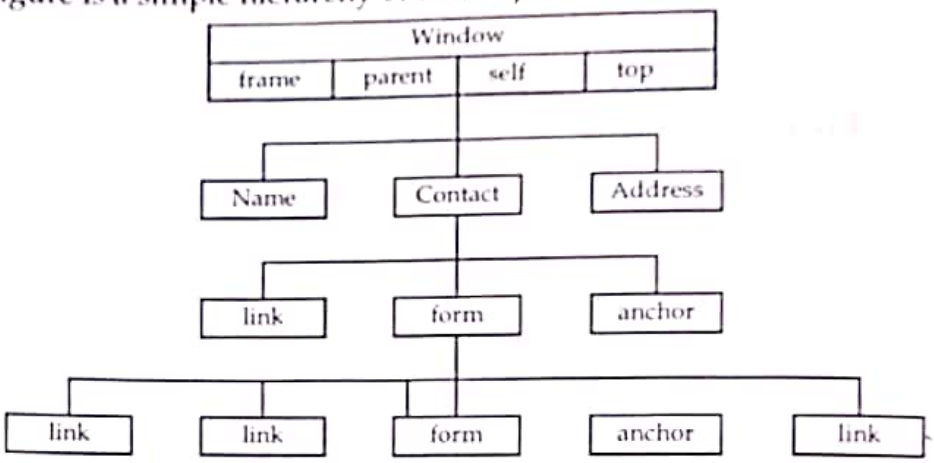
A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** – Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** – Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** – Everything enclosed in the <form>...</form> tags sets the form object.

- **Form control elements** – The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

The following figure is a simple hierarchy of few objects:



HTML DOM in Java Script

document.get ElementById()

The getElementById() method returns the element that has the ID attribute with the specified value.This method is one of the most common methods in the HTML DOM, and is used almost every time you want to manipulate, or get info from, an element on your document.Returns *null* if no elements with the specified ID exists.An ID should be unique within a page. However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code.

Syntax:

document.getElementById(elementID)

document.get Element Byld ("Elementld").value

The value property sets or returns the value of the value attribute of a text field.The value property contains the default value OR the value a user types in (or a value set by a script).

Example:

```
<html>
<head>
<body>
<form>
    Enter No:<input type="text" id="number" name="number"/><br/>
    <input type="button" value="Submit" onclick="getcube()"/>
</form>
</body>
<script type="text/javascript">
    function getcube(){
    var number=document.getElementById("number").value;
    alert(number*number*number);
    }
    </script>
</html>
```

Output:

document.get Element By Name()

The document.get Elements By Name () method returns all the element of specified name.

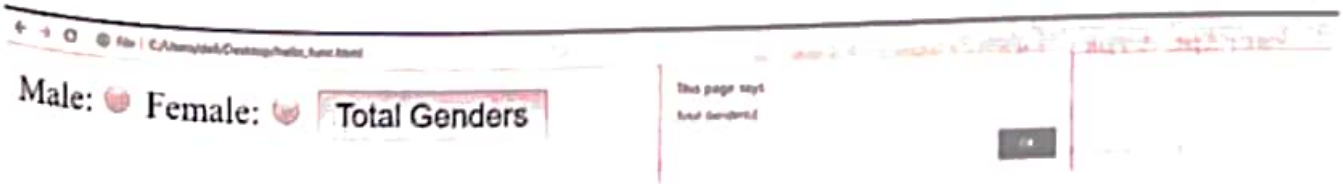
Syntax:

document.get Elements By Name("name")

Example:

```
<html>
<head>
<script type="text/javascript">
    function totalelements()
    {
    var allgenders=document.getElementsByName("gender");
    alert("Total Genders:"+allgenders.length);
    }
    </script>
    <form>
    Male:<input type="radio" name="gender" value="male">
    Female:<input type="radio" name="gender" value="female">
    <input type="button" onclick="totalelements()" value="Total Genders">
    </form>
</body>
</html>
```

Output:



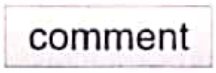
Java script - innerHTML

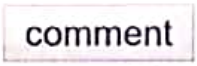
The innerHTML property can be used to write the dynamic html on the html document. It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example

```
<html>
    <script type="text/javascript" >
        function showcommentform() {
        var data="Javascript is fun";
        document.getElementById('mylocation').innerHTML=data;
        }
        </script>
        <form name="myForm">
        <input type="button" value="Show Magic" onclick="showcommentform()">
        <div id="mylocation"></div>
        </form>
</html>
```

Output:





Javascript is fun

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field. Here, **document** is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

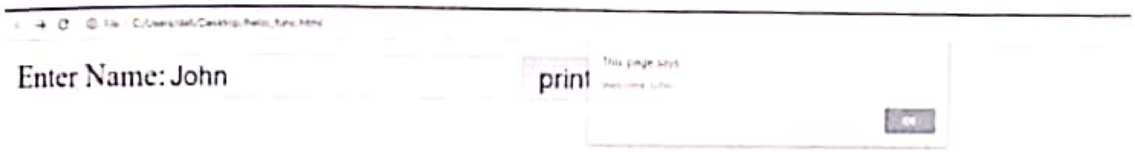
Example:

```
<html>
    <script type="text/javascript">
        function printvalue(){
        var name=document.form1.name.value;
        alert("Welcome: "+name);
```

```
    }
    </script>

    <form name="form1">
    Enter Name:<input type="text" name="name"/>
    <input type="button" onclick="printvalue()" value="print name"/>
    </form>
</html>
```

Output:



Form Validation

Forms are used in web pages for the user to enter their required details that are further send it to the server for processing. A form is also known as web form or HTML form.

If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Some terms used in Form validation

Check for empty: To check if all the fields are filled or not, check for empty which can be done by checking with empty string

- **Number validation:** To check for the numbers , use regular expression with **match** properties

Example:

```
<script type="text/javascript">
function printvalue(){
var pattern = /^[0-9]+$/;
var number = document.getElementById("num").value;
if(number.match(pattern))
{
alert('Your Registration number has accepted....');
document.form1.text1.focus();
return true;
}
}
</script>
```

- **Email validation:** Validating email is a very important point while validating an HTML form. In this page we have discussed how to validate an email using JavaScript. An email is a string (a subset of ASCII characters) separated into two parts by @ symbol, a "personal_info" and a domain, that is personal_info@domain. The length of the personal_info part may be up to 64 characters long and domain name may be up to 255 characters

Example:

```
<script type = "text/javascript">
      function validateEmail() {
         var emailID = document.getElementById("email").value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");
         if (atpos < 1 || ( dotpos - atpos < 2 )) {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
         }
         return( true );
      }
 </script>
```

- Letters and Number validation:

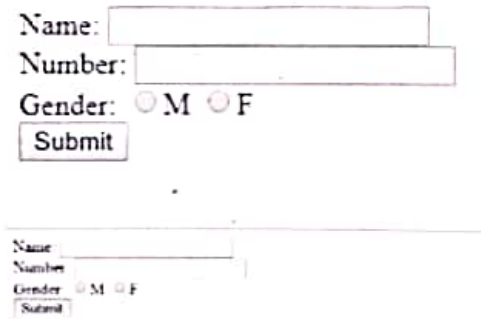
```
<script type = "text/javascript">
function alphanumeric(inputtxt)
{
var letters = /^[0-9a-zA-Z]+$/;
var string = document.getElementById('name').value;
if(string.match(letters))
{
alert('Your registration number have accepted : you can try another');
document.form1.text1.focus();
return true;
}
else
{
alert('Please input alphanumeric characters only');
return false;
}
}
</script>
```

Example: Combining all for form validation

```
<html>
<head>
<title>Form Validation</title>
</head>
<body>
<label>Name:</label>
<input type="text" id="name"><br>
<label>Number:</label>
<input type="text" id="num"><br>
<label>Gender:</label>
<input type="radio" name="gender" value="Male" id="male">M
<input type="radio" name="gender" value="Female" id="female">F<br>
<input type="submit" onclick="check();">
</body>
<script>
function check(){
var n=document.getElementById("num").value;
if(n == ""){
alert("Enter number");
}
else if(isNaN(n)){
```

```
alert('is not a number');
}
var condition=/^[A-Za-z]+$/;
var name=document.getElementById("name").value;
var result=name.match(condition);
if(name = ""){
alert("Enter name");
}
else if(!result){
alert("Name format not valid");
}
var female=document.getElementById("female").checked;
var male=document.getElementById("male").checked;
if (male||female){
alert("Gender selected");
}
else{
alert("Gender required");
}
}
</script>
</html>
```

Output:



Error Handling

There are three types of errors in programming:

- Syntax Errors,
- Runtime Errors
- Logical Errors.

Syntax Errors

Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript. For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">
      window.print(;
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

Runtime errors, also called exceptions, occur during execution (after compilation/ interpretation). For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
      window.printme();
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax

```
<script type = "text/javascript">
      try {
         // Code to run
         [break;]
      }
      catch ( e ) {
         // Code to run if an exception occurs
         [break;]
      }
      [ finally {
         // Code that is always executed regardless of
         // an exception occurring
      }]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.

JavaScript Cookies

A cookie is variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values

Example of Cookies:

a. Name cookie
b. Date cookie

How It Works?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields :

- Expires – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- Domain – The domain name of your site.
- Path – The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- Secure – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- Name=Value – Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the cookie property of the Document object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

Storing Cookies

The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this.

```
document.cookie = "key1 = value1;key2 = value2;expires = date";
```

Here the expires attribute is optional. If you provide this attribute with a valid date or time then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

Example:

```
<html>
  <head>
    <script type = "text/javascript">
        function WriteCookie() {
           if( document.myform.customer.value == "" ) {
              alert("Enter some value!");
              return;
           }
           cookievalue = escape(document.myform.customer.value) + ";";
           document.cookie = "name=" + cookievalue;
           document.write ("Setting Cookies : " + "name=" + cookievalue );
        }
    </script>
  </head>

  <body>
    <form name = "myform" action = "">
       Enter name: <input type = "text" name = "customer"/>
       <input type = "button" value = "Set Cookie" onclick = "WriteCookie();
"/>
    </form>
  </body>
</html>
```

Output:

Setting Cookies: name= web_Tech;

Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' split() function to break a string into key and values

Example:

```
<html>
   <head>
      <script type = "text/javascript">
            function ReadCookie() {
               var allcookies = document.cookie;
               document.write ("All Cookies : " + allcookies );

               // Get all the cookies pairs in an array
               cookiearray = allcookies.split(';');

               // Now take key value pair out of this array
               for(var i=0; i<cookiearray.length; i++) {
                  name = cookiearray[i].split('=')[0];
                  value = cookiearray[i].split('=')[1];
                  document.write ("Key is : " + name + " and Value is : " + va
lue);
               }
            }
      </script>
   </head>
   <body>
      <form name = "myform" action = "">
         <p> click the following button and see the result:</p>
         <input type = "button" value = "Get Cookie" onclick = "ReadCookie()"/
>
      </form>
   </body>
</html>
```

Output:

click the following button and see the result:

Get Cookie

Setting Cookies Expiry Date

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the 'expires' attribute to a date and time.

Example

```
<html>
   <head>
      <script type = "text/javascript">
            function WriteCookie() {
               var now = new Date();
               now.setMonth( now.getMonth() + 1 );
               cookievalue = escape(document.myform.customer.value) + ";"
               document.cookie = "name=" + cookievalue;
               document.cookie = "expires=" + now.toUTCString() + ";"
               document.write ("Setting Cookies : " + "name=" + cookievalue );
            }
      </script>
   </head>
   <body>
      <form name = "myform" action = "">
         Enter name: <input type = "text" name = "customer"/>
         <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"
/>
      </form>
   </body>
</html>
```

Deleting a Cookie

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

Example:

```
<html>
   <head>
      <script type = "text/javascript">
            function WriteCookie() {
               var now = new Date();
               now.setMonth( now.getMonth() - 1 );
               cookievalue = escape(document.myform.customer.value) + ";"
               document.cookie = "name=" + cookievalue;
               document.cookie = "expires=" + now.toUTCString() + ";"
               document.write("Setting Cookies : " + "name=" + cookievalue );
            }
      </script>
   </head>

   <body>
      <form name = "myform" action = "">
         Enter name: <input type = "text" name = "customer"/>
         <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"
/>
      </form>
   </body>
</html>
```

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

Download the jQuery library from jQuery.com

Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.4.1.min.js"></script>
</head>
```

or just use

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
</head>
```

jQuery Syntax

It is used for selecting elements in HTML and performing the action on those elements.

Syntax:

$(selector).action()

The selector may be id selector or class selector

- **$ sign:** It grants access to jQuery.
- **(selector):** It is used to find HTML elements.
- **jQuery action():** It is used to perform actions on the elements

Examples:

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.
$(".test").hide() - hides all elements with class="test".
$("#test").hide() - hides the element with id="test".

Events and Effects

jQuery Methods are inside a Document ready event for easy reading of code.

Syntax:

```
$(document).ready(function(){
 // jQuery Method
});
```

This is to check and stop the jquery before the document is finished loading. This method also allows you to have JavaScript code before the body of your document, in the head section.

What are Events?

All the different visitors' actions that a web page can respond to are called events.An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".

Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

Effects

jQuery hide() and show()

Example:

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"
></script>
<script>
$(document).ready(function(){
  $("#hide").click(function(){
```

```
    $("p").hide();
  });
  $("#show").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>
<p>If you click on the "Hide" button, I will disappear.</p>
<button id="hide">Hide</button>
<button id="show">Show</button>
</body>
</html>
```

Output:

If you click on the "Hide" button, I will disappear.

Hide Show

The some other jQuery Effects are:

animate()	Runs a custom animation on the selected elements
clearQueue()	Removes all remaining queued functions from the selected elements
delay()	Sets a delay for all queued functions on the selected elements
dequeue()	Removes the next function from the queue, and then executes the function
fadeIn()	Fades in the selected elements
fadeOut()	Fades out the selected elements
fadeTo()	Fades in/out the selected elements to a given opacity
fadeToggle()	Toggles between the fadeIn() and fadeOut() methods
finish()	Stops, removes and completes all queued animations for the selected elements
hide()	Hides the selected elements
queue()	Shows the queued functions on the selected elements
show()	Shows the selected elements
slideDown()	Slides-down (shows) the selected elements
slideToggle()	Toggles between the slideUp() and slideDown() methods
slideUp()	Slides-up (hides) the selected elements
stop()	Stops the currently running animation for the selected elements
toggle()	Toggles between the hide() and show() methods

Introduction to JSON

JSON: short for JavaScript Object Notation, is a format for sharing data. As its name suggests, JSON is derived from the JavaScript programming language, but it's available for use by many languages including Python, Ruby, PHP, and Java. JSON is usually pronounced like the name "Jason."

JSON uses the .json extension when it stands alone. When it's defined in another file format (as in .html), it can appear inside of quotes as a JSON string, or it can be an object assigned to a variable. This format is easy to transmit between web server and client or browser.

Very readable and lightweight, JSON offers a good alternative to XML and requires much less formatting. This informational guide will get you up to speed with the data you can use in JSON files, and the general structure and syntax of this format.

Syntax and Structure

A JSON object is a key-value data format that is typically rendered in curly braces. When you're working with JSON, you'll likely see JSON objects in a .json file, but they can also exist as a JSON object or string within the context of a program.

A JSON object looks something like this:

```
{
"first_name" :"Sammy",
"last_name" :"Shark",
"location" :"Ocean",
"online" :true,
"followers" :987
}
```

Although this is a very short example, and JSON could be many lines long, this shows that the format is generally set up with two curly braces (or curly brackets) that look like this { } on either end of it, and with key-value pairs populating the space between. Most data used in JSON ends up being encapsulated in a JSON object.

Key-value pairs have a colon between them as in "key" : "value". Each key-value pair is separated by a comma, so the middle of a JSON looks like this: "key" : "value", "key" : "value", "key": "value". In our example above, the first key-value pair is "first_name" : "Sammy".

JSON keys are on the left side of the colon. They need to be wrapped in double quotation marks, as in "key", and can be any valid string. Within each object, keys need to be unique. These key strings *can* include whitespaces, as in "first name", but that can make it harder to access when you're programming, so it's best to use underscores, as in "first_name".

JSON values are found to the right of the colon. At the granular level, these need to be one of 6 simple data types:

- strings
- numbers
- objects
- arrays
- Booleans (true or false)
- null

At the broader level, values can also be made up of the complex data types of JSON object or array, which is covered in the next section.

Each of the data types that are passed in as values into JSON will maintain their own syntax, so strings will be in quotes, but numbers will not be.

Though in .json files, we'll typically see the format expanded over multiple lines, JSON can also be written all in one line.

```
{ "first_name" : "Sammy", "last_name": "Shark", "online" : true, }
```

Chapter 5

AJAX AND XML

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- Basics of AJAX
- Introduction to XML and its Application
- Syntax Rules for creating XML document
- XML Elements
- XML Attributes
- XML Tree
- XML Namespace
- XML schema languages: Document Type Definition (DTD), XML Schema Definition (XSD)
- XSD Simple Types, XSD Attributes
- XSD Complex Types
- XML Style Sheets (XSLT), XQuery

BASICS OF AJAX

AJAX stands for Asynchronous JavaScript and XML, and it describes a set of development techniques used for building websites and web applications. AJAX's major function is to update web content asynchronously i.e. a user's web browser doesn't need to reload an entire web page when only a small portion of content on the page needs to change.

One of the most ubiquitous examples of asynchronous updating is Google's "Google Suggest" feature. When you enter a search query into Google's search bar and the Google website automatically begins offering auto-complete options while you type, that's AJAX in action. The content on the page changes (in this case, the auto-complete options in the search bar) without having to manually refresh the page (something that would make Google Suggest impractical to use). Features like Google Suggest are a fundamental part of contemporary web browsing, which points to how essential AJAX is in web development.

WORKING OF AJAX

JavaScript and XML combine to make asynchronous updating happen through the use of something called an XML HttpRequest object. When a user visits a web page designed to make use of AJAX and a prescribed event occurs (the user loads the page, clicks a button, fills out a form, etc.) JavaScript creates an XMLHttpRequest object, which then transfers data in an XML format between a web browser (the program being used to view the website) and a web server (the software or hardware where a website's data is stored). The XMLHttpRequest object sends a request for updated page data to the web server, the server process the request, a response is created server-side and sent back to the browser, which then uses JavaScript to process the response and display it on the screen as updated content.

ADVANTAGE OF AJAX

Speed

It reduces the server traffic in both side request. Also reduce the time consuming on both side response.

Interaction

AJAX is much responsive, whole page (small amount of) data transfer at a time.

XMLHttpRequest

XMLHttpRequest has an important role in the Ajax web development technique. XMLHttpRequest is special JavaScript object that was designed by Microsoft. XMLHttp Request object call as a asynchronous HTTP request to the Server for transferring data both side. It's used for making requests to the non-Ajax pages.

Asynchronous calls

AJAX make asynchronous calls to a web server. This means client browsers are avoid waiting for all data arrive before start the rendering.

Form Validation

This is the biggest advantage. Form are common element in web page. Validation should be instant and properly, AJAX gives you all of that, and more.

Bandwidth Usage

No require to completely reload page again. AJAX is improve the speed and performance. Fetching data from database and storing data into database perform background without reloading page.

DISADVANTAGES OF AJAX

1. AJAX application would be a mistake because search engines would not be able to index an AJAX application.
2. Open Source: View source is allowed and anyone can view the code source written for AJAX.
3. ActiveX requests are enabled only in Internet Explorer and newer latest browser.
4. The last disadvantage, XMLHttpRequest object itself. For a security reason you can only use to access information from the web host that serves initial pages. If you need to fetching information from another server, it's is not possible with in the AJAX.

Sample Ajax Program using PHP & Ajax

Ajax.html

```
<html>
<head>
<title>Programming Language - Search</title>
<script type="text/javascript" src="js/auto_complete.js"></script>
</head>
<body>
<h2>Programming Language - Search</h2>
<p><b>Type the first letter of the Programming Language</b></p>
<form method="POST" action=" ">
<p><input type="text" size="40" id="txtHint"
onkeyup="showName(this.value)"></p>
</form>
<p>Finding: <span id="txtName"></span></p>
</body>
</html>
```

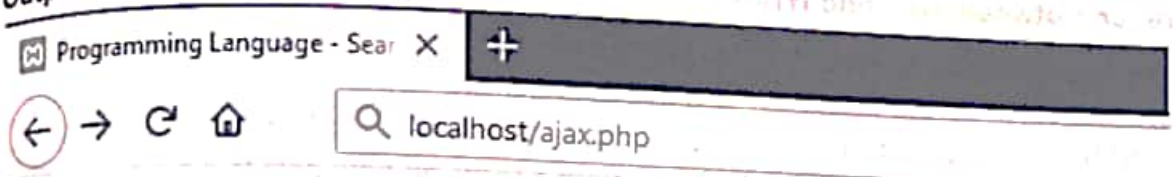
Auto_complete.js

```
functionshowName(str){
if (str.length == 0){
document.getElementById("txtName").innerHTML="";
return;
    }
if (window.XMLHttpRequest) {
xmlhttp=new XMLHttpRequest();
    } else {
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState == 4 &&xmlhttp.status == 200){
document.getElementById("txtName").innerHTML=xmlhttp.responseText;
        }
    }
xmlhttp.open("GET","language.php?name="+str,true);
xmlhttp.send();
}
```

Language.php

```
functionshowName(str){
if (str.length == 0){
document.getElementById("txtName").innerHTML="";
return;
    }
if (window.XMLHttpRequest) {
xmlhttp=new XMLHttpRequest();
    } else {
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState == 4 &&xmlhttp.status == 200){
document.getElementById("txtName").innerHTML=xmlhttp.responseText;
        }
    }
xmlhttp.open("GET","language.php?name="+str,true);
xmlhttp.send();
}
```

Output: When we type j into textbox:



Programming Language - Search

Type the first letter of the Programming Language

j

Finding: Javascript , Java

XML

XML is a markup language created by the World Wide Web Consortium (W3C) to define a syntax for encoding documents that both humans and machines could read. It does this through the use of tags that define the structure of the document, as well as how the document should be stored and transported.

It's probably easiest to compare it to another markup language with which you might be familiar—the Hypertext Markup Language (HTML) used to encode web pages. HTML uses a predefined set of markup symbols (short codes) that describe the format of content on a web page.

- Xml (eXtensible Markup Language) is a markup language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

The design goal of XML are

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.

The Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
  - XML was designed to describe data, with a focus on what data is
  - HTML was designed to display data, with a focus on how data looks
- HTML is about displaying information, while XML is about carrying information.

Features and Advantages of XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

- **XML separates data from HTML**

  If we need to display dynamic data in HTML document, it will take a lot of work to edit the HTML each time the data changes.

  With XML, data can be stored in separate XML files. This way can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

  With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

- **XML simplifies data sharing**

  In the real world, computer systems and databases contain data in incompatible formats.

  XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

  This makes it much easier to create data that can be shared by different applications.

- **XML simplifies data transport**

  One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

  Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

- **XML simplifies Platform change**

  Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

  XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

- **XML increases data availability**

  Different applications can access your data, not only in HTML pages, but also from XML data sources.

  With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people or people with other disabilities.

- **XML can be used to create new internet languages**

  A lot of new Internet languages are created with XML.

Here are some examples:

- XHTML
- WSDL for describing available web services
- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

STRUCTURE OF XML DOCUMENT

XML ELEMENTS

- XML documents must contain a **root element.** This element is "the parent" of all other elements.
- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.
- All elements can have sub elements (child elements).
- XML element contents are part of the **basic document contents.** that are **store information data.**
- XML elements are represented by a **tags.**

RULES FOR BUILDING GOOD XML

Rule 1: All XML Must Have a Root Element

A root element is simply a set of tags that contains your XML content.

```
<book>
  <author>Ernest Hemingway</author>
  <author>John Steinbeck</author>
  <author>James Joyce</author>
</book>
```

Rule 2: All Tags Must Be Closed

When a tag is declared (opened), it must also be closed. Any unclosed tags will break the code. Even tags that don't need to be closed in HTML must be closed in XML or XHTML. To open a tag, type the name of the element between less-than (<) and greater-than (>) characters, like this opening tag:

```
<author>Ernest Hemingway</author>
<p>Roses are Red</p>
<hr />
```

Rule 3: All Tags Must Be Properly Nested

When you insert (nest) one tag within another, pay attention to the order in which you open each tag, and then close the tags in the reverse order. If you open element A and then element B, you must first close B before closing A. Even HTML tags that usually will work without a strict structure must follow the stricter XML rules when they're used within an XML file.

```
<A><B>Text</B></A>
<b><i>Text</i></b>
```

Rule 4: Tag Names Have Strict Limits

Tag names can't start with the letters *xml*, a number, or punctuation, except for the underscore character (_).
The letters *XML* are used in various commands and can't start your tag name. Numbers and punctuation also aren't allowed in the beginning of the tag name.

```
<author>
<_author>
```

Rule 5: Tag Names Are Case Sensitive

Uppercase and lowercase matter in XML. Opening and closing tags must match exactly. For example, <ROOT>, <Root>, and <root> are three different tags.

```
<author>Hemingway</author>
<AUTHOR>Hemingway</AUTHOR>
```

Rule 6: Tag Names Cannot Contain Spaces

Spaces in tag names can cause all sorts of problems with data-intensive applications, so they're prohibited in XML.

Rule 7: Attribute Values Must Appear Within Quotes

Attribute values modify a tag or help identify the type of information being tagged. If you're a web designer, you may be used to the flexibility of HTML, in which some attributes don't require quotes. In XML, all attribute values must appear within quotes. For example:

```
<chapter number="1">
  <artist title="author" nationality="USA">
```

Rule 8: White Space Is Preserved

If you're in the habit of adding extra spaces and hard returns in your HTML code, watch out! Such spacing is honored by XML and can play havoc with your applications. Use extra spacing judiciously.

Rule 9: Avoid HTML Tags (Optional)

Because you can name tags anything you want, you could use tags reserved for HTML markup such as `<h1>`, `<p>`, `<li>`, and so on. Although permissible in XML, avoid using such tag names unless you want the data to be formatted that way when it's viewed in a browser window.

XML Element Name must be follow this things

- Element names must be alphabetic or numeric character contains.
- Element name can't have white spaces contains and
- name can't start with a capital letter, numeric or mixed letter.

If element contents **absence(empty)** then you can write element following **two way** to represent valid standard.

```
<element />
<element></element>
```

Nested syntax of XML Element

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and siblings are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

XML ATTRIBUTES

XML element can have **attributes to identify elements.**

XML attributes specified by ***name="value"*** pair inside the starting element. XML attribute values must be **quoted.**

XML attributes enhance the properties of the elements.

XML standard specifies element may have **define multiple attributes** along with **unique attribute name.**

```
<note id="1" type="daily"></note>
```

AVOIDING XML ATTRIBUTES

- Attributes cannot contain multiple values but child elements can have multiple values.
- Attributes cannot contain tree structure but child element can.
- Attributes are not easily expandable. If you want to change in attribute's values in the future, it may be complicated.
- Attributes cannot describe structure but child elements can.
- Attributes are more difficult to be manipulated by program code.
- Attributes values are not easy to test against a DTD, which is used to define the legal elements of an XML document.

Example of XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<note id="1">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
<note id="2">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Explanation:

- The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (UTF-8 character set).
- The next line describes the root element of the document (like saying: "this document is a note"): `<note>`

- The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

- And finally the last line defines the end of the root element.</note>

XML Comments

XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.

XML Comments add notes or lines for understanding the purpose of an XML code. Although XML is known as self-describing data but sometimes XML comments are necessary.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Students marks are uploaded by months-->
<students>
   <student>
      <name>Ratan</name>
      <marks>70</marks>
   </student>
   <student>
      <name>Aryan</name>
      <marks>60</marks>
   </student>
</students>
```

Rules for adding XML comments

- Don't use a comment before an XML declaration.
- You can use a comment anywhere in XML document except within attribute value.
- Don't nest a comment inside the other comment.

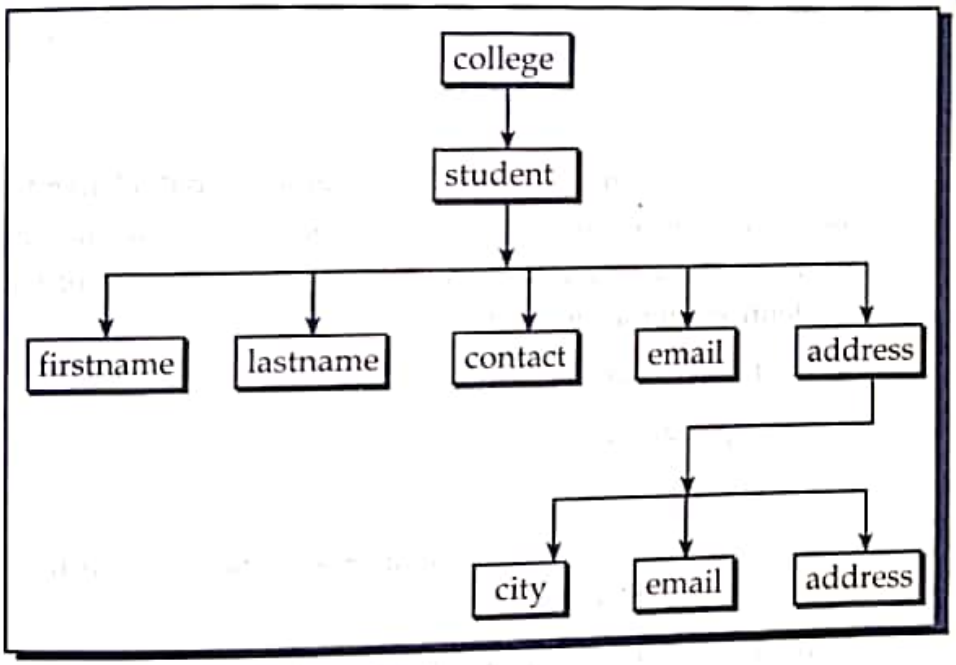
XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

```
<?xml version="1.0"?>
```

```
<college>
  <student>
      <firstname>Tamanna</firstname>
      <lastname>Bhatia</lastname>
      <contact>09990449935</contact>
      <email>tammanabhatia@abc.com</email>
      <address>
          <city>Ghaziabad</city>
          <state>Uttar Pradesh</state>
          <pin>201007</pin>
      </address>
  </student>
</college>
```



XML NAMESPACES

XML Namespaces is primary purpose to distinguish between duplicate elements and attribute names. XML data has to be exchanged between several applications. Same tag name may have different meanings in different applications. So It's create confusion on exchanging documents. Specifying prefix name to an element or attribute names to avoid this confusion.

<prefix_name:element_name>

- Elements and attributes name along with namespace have **exactly one colon.**
- **Colon before** text **prefix name** and **colon after** text **element or attribute name.**
- Each and every prefix name is **associated** with **one URI.**
- Prefix name associated with the **same URI** are in the **same namespace.**
- Fully qualified name including **prefix, colon** is called the **XML qualified name.**

Prefixes are bind to a namespace URI using xmlns:prefix attribute to the prefixed element.

```
<r:student xmlns:r="http://www.way2tutorial.com/xml/"></r:student>
```

Name Conflicts Example

Following example XML data for storing student marks,

```
<student>
  <result>
    <name>Opal Kole</name>
    <sgpa>8.1</sgpa>
    <cgpa>8.4</cgpa>
  </result>
  <cv>
    <name>Opal Kole</name>
    <cgpa>8.4</cgpa>
  </cv>
</student>
```

Above XML document both <result> and <cv> have the same <cgpa> element, so XML parser doesn't know which one is parse. That's why XML namespaces is used for mapping between an element prefix and a URI. XML namespace URI not a point to an information about the namespace but they are identify unique elements.

Convert the Name Conflict to XML Namespaces

We are specify prefix name as per different element.

xmlns attribute with XML namespaces

Specify all XML namespaces within the root element. Specification given for that element is valid for all elements occurring within scope.

The namespace declaration Syntax: xmlns:prefix_name="URI".

```
<s:student xmlns:s="http://www.way2tutorial.com/some_url1"
           xmlns:res="http://www.way2tutorial.com/some_url2">
  <r:result>
    <r:name>Opal Kole</r:name>
    <r:sgpa>8.1</r:sgpa>
    <r:cgpa>8.4</r:cgpa>
  </r:result>
  <res:cv>
    <res:name>Opal Kole</res:name>
    <res:cgpa>8.4</res:cgpa>
  </res:cv>
</s:student>
```

DTD (Document Type Definition)

DTD stands for *Document Type Definition*. A DTD allows you to create rules for the elements within your XML documents. Although XML itself has rules, the rules defined in a DTD are specific to your own needs.

So, for an XML document to be well-formed, it needs to use correct XML syntax, and it needs to conform to its DTD or schema

Do I Need to Create a DTD?

If you have created your own XML elements, attributes, and/or entities, then you should create a DTD.

If you are creating an XML document using predefined elements/attributes/entities (i.e. ones that have been created by someone else), then a DTD should already exist. All you need to do is link to that DTD using the DOCTYPE declaration.

What's in a DTD?

A DTD consists of a list of syntax definitions for each element in your XML document.

When you create a DTD, you are creating the syntax rules for any XML document that uses the DTD. You are specifying which element names can be included in the document, the attributes that each element can have, whether or not these are required or optional, and more.

```
DTD <!DOCTYPE>
```

To use a DTD within your XML document, you need to declare it. The DTD can either be internal (written into the same document that it's being used in), or external (located in another document).

You declare a DTD at the top of your XML document (in the prolog) using the !DOCTYPE declaration. The basic syntax is:

```
<!DOCTYPE rootname [DTD]>
```

- where, rootname is the root element, and [DTD] is the actual definition.

DTD Variations

```
<!DOCTYPE rootname [DTD]>
```

- This is an internal DTD (the DTD is defined between the square brackets within the XML document).

```
<!DOCTYPE tutorials [
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
]>
```

```
<!DOCTYPE rootname SYSTEM URL>
<!DOCTYPE rootname SYSTEM URL>
```

- The keyword SYSTEM indicates that it's a private DTD (not for public distribution).
- The presence of URL and [DTD] together indicates that this is both an external and internal DTD (part of the DTD is defined in a document located at the URL, the other part is defined within the XML document).

<!DOCTYPE tutorials SYSTEM "tutorials.dtd">

<!DOCTYPE *rootname* SYSTEM *URL* [*DTD*]>

- The keyword SYSTEM indicates that it's a private DTD (not for public distribution).
- The presence of URL and [DTD] together indicates that this is both an external and internal DTD (part of the DTD is defined in a document located at the URL, the other part is defined within the XML document).

```
<!DOCTYPE tutorials SYSTEM "tutorials.dtd" [
<!ELEMENT tutorial (summary)>
<!ELEMENT summary (#PCDATA)>
]>
```

<!DOCTYPE *rootname* PUBLIC *identifier URL*>

- The keyword PUBLIC indicates that it's a public DTD (for public distribution).
- The presence of URL indicates that this is an external DTD (the DTD is defined in a document located at the URL).
- The *identifier* indicates the *formal public identifier* and is required when using a public DTD.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<!DOCTYPE rootname PUBLIC identifier URL[DTD]>
```

- The keyword PUBLIC indicates that it's a public DTD (for public distribution).
- The presence of URL and [DTD] together indicates that this is both an external and internal DTD (part of the DTD is defined in a document located at the URL, the other part is defined within the XML document).
- The *identifier* indicates the *formal public identifier* and is required when using a public DTD.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
]>
```

INTERNAL DTD

Whether you use an external or internal DTD, the actual syntax for the DTD is the same – the same code could just as easily be part of an internal DTD or an external one. The only difference between internal and external is in the way it's declared with DOCTYPE.

Using an internal DTD, the code is placed between the DOCTYPE tags (eg, `<!DOCTYPE tutorials [ and ]>`.

Example Internal DTD

This is an example of an internal DTD. It's internal because the DTD is included in the target XML document:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE tutorials [
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
]>
<tutorials type="web">
  <tutorial>
    <name>XML Tutorial</name>
    <url>https://www.quackit.com/xml/tutorial</url>
  </tutorial>
  <tutorial>
    <name>HTML Tutorial</name>
    <url>https://www.quackit.com/html/tutorial</url>
  </tutorial>
</tutorials>
```

EXTERNAL DTD

An external DTD is one that resides in a separate document.

To use the external DTD, you need to link to it from your XML document by providing the URI of the DTD file. This URI is typically in the form of a URL. The URL can point to a local file using a relative reference, or a remote one (eg, using HTTP) using an absolute reference.

Example: External_DTD

Here's an example of an XML document that uses an external DTD. Note that the standalone attribute is set to no. This is because the document relies on an external resource (the DTD):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE tutorials SYSTEM "tutorials.dtd">
<tutorials type="web">
  <tutorial>
    <name>XML Tutorial</name>
    <url>https://www.quackit.com/xml/tutorial</url>
  </tutorial>
  <tutorial>
    <name>HTML Tutorial</name>
    <url>https://www.quackit.com/html/tutorial</url>
  </tutorial>
</tutorials>
```

And, using the above XML document as an example, here's an example of what tutorials.dtd (the external DTD file) could look like. Note that the external DTD file doesn't need the DOCTYPE declaration – it is already on the XML file that is using this DTD:

```
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
```

COMBINED DTD

You can use both an internal DTD and an external one at the same time. This could be useful if you need to adhere to a common DTD, but also need to define your own definitions locally.

Example of Combined DTD

This is an example of using both an external DTD and an internal one for the same XML document. The external DTD resides in tutorials.dtd and is called first in the DOCTYPE declaration. The internal DTD follows the external one but still resides within the DOCTYPE declaration:

Here, I've added a new element called summary. This element must be present under the tutorial element. Because this element hasn't been defined in the external DTD, I need to define it internally. Once again, we're setting the standalone attribute to no because we rely on an external resource.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE tutorials SYSTEM "tutorials.dtd" [
<!ELEMENT tutorial (summary)>
<!ELEMENT summary (#PCDATA)>
]>
<tutorials>
  <tutorial>
    <name>XML Tutorial</name>
    <url>https://www.quackit.com/xml/tutorial</url>
```

```
    <summary>Best XML tutorial on the web!</summary>
  </tutorial>
  <tutorial>
    <name>HTML Tutorial</name>
    <url>https://www.quackit.com/html/tutorial</url>
    <summary>Best HTML tutorial on the web!</summary>
  </tutorial>
</tutorials>
```

DTD Formal Public Identifier (FPI)

When declaring a DTD available for public use, you need to use the PUBLIC keyword within your DOCTYPE declaration. When you use the PUBLIC keyword, you also need to use an FPI (which stands for Formal Public Identifier).

FPI Syntax:

An FPI is made up of 4 fields, each separated by double forward slashes (//):

field 1//field 2//field 3//field 4

FPI Example

Here's a real life example of an FPI. In this case, the DTD was created by the W3C for XHTML:

-//W3C//DTD XHTML 1.0 Transitional//EN

FPI Fields

An FPI must contain the following fields:

Field	Example	Description
Separator	//	This is used to separate the different fields of the FPI.
First field	-	Indicates whether the DTD is connected to a formal standard or not. If the DTD hasn't been approved (for example, you've defined the DTD yourself), use a hyphen (-). If the DTD has been approved by a non standards body, use a plus sign "+". If the DTD has been approved by a formal standards body this field should be a reference to the standard itself.
Second field	W3C	Holds the name of the group (or person) responsible for the DTD. The above example is maintained by the W3C, so "W3C" appears in the second field.
Third field	DTD XHTML 1.0 Transitional	Indicates the type of document that is being described. This usually contains some form of unique identifier (such as a version number).
Fourth field	EN	Specifies the language that the DTD uses. This is achieved by using the two letter identifier for the language (i.e. for english, use "EN").

FPI DOCTYPE Syntax

When using a public DTD, place the FPI between the PUBLIC keyword and the URI/URL.

```
<!DOCTYPE rootname PUBLIC FPI URL>
```

FPI DOCTYPE Example

You can see an example of an FPI in the following DOCTYPE declaration (the FPI is in bold):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

DTD Elements

Creating a DTD is quite straight forward. It's really just a matter of defining your elements, attributes, and/or entities.

To define an element in your DTD, you use the `<!ELEMENT>` declaration. The actual contents of your `<!ELEMENT>` declaration will depend on the syntax rules you need to apply to your element.

Basic Syntax

The `<!ELEMENT>` declaration has the following syntax:

```
<!ELEMENT element_name content_model>
```

Here, element_name is the name of the element you're defining. The content model could indicate a specific rule, data or another element.

- If it specifies a rule, it will be set to either ANY or EMPTY.
- If specifies data or another element, the data type/element name needs to be surrounded by brackets (i.e. (tutorial) or (#PCDATA)).

The following examples show you how to use this syntax for defining your elements.

Plain Text

If an element should contain plain text, you define the element using #PCDATA. PCDATA stands for Parsed Character Data and is the way you specify non-markup text in your DTDs. Using this example - `<name>XML Tutorial</name>` – the XML Tutorial part is the PCDATA. The other part consists of markup.

Syntax:

```
<!ELEMENT element_name (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<name>XML Tutorial</name>
```

Unrestricted Elements

If it doesn't matter what your element contains, you can create an element using the content_model of ANY. Note that doing this removes all syntax checking, so you should avoid using this if possible. You're better off defining a specific content model.

```
<!ELEMENT element_name ANY>
<!ELEMENT tutorials ANY>
```

Empty Elements

You might remember that an empty element is one without a closing tag. For example, in XHTML, the `<br />` and `<img />` tags are empty elements. Here's how you define an empty element:

Syntax:

```
<!ELEMENT element_name EMPTY>
<!ELEMENT header EMPTY>
<header />
```

Child Elements

You can specify that an element must contain another element, by providing the name of the element it must contain. Here's how you do that:

```
<!ELEMENT element_name (child_element_name)>
<!ELEMENT tutorials (tutorial)>
<tutorials>
  <tutorial></tutorial>
</tutorials>
```

Multiple Child Elements (Sequences)

You can also provide a comma separated list of elements if it needs to contain more than one element. This is referred to as a sequence. The XML document must contain the tags in the same order that they're specified in the sequence.

```
<!ELEMENT element_name (child_element_name, child_element_name,...)>
<!ELEMENT tutorial (name, url)>
<tutorials>
  <tutorial>
    <name></name>
    <url></url>
  </tutorial>
</tutorials>
```

DTD Element Operators

An element (tutorials) must contain one instance of another element (tutorial). This is fine if there only needs one instance of tutorial, but what if we didn't want a limit. What if the tutorials element should be able to contain any number of tutorial instances? Fortunately we can do that using DTD operators.

Here's a list of operators/syntax rules we can use when defining child elements:

Operator	Syntax	Description
+	*a*+	One or more occurrences of *a*
*	*a**	Zero or more occurences of *a*
?	*a*?	Either *a* or nothing
,	*a*, *b*	*a* followed by *b*
|	*a* | *b*	*a* or *b*
()	(expression)	An expression surrounded by parentheses is treated as a unit and could have any one of the following suffixes?, *, or +.

Zero or More

To allow zero or more of the same child element, use an asterisk (*):

<!ELEMENT element_name (child_element_name*)>

<!ELEMENT tutorials (tutorial*)>

One or More

To allow one or more of the same child element, use a plus sign (+):

```
<!ELEMENT element_name (child_element_name+)>
<!ELEMENT tutorials (tutorial+)>
```

Zero or One

To allow either zero or one of the same child elements, use a question mark (?):

```
<!ELEMENT element_name (child_element_name?)>
<!ELEMENT tutorials (tutorial?)>
```

Choices

You can define a choice between one or another element by using the pipe (|) operator. For example, if the tutorial element requires a child called either name, title, or subject (but only one of these), you can do the following:

```
<!ELEMENT element_name (choice_1 | choice_2 | choice_3)>
<!ELEMENT tutorial (name | title | subject)>
```

Mixed Content

You can use the pipe (|) operator to specify that an element can contain both PCDATA and other elements:

```
<!ELEMENT element_name (#PCDATA | child_element_name)>
<!ELEMENT tutorial (#PCDATA | name | title | subject)*>
```

DTD Operators with Sequences

You can apply any of the DTD operators to a sequence:

```
<!ELEMENT element_name (child_element_name dtd_operator,
child_element_name dtd_operator,...)>
<!ELEMENT tutorial (name+, url?)>
```

The above example allows the tutorial element to contain one or more instance of the name element, and zero or one instance of the url element.

Subsequences

You can use parentheses to create a subsequence (i.e. a sequence within a sequence). This enables you to apply DTD operators to a subsequence:

```
<!ELEMENT element_name ((sequence) dtd_operator sequence)>
<!ELEMENT tutorial ((author,rating?)+ name, url*)>
```

The above example specifies that the tutorial element can contain one or more author elements with each occurence having an optional rating element.

DTD Attributes

Just as you need to define all elements in your DTD, you also need to define any attributes they use. You use the `<!ATTLIST>` declaration to define attributes in your DTD.

Syntax

You use a single `<!ATTLIST>` declaration to declare all attributes for a given element. In other words, for each element (that contains attributes), you only need one `<!ATTLIST>` declaration.

The `<!ATTLIST>` declaration has the following syntax:

```
<!ATTLIST element_name
  attribute_name TYPE DEFAULT_VALUE
  attribute_name TYPE DEFAULT_VALUE
  attribute_name TYPE DEFAULT_VALUE
  ...>
```

Here, element_name refers to the element that you're defining attributes for, attribute_name is the name of the attribute that you're declaring, TYPE is the attribute type, and DEFAULT_VALUE is it's default value.

```
<!ATTLIST tutorial  published CDATA "No">
```

Here, we are defining an attribute called published for the tutorial element. The attribute's type is CDATA and it's default value is No.

DTD Attribute Default Values

The attribute TYPE field can be set to one of the following values:

Value	Description
value	A simple text value, enclosed in quotes.
#IMPLIED	Specifies that there is no default value for this attribute, and that the attribute is optional.
#REQUIRED	There is no default value for this attribute, but a a value must be assigned.
#FIXED *value*	The *#FIXED* part specifies that the value must be the value provided. The *value* part represents the actual value.

value

You can provide an actual value to be the default value by placing it in quotes.

```
<!ATTLIST element_name attribute_name CDATA "default_value">
<!ATTLIST tutorial published CDATA "No">
```

#REQUIRED

The #REQUIRED keyword specifies that you won't be providing a default value, but that you require that anyone using this DTD does provide one.

```
<!ATTLIST element_name attribute_name CDATA #REQUIRED>
<!ATTLIST tutorial published CDATA #REQUIRED>
```

#IMPLIED

The #IMPLIED keyword specifies that you won't be providing a default value, and that the attribute is optional for users of this DTD.

```
<!ATTLIST element_name attribute_name CDATA #IMPLIED>
```

```
<!ATTLIST tutorial rating CDATA #IMPLIED>
```

#FIXED

The #FIXED keyword specifies that you will provide value, and that's the only value that can be used by users of this DTD.

```
<!ATTLIST element_name attribute_name CDATA #FIXED "value">
```

```
<!ATTLIST tutorial language CDATA #FIXED "EN">
```

DTD Attribute Types

CDATA is probably the most common attribute type as it allows for plain text to be used for the attribute's value. There may however, be cases where you need to use a different attribute type.

When setting attributes for your elements, the attribute TYPE field can be set to one of the following values.

Type	Description
CDATA	Character Data (text that doesn't contain markup)
ENTITY	The name of an entity (which must be declared in the DTD)
ENTITIES	A list of entity names, separated by whitespaces. (All entities must be declared in the DTD)
Enumerated	A list of values. The value of the attribute must be one from this list.
ID	A unique ID or name. Must be a valid XML name.
IDREF	Represents the value of an ID attribute of another element.
IDREFS	Represents multiple IDs of elements, separated by whitespace.
NMTOKEN	A valid XML name.
NMTOKENS	A list of valid XML names, separated by whitespace.
NOTATION	A notation name (which must be declared in the DTD).

CDATA

As with all attribute types, the attribute type of CDATA is placed after the attribute name and before the default value.

Syntax:

```
<!ATTLIST element_name attribute_name CDATA default_value>
```

Example : DTD

```
<!ATTLIST mountain country CDATA "New Zealand">
```

Example: XML

```
<mountains>
  <mountain country="New Zealand">
    <name>Mount Cook</name>
  </mountain>
  <mountain country="Australia">
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

Enumerated

The enumerated attribute type provides for a list of possible values. This enables the DTD user to provide one value from the list of possible values.

The values must be surrounded by parentheses, and each value must be separated by a pipe (|).

Syntax:

```
<!ATTLIST element_name attribute_name (value1 | value2 | value3)
default_value>
```

Example: DTD

```
<!ATTLIST tutorial published (yes | no) "no">
```

Example: XML

```
<tutorials>
  <tutorial published="yes">
    <name>XML Tutorial</name>
  </tutorial>
  <tutorial published="no">
    <name>HTML Tutorial</name>
  </tutorial>
  <tutorial>
    <name>CSS Tutorial</name>
  </tutorial>
</tutorials>
```

ENTITY

The attribute type of ENTITY is used for referring to the name of an entity you've declared in your DTD.

Syntax:

```
<!ATTLIST element_name attribute_name ENTITY default_value>
```

Example: DTD

```
<!ATTLIST mountain photo ENTITY #IMPLIED>
<!ENTITY mt_cook_1 SYSTEM "mt_cook1.jpg">
```

Example: XML

```
<mountains>
  <mountain photo="mt_cook_1">
    <name>Mount Cook</name>
  </mountain>
  <mountain>
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

ENTITIES

The attribute type of ENTITIES allows you to refer to multiple entity names, separated by a space.

Syntax

```
<!ATTLIST element_name attribute_name ENTITIES default_value>
```

Example: DTD

```
<!ATTLIST mountain photo ENTITIES #IMPLIED>
<!ENTITY mt_cook_1 SYSTEM "mt_cook1.jpg">
<!ENTITY mt_cook_2 SYSTEM "mt_cook2.jpg">
```

Example: XML

```
<mountains>
  <mountain photo="mt_cook_1 mt_cook_2">
    <name>Mount Cook</name>
  </mountain>
  <mountain>
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

ID

The attribute type of ID is used specifically to identify elements.

NOTE: Because of this, no two elements can contain the same value for attributes of type ID. Also, you can only give an element one attribute of type ID. The value that is assigned to an attribute of type ID must be a valid XML name.

Syntax

```
<!ATTLIST element_name attribute_name ID default_value>
```

Example: DTD

```
<!ATTLIST mountain mountain_id ID #REQUIRED>
```

Example: XML

```
<mountains>
  <mountain mountain_id="m10001">
    <name>Mount Cook</name>
  </mountain>
  <mountain  mountain_id="m10002">
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

IDREF

The attribute type of IDREF is used for referring to an ID value of another element in the document.

Syntax:

```
<!ATTLIST element_name attribute_name IDREF default_value>
```

Example: DTD

```
<!ATTLIST employee employee_id ID #REQUIRED manager_id IDREF #IMPLIED>
```

Example: XML

```
<employees>
  <employee employee_id="e10001" manager_id="e10002">
    <first_name>Homer</first_name>
    <last_name>Flinstone</last_name>
  </employee>
  <employee employee_id="e10002">
    <first_name>Fred</first_name>
    <last_name>Burns</last_name>
  </employee>
</employees>
```

IDREFS

The attribute type of IDREFS is used for referring to the ID values of more than one other element in the document. Each value is separated by a space.

Syntax:

```
<!ATTLIST element_name attribute_name IDREFS default_value>
```

Example: DTD

```
<!ATTLIST individual individual_id ID #REQUIRED parent_id IDREFS
#IMPLIED>
```

Example: XML

```
<individuals>
  <individual individual_id="e10001" parent_id="e10002 e10003">
    <first_name>Bart</first_name>
    <last_name>Simpson</last_name>
  </individual>
  <individual individual_id="e10002">
    <first_name>Homer</first_name>
    <last_name>Simpson</last_name>
  </individual>
  <individual individual_id="e10003">
    <first_name>Marge</first_name>
    <last_name>Simpson</last_name>
  </individual>
</individuals>
```

NMTOKEN

An NMTOKEN (name token) is any mixture of Name characters. It cannot contain whitespace (although leading or trailing whitespace will be trimmed/ignored).

While Names have restrictions on the initial character (the first character of a Name cannot include digits, diacritics, the full stop and the hyphen), the NMTOKENdoesn't have these restrictions.

Syntax:

```
<!ATTLIST element_name attribute_name NMTOKEN default_value>
```

Example: DTD

```
<!ATTLIST mountain country NMTOKEN #REQUIRED>
```

Example: XML

```
<mountains>
  <mountain country="NZ">
    <name>Mount Cook</name>
  </mountain>
  <mountain country="AU">
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

NMTOKENS

The attribute type of NMTOKENS allows the attribute value to be made up of multiple NMTOKENSs, separated by a space.

Syntax

```
<!ATTLIST element_name attribute_name NMTOKENS default_value>
```

Example: DTD

```
<!ATTLIST mountains country NMTOKENS #REQUIRED>
```

Example: XML

```
<mountains country="NZ AU">
  <mountain>
    <name>Mount Cook</name>
  </mountain>
  <mountain>
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

NOTATION

The attribute type of NOTATION allows you to use a value that has been declared as a notation in the DTD.A notation is used to specify the format of non-XML data. A common use of notations is to describe MIME types such as image/gif, image/jpeg etc.

Syntax

```
<!NOTATION name SYSTEM "external_id">
<!ATTLIST element_name attribute_name NOTATION default_value>
```

Example: DTD

```
<!NOTATION GIF SYSTEM "image/gif">
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION PNG SYSTEM "image/png">
<!ATTLIST mountain
  photo ENTITY #IMPLIED
  photo_type NOTATION (GIF | JPG | PNG) #IMPLIED>
<!ENTITY mt_cook_1 SYSTEM "mt_cook1.jpg">
```

Example: XML

```
<mountains>
  <mountain photo="mt_cook_1" photo_type="JPG">
    <name>Mount Cook</name>
  </mountain>
  <mountain>
    <name>Cradle Mountain</name>
  </mountain>
</mountains>
```

Problems with DTDs

- Written in a language other than XML; so need a separate parser.
- All definitions in a DTD are global, applying to the entire document. Cannot have two elements with the same name but with different content in separate contexts.
- The text content of an element can be PCDATA only; no finer typing for numbers or special string formats.
- Limited typing for attribute values.
- DTDs are not truly aware of namespaces; they recognize prefixes but not the underlying URI.

XML Schema

XML Schema is an XML-based language used to create XML-based languages and data models. An XML schema defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

XML documents that attempt to adhere to an XML schema are said to be instances of that schema. If they correctly adhere to the schema, then they are valid instances. This is not the same as being well formed. A well-formed XML document follows all the syntax rules of XML, but it does not necessarily adhere to any particular schema. So, an XML document can be well formed without being valid, but it cannot be valid unless it is well formed.

An XML schema describes the structure of an XML instance document by defining what each element must or may contain.

Form of an XML Schema Definition

An XML Schema is also an XML document with XSD extension.

First item: xml declaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML comments and processing instructions are allowed.

Root element: schema with a namespace declaration.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

Possible namespace prefixes: xs, xsd, or none.

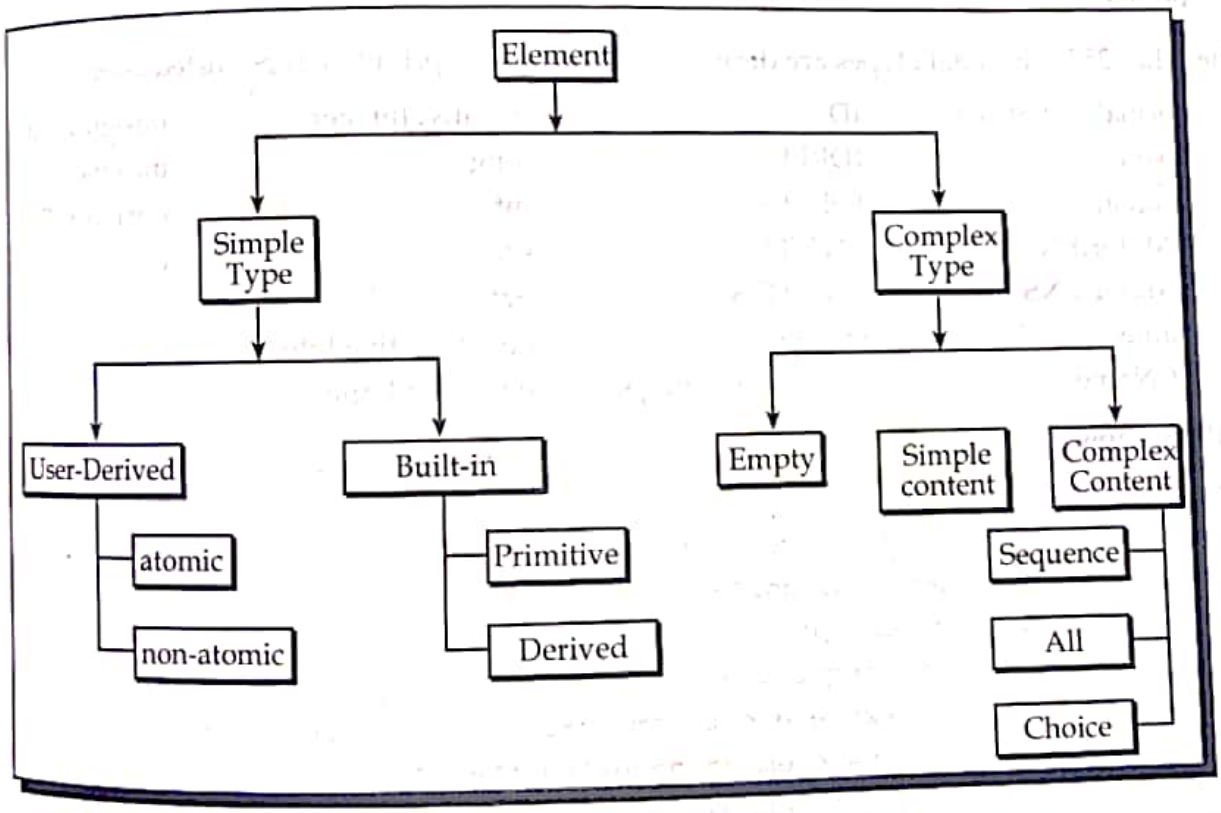
The following is a high-level overview of Schema types.

- Elements can be of simple type or a complex type.
- Simple type elements can only contain text. They can not have child elements or attributes.
- All the built-in types are simple types (e.g, xs:string).
- Schema authors can derive simple types by restricting another simple type. For example, an email type could be derived by limiting a string to a specific pattern.
- Simple types can be atomic (e.g, strings and integers) or non-atomic (e.g, lists).

- Complex-type elements can contain child elements and attributes as well as text.
- By default, complex-type elements have complex content, meaning that they have child elements.
- Complex-type elements can be limited to having simple content, meaning they only contain text. They are different from simple type elements in that they have attributes.
- Complex types can be limited to having no content, meaning they are empty, but they may have attributes.

1. Complex types may have mixed content - a combination of text and child elements.

Schema Elements



XML Schema Element Declaration

Elements are declared using an element named xs:element with an attribute that gives the name of the element being defined.

Element declarations can be one of two sorts.

Simple Type

Simple-type elements have no children or attributes. Content of these elements can be text only.A simple-type element is defined using the type attribute.

Examples

```
<xs:element name="item" type="xs:string"/>
<xs:element name="price" type="xs:decimal"/>
```

The values xs:string and xs:decimal are two different simple types predefined in the XML Schema language.

XML Schema specifies 44 built-in types, 19 of which are primitive.

The 19 built-in primitive types are listed below.

string	duration	gYear	base64Binary
boolean	dateTime	gMonthDay	anyURI
decimal	time	gDay	QName
float	date	gMonth	NOTATION
double	gYearMonth	hexBinary	

The other 25 built-in data types are derived from one of the primitive types listed above.

normalized String	ID	negativeInteger	unsigned Int
token	IDREF	long	unsigned Short
language	IDREFS	int	unsigned Byte
NMTOKEN	ENTITY	short	positive Integer
NMTOKENS	ENTITIES	byte	
Name	integer	non Negative Integer	
NCName	nonPositiveInteger	unsigned Long	

Code Sample

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="Author">
            <xs:complexType>
                  <xs:sequence>
                     <xs:element name="FirstName" type="xs:string"/>
                     <xs:element name="LastName" type="xs:string"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
Xml for above code:
<Author>
      <FirstName>Ram</FirstName>
<LastName>Shah</LastName>
</Author>
```

Notice the First Name and Last Name elements in the code sample above. They are not explicitly defined as simple type elements. Instead, the type is defined with the type attribute. Because the value (string in both cases) is a simple type, the elements themselves are simple-type elements.

COMPLEX TYPE

Complex-type elements have attributes, child elements, or some combination of both. As the above diagram shows, a complex-type element can be empty, contain simple content such as a string, or can contain complex content such as a sequence of elements.

Order Indicator: Content Models

Content models are used to indicate the structure and order in which child elements can appear within their parent element. Content models are made up of model groups. The three types of model groups are listed below.

1. xs:sequence - the elements must appear in the order specified.
2. xs:all - the elements must appear, but the order is not important.
3. xs:choice - only one of the elements can appear.

Xs:sequence

The following sample shows the syntax for declaring a complex-type element as a sequence, meaning that the elements must show up in the order they are declared.

```
<xs:element name="ElementName">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Child1" type="xs:string"/>
            <xs:element name="Child2" type="xs:string"/>
            <xs:element name="Child3" type="xs:string"/
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Xs:all

The following sample shows the syntax for declaring a complex-type element as a conjunction, meaning that the elements can show up in any order.

```
<xs:element name="ElementName">
    <xs:complexType>
        <xs:all>
            <xs:element name="Child1" type="xs:string"/>
            <xs:element name="Child2" type="xs:string"/>
            <xs:element name="Child3" type="xs:string"/
        </xs:all>
    </xs:complexType>
</xs:element>
```

xs:choice

The following sample shows the syntax for declaring a complex-type element as a choice, meaning that only one of the child elements may show up.

```
<xs:element name="ElementName">
      <xs:complexType>
            <xs:choice>
                  <xs:element name="Child1" type="xs:string"/>
                  <xs:element name="Child2" type="xs:string"/>
                  <xs:element name="Child3" type="xs:string"/>
            </xs:choice>
      </xs:complexType>
</xs:element>
```

Example:

```
<xs:element name="location">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="city" type="xs:string"/>
                  <xs:element name="state" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

The element xs:sequence is one of several ways to combine elements in the content.

Corresponding DTD: <!ELEMENT location (city, state)>

Corresponding XML:

```
<location>
      <city>Kathmandu</city>
      <state>Province 3</state>
</location>
```

An xs:element element may also have attributes that specify the number of occurrences of the element at this position in the sequence.

minOccurs="0" // default = 1

maxOccurs="5" // default = maximum(1, minOccurs)

maxOccurs="unbounded"

Example:

Sample DTD

```
<!ELEMENT phoneNumbers (title, entries)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT entries (entry*)>
<!ELEMENT entry (name, phone, city?)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

Convert DTD to XML Schema:

We use xs:string in place of PCDATA. Element sequencing is handled with xs:sequence. The attributes minOccurs and maxOccurs take care of the * and ? in the DTD.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>?
  <xs:element name="phoneNumber">
    <xs: complex Type>
     <xs: sequence>
       <xs:element name = "title" type="xs:string"/>
       <xs:element name = "entries">
        <xs:complex Type>
          <xs: sequence>
          <xs: element name = "entry" min Occurs = "0"
                                maxOccurs="unbounded">
      <xs: complex Type>
        <xs: sequnece>
          <xs: element name = "name">
          <xs: complex Type>
          <xs: element name = "first" type =" sx: string"/>
          <xs: element name = "middle" type =" xs:string" min Occurs ="0"/>
          <xs: element name = "last" type = "sx: string"/>
          </xs: sequence>
        </xs: complex Type>
      </xs: element>
        </xs: element name = "phone" type = "xs: string"/>
       </xs: element name = "city" type = "xs: string" min Occurs = "0"/>
    </xs: sequence>
   <xs: complex Type>
 </xs: element>
</xs: sequence>
   </xs: complex Type>
 </xs: element>
</xs: schema>
```

CREATE NAMED TYPES

Define the complex types in the XML Schema definition and give them each a name. These definitions will lie at the top level of the schema element.
The scope of each complex type definition covers the entire schema element (order is of no consequence).
Naming Convention: Use the suffix "Type" when defining a new type in the XML Schema definition.

```
<xs:simpleType name="SalaryType">
        <xs:restriction base="xs:decimal">
            <xs:minInclusive value="10000"/>
            <xs:maxInclusive value="90000"/>
        </xs:restriction>
    </xs:simpleType>
```

To use this :

```
<xs:element name="Salary" type="SalaryType"/>
```

Example: Named Element:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="nameType">
        <xs:sequence>
            <xs:element name="first" type="xs:string"/>
            <xs:element name="middle" type="xs:string" minOccurs="0"/>
            <xs:element name="last" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="entryType">
        <xs:sequence>
            <xs:element name="name" type="nameType"/>
            <xs:element name="phone" type="xs:string"/>
            <xs:element name="city" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="entriesType">
        <xs:sequence>
            <xs:element name="entry" type="entryType" minOccurs="0"
        maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="phoneType">
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="entries" type="entriesType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="phoneNumbers" type="phoneType"/>
</xs:schema>
```

MIXED CONTENT

Mixed content refers to the situation where an element has both text and elements in its content. Because of the sub-elements, the element being defined must be a complex type.

To allow mixed content in an element definition, simply add an attribute mixed to the xs:complexType starting tag that asserts:

mixed="true"

This attribute has a default value of "false".

XSD:

```
<xs:element name="narrative">
<xs:complexType mixed="true">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="bold" type="xs:string"/>
<xs:element name="italics" type="xs:string/>
<xs:element name="underline" type="xs:string/>
</xs:choice>
</xs:complexType>
</xs:element>
```

XML:

```
<narrative>
```

Higher beings from `<italics>`outer space`</italics>` may not want to tell us the `<underline>`secrets of life `</underline>`because we're not ready. But maybe they'll change their tune after a little `<bold>`torture`</bold><italics>`Jack Handey`</italics>`

```
</narrative>
```

ATTRIBUTE SPECIFICATION

Attributes are defined using the element xs:attribute with its own attributes, name and type.

The xs:attribute element must lie inside a complex type specification, but the type of the new attribute (its value) must be a simple type. Attribute values may not contain elements or other attributes.

XSD:

```
<xs:element name="name">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="first" type="xs:string"/>
            <xs:element name="middle" type="xs:string" minOccurs="0"/>
            <xs:element name="last" type="xs:string"/>
        </xs:sequence>
    <xs:attribute name="gender" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

XML:

```
<name gender="male">
      <first>Ram</first>
      <middle>Kumar</middle>
      <last>Nepali</last>
</name>
```

In above example name have gender attribute with value male.

The attribute specification(s) must lie inside a complex type, but they must fall after the structure definition (xs:sequence, xs:all, or xs:choice) in the complex type.

Attributes with Empty Elements

These specifications are easy. The only content inside the xs:complexType element will be the attribute definitions. This complex type definition can be an anonymous type or a named type.

```
<xs:complexType>
      <xs:attribute name="src" type="xs:anyURI"/>
      <xs:attribute name="width" type="xs:integer"/>
      <xs:attribute name="height" type="xs:integer"/>
</xs:complexType>
```

Attributes for Elements with Only Text Content

These specifications are a bit more complicated. We need a complex type to allow for attribute definitions, but we need a simple type to specify the text content.

The solution is to place an xs:simpleContent element inside of the complex type and use an xs:extension element to specify the simple type.

```
<xs:element name="bookTitle">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:string">
                        <xs:attribute name="author" type="xs:string"/>
                        <xs:attribute name="isbn" type="xs:string"/>
                  </xs:extension>
            </xs:simpleContent>
      </xs:complexType>
<xs:element>
```

Default and Fixed Value for Attribute:

Default Values

Attributes can have default values. To specify a default value, use the default attribute of the xs:attribute element. Default values for attributes work slightly differently than they do for elements. If the attribute is not included in the instance document, the schema processor inserts it with the default value.

```
<xs:element name="FirstName">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="Full"
type="xs:boolean"       default="true"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

Fixed Values

Attribute values can be fixed, meaning that, if they appear in the instance document, they must contain a specified value. Like with simple-type elements, this is done with the fixed attribute.

```
<xs:element name="Name">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="FirstName">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">

                                                    <xs:attribute
                    name="Full" type="xs:boolean" default="true"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="LastName" type="xs:string"/>
        </xs:sequence>
        <xs:attribute   name="Pseudonym"   type="xs:boolean"
fixed="true"/>
        <xs:attribute name="HomePage" type="xs:anyURI"/>
    </xs:complexType>
</xs:element>
```

Requiring Attributes

By default, attributes are optional, but they can be required by setting the use attribute of xs:attribute to required as shown below.

```
<xs:attribute name="HomePage" type="xs:anyURI" use="required"/>
```

Restrictions on Content(facets):

When an XML element or attribute has a data type defined, it puts restrictions on the element or attribute's content.

If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate. With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets.

XSD RESTRICTIONS/ FACETS

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
      <xs:simpleType>
            <xs:restriction base="xs:integer">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="120"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint. The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="Audi"/>
                  <xs:enumeration value="Golf"/>
                  <xs:enumeration value="BMW"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

The example above could also have been written like this:

```
<xs:element name="car" type="carType"/>
      <xs:simpleType name="carType">
            <xs:restriction base="xs:string">
                  <xs:enumeration value="Audi"/>
                  <xs:enumeration value="Golf"/>
                  <xs:enumeration value="BMW"/>
      </xs:restriction>
</xs:simpleType>
```

Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        </xs:restriction>
        </xs:simpleType>
</xs:element>
```

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[xyz]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element
```

The next example defines an element called "zipcode" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="zipcode">
      <xs:simpleType>
            <xs:restriction base="xs:integer">
                  <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element >
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the white Space constraint. This example defines an element called "address" with a restriction. The white Space constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:whiteSpace value="preserve"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

```
<xs:element name="address">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:whiteSpace value="replace"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

```
<xs:element name="address">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:whiteSpace value="collapse"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

Restrictions on Length:

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints. This example defines an element called "password" with a restriction. The value must be exactly eight characters:

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="5"/>
            <xs:maxLength value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restrictions for Data types

Constraint	Description
Enumeration	Defines a list of acceptable values
fraction Digits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
Length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
max Exclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
max Inclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
max Length	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
min Exclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
min Inclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
min Length	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
Pattern	Defines the exact sequence of characters that are acceptable
total Digits	Specifies the exact number of digits allowed. Must be greater than zero
white Space	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XSL (Extensible Style sheet Language)/XSLT (XSL Transformations)

Before learning XSLT, we should first understand XSL which stands for Extensible Style sheet Language. It is similar to XML as CSS is to HTML.

XSL is a powerful language for applying styles to XML documents.

XML also has it's own styles language - XSL. XSL stands for Extensible Styles Language and is a very powerful language for applying styles to XML documents. XSL has two parts – a formatting language and a transformation language.

The formatting language allows you to apply styles similar to what CSS does. Browser support for the XSL formatting language is limited at this stage.

The transformation language is known as XSLT (XSL Transformations). XSLT allows you to transform your XML document into another form. For example, you could use XSLT to dynamically output some (or all) of the contents of your XML file into an HTML document containing other content.

XSLT, which stands for Extensible Styles Language Transformations, enables you to transform XML documents into another form. For example, you can take your XML document, combine it with HTML/CSS, and it will look completely different when viewing it in your user agent/browser.

XSLT Documents

An XSLT document is a valid XML document. An XSLT document consists of a number of elements/tags/attributes. These can be XSL elements or elements from another language (such as HTML). When you look at an XSLT document, you will notice that it is constructed like any other XML document.

Processing a Transformation

A transformation can take place in one of three locations:

- On the server
- On the client (for example, your web browser)
- With a standalone program

Need for XSL

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Style sheet Language. An XSL document specifies how a browser should render an XML document.

Following are the main parts of XSL –

- XSLT – used to transform XML documents into various other types of document.
- XPath – used to navigate XML document.
- XSL-FO – used to format XML document.

XSLT

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

How XSLT Works?

An XSLT style sheet is used to define the transformation rules to be applied on the target XML document. XSLT style sheet is written in XML format. XSLT Processor takes the XSLT style sheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.

XSLT <template> Element

<xsl:template> defines a way to reuse templates in order to generate the desired output for nodes of a particular type/context.

```
<xsl:template
  name = Qname
  match = Pattern
  priority = number
  mode = QName >
</xsl:template>
```

SN	Name & Description
1	Name : Name of the element on which template is to be applied.
2	Match : Pattern which signifies the element(s) on which template is to be applied.
3	Priority : Priority number of a template. Matching template with low priority is not considered in from in front of high priority template.
4	Mode: Allows element to be processed multiple times to produce a different result each time.

XSLT <value-of>

<xsl:value-of> tag puts the value of the selected node as per XPath expression, as text.

```
<xsl:value-of
  select = Expression
  disable-output-escaping = "yes" | "no" >
</xsl:value-of>
```

Attributes

S.N.	Name & Description
1	Select: XPath Expression to be evaluated in current context.
2	disable-output escaping : Default-"no". If "yes", output text will not escape xml characters from text.

XSLT <for-each>

<xsl:for-each> tag applies a template repeatedly for each node.

```
<xsl:for-each
   select = Expression >
</xsl:for-each>
```

Attributes

SN	Name & Description
1	Select : XPath Expression to be evaluated in the current context to determine the set of nodes to be iterated.

XSLT <sort>

<xsl:sort> tag specifies a sort criteria on the nodes.

```
<xsl:sort
   select = string-expression
   lang = { nmtoken }
   data-type = { "text" | "number" | QName }
   order = { "ascending" | "descending" }
   case-order = { "upper-first" | "lower-first" } >
</xsl:sort>
```

Attributes

S.N.	Name & Description
1	Select : Sorting key of the node.
2	Lang : Language alphabet used to determine the sort order.
3	Data-type :Data type of the text.
4	Order : Sorting order. Default is "ascending".
5	Case-order :Sorting order of string by capitalization. Default is "upper-first".

<xsl:if> Element

<xsl:if> tag specifies a conditional test against the content of nodes.

```
<xsl:if
 test = boolean-expression >
</xsl:if>
```

Example: <xsl:if test = "marks > 90">

<xsl:choose> Element

<xsl:choose> tag specifies a multiple conditional tests against the content of nodes in conjunction with the <xsl:otherwise> and <xsl:when> elements.

```
<xsl:choose>
        <xsl:when test = "marks > 90">
        High
         </xsl:when>

        <xsl:when test = "marks > 85">
        Medium
        </xsl:when>
        <xsl:otherwise>
        Low
        </xsl:otherwise>
        </xsl:choose>
```

Sample Program and Describe with XSLT:

XSL File:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
 <xsl:template match="/">
 <html>
 <body>
 <h1><xsl:value-of select="books/heading" /></h1>
 <table border="1">
  <tr>
    <th>Title</th>
    <th>Author</th>
    <th>Publisher</th>
    <th>Edition</th>
    <th>Price</th>
  </tr>
  <xsl:for-each select="books/book">
    <tr>
      <td><xsl:value-of select="title" /></td>
      <td><xsl:value-of select="author" /></td>
      <td><xsl:value-of select="publisher" /></td>
      <td><xsl:value-of select="edition" /></td>
      <td><xsl:value-of select="price" /></td>
    </tr>
  </xsl:for-each>
</table>
</body>
</html>
 </xsl:template>
</xsl:stylesheet>
```

XML File:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="bookstyle1.xsl"?>
<books>
      <heading>Welcome To GeeksforGeeks </heading>
      <book>
            <title>Web Programming</title>
            <author>Chrisbates</author>
            <publisher>Wiley</publisher>
            <edition>3</edition>
            <price> 300</price>
      </book>
      <book>
            <title>Internet world-wide-web</title>
            <author>Ditel</author>
            <publisher> Pearson</publisher>
            <edition>3</edition>
            <price>400</price>
      </book>
      <book>
            <title>Computer Networks</title>
            <author> Foruouzan</author>
            <publisher>Mc Graw Hill</publisher>
            <edition> 5</edition>
            <price> 700</price>
      </book>
      <book>
            <title> DBMS Concepts</title>
            <author>Navath</author>
            <publisher> Oxford</publisher>
            <edition>5</edition>
            <price>600</price>
      </book>
      <book>
            <title>Linux Programming</title>
            <author>Subhitab Das</author>
            <publisher>Oxford</publisher>
            <edition>8</edition>
            <price>300</price>
      </book>
</books>
```

XQUERY

XQuery is a functional query language used to retrieve information stored in XML format. It is same as for XML what SQL is for databases. It was designed to query XML data. XQuery is built on XPath expressions. It is a W3C recommendation which is supported by all major databases.

The as it is definition of XQuery given by its official documentation is as follows:

It is a standardized language for combining documents, databases, Web pages and almost anything else. It is very widely implemented. It is powerful and easy to learn. XQuery is replacing proprietary middleware languages and Web Application development languages. XQuery is replacing complex Java or C++ programs with a few lines of code. XQuery is simpler to work with and easier to maintain than many other alternatives.

Use of XQuery

XQuery is a functional language which is responsible for finding and extracting elements and attributes from XML documents.

It can be used for following things:

- To extract information to use in a web service.
- To generates summary reports.
- To transform XML data to XHTML.
- Search Web documents for relevant information.

XQuery Features

There are many features of XQuery query language. A list of top features are given below:

- XQuery is a functional language. It is used to retrieve and query XML based data.
- XQuery is expression-oriented programming language with a simple type system.
- XQuery is analogous to SQL. For example: As SQL is query language for databases, same as XQuery is query language for XML.
- XQuery is XPath based and uses XPath expressions to navigate through XML documents.
- XQuery is a W3C standard and universally supported by all major databases.

Advantages of XQuery

- XQuery can be used to retrieve both hierarchal and tabular data.
- XQuery can also be used to query tree and graphical structures.
- XQuery can used to build web pages.
- XQuery can be used to query web pages.
- XQuery is best for XML-based databases and object-based databases. Object databases are much more flexible and powerful than purely tabular databases.
- XQuery can be used to transform XML documents into XHTML documents.

XQuery Example

Course XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<courses>
   <course category="JAVA">
      <title lang="en">Learn Java in 3 Months.</title>
      <trainer>Sonoo Jaiswal</trainer>
      <year>2008</year>
      <fees>10000.00</fees>
   </course>
    <course category="Dot Net">
      <title lang="en">Learn Dot Net in 3 Months.</title>
      <trainer>Vicky Kaushal</trainer>
      <year>2008</year>
      <fees>10000.00</fees>
   </course>
    <course category="C">
      <title lang="en">Learn C in 2 Months.</title>
      <trainer>Ramesh Kumar</trainer>
      <year>2014</year>
      <fees>3000.00</fees>
   </course>
    <course category="XML">
      <title lang="en">Learn XML in 2 Months.</title>
      <trainer>Ajeet Kumar</trainer>
      <year>2015</year>
      <fees>4000.00</fees>
   </course>
</courses>
```

Xquery file courses.xqy

```
for $x in doc("courses.xml")/courses/course
where $x/fees>5000
return $x/title
```

XPATH

XPath is a language that describes a way to locate and process items in Extensible Markup Language (XML) documents by using an addressing syntax based on a path through the document's logical structure or hierarchy. This makes writing programming expressions easier than if each expression had to understand typical XML markup and its sequence in a document. XPath also allows the programmer to deal with the document at a higher level of abstraction. XPath is a language that is used by and specified as part of both the Extensible Stylesheet Language Transformations (XSLT) and XPointer (SML Pointer Language). It uses the information abstraction defined in the XML Information Set (Infoset). Since XPath does not use XML syntax itself, it could be used in contexts other than those of XML.

- XPath stands for XML Path Language
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions
- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

SAX

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

SAX is an *event-driven* interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser.

DOM

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

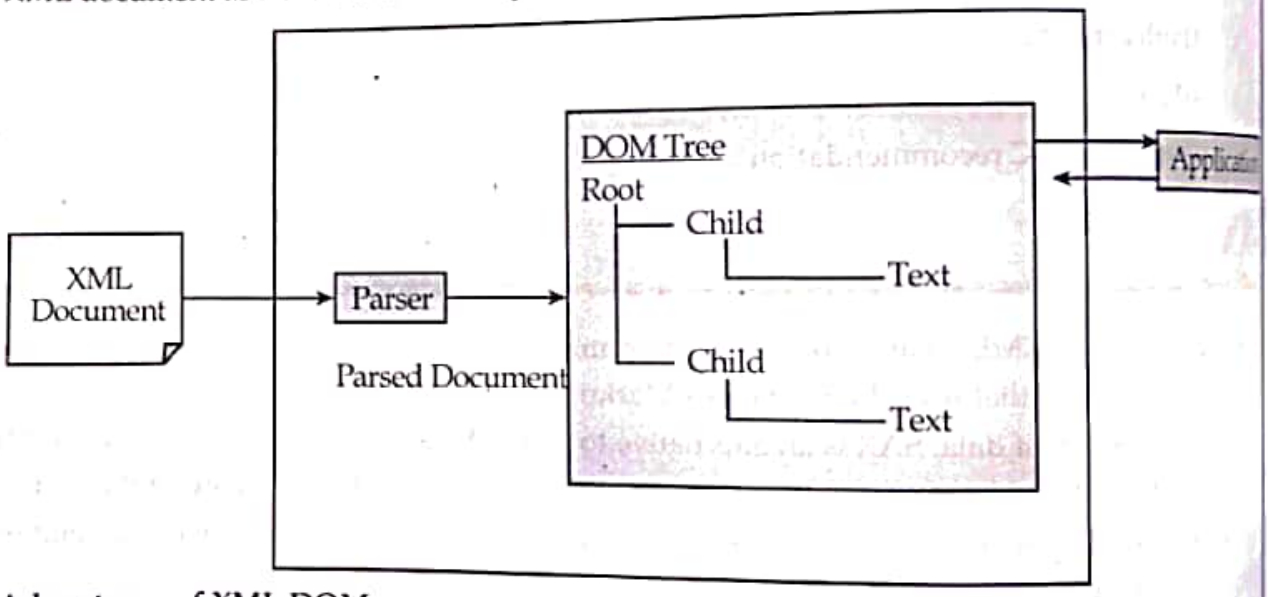
With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML

document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The XML DOM is a standard object model for XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a standard programming interface for describing those nodes and the relationships between them.

As XML DOM also provides an API that allows a developer to add, edit, move or remove nodes at any point on the tree in order to create an application.

Following is the diagram for the DOM structure. The diagram depicts that parser evaluates an XML document as a DOM structure by traversing through each node.



Advantages of XML DOM

The following are the advantages of XML DOM.

- XML DOM is language and platform independent.
- XML DOM is **traversable** - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is **modifiable** - It is dynamic in nature providing the developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages of XML DOM

- It consumes more memory (if the XML structure is large) as program written on remains in memory all the time until and unless removed explicitly.
- Due to the extensive usage of memory, its operational speed, compared to SAX is slower.

XML DOM - Model

A DOM document is a collection of *nodes* or pieces of information, organized in a hierarchy. Some types of *nodes* may have *child* nodes of various types and others are leaf nodes that cannot have anything under them in the document structure. Following is a list of the node types, with a list of node types that they may have as children –

- **Document**: Element (maximum of one), Processing Instruction, Comment, Document Type (maximum of one)
- **Document Fragment**: Element, Processing Instruction, Comment, Text, CDATA Section, Entity Reference
- **Entity Reference**: Element, Processing Instruction, Comment, Text, CDATA Section, Entity Reference
- **Element**: Element, Text, Comment, Processing Instruction, CDATA Section, Entity Reference
- **Attr**: Text, Entity Reference
- **Processing Instruction**: No children
- **Comment**: No children
- **Text**: No children
- **CDATA Section**: No children
- **Entity**: Element, Processing Instruction, Comment, Text, CDATA Section, Entity Reference
- **Notation**: No children

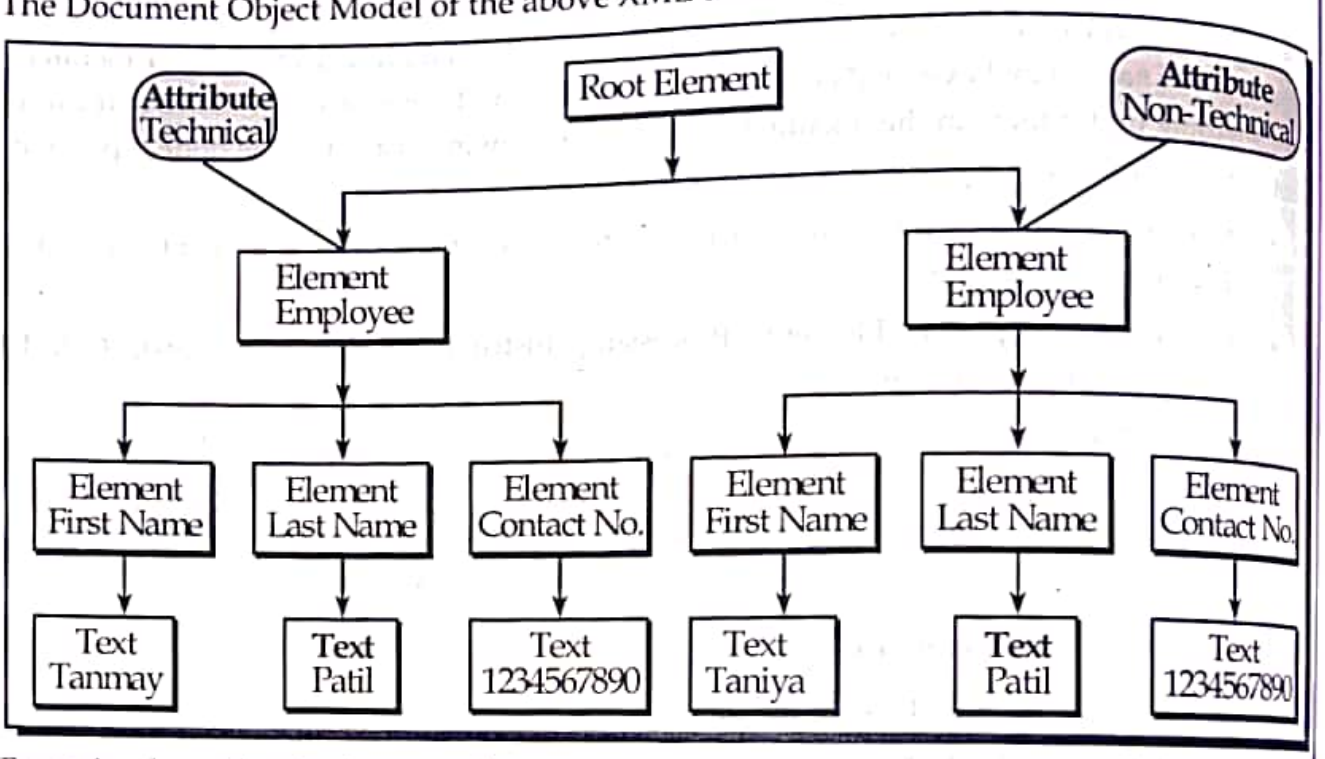
Example

Consider the DOM representation of the following XML document **node.xml**.

```
<?xml version = "1.0"?>
<Company>
  <Employee category = "technical">
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
    <ContactNo>1234567890</ContactNo>
  </Employee>

  <Employee category = "non-technical">
    <FirstName>Taniya</FirstName>
    <LastName>Mishra</LastName>
    <ContactNo>1234667898</ContactNo>
  </Employee>
</Company>
```

The Document Object Model of the above XML document would be as follows –



From the above flowchart, we can infer –

- *Node* object can have only one *parent node* object. This occupies the position above all the nodes. Here it is *Company*.
- The *parent node* can have multiple nodes called the *child* nodes. These *child* nodes can have additional nodes called the *attribute* nodes. In the above example, we have two attribute nodes *Technical* and *Non-technical*. The *attribute* node is not actually a child of the element node, but is still associated with it.
- These *child* nodes in turn can have multiple child nodes. The text within the nodes is called the *text* node.
- The node objects at the same level are called as siblings.
- The DOM identifies –
  - the objects to represent the interface and manipulate the document.
  - the relationship among the objects and interfaces.

Creating XML Parser

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

However, before an XML document can be accessed, it must be loaded into an XML DOM object.

All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

Parsing a Text String

This example parses a text string into an XML DOM object, and extracts the info from it with JavaScript:

Example

```
<html>
<body>
<p id="demo"></p>
<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

Some Practical Problems With Solutions

1. XML Code: note with root tag note
   - Note element with child element to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>

<note>

<to>Ramesh</to>

<from>Krishna</from>

<subject>This is my first xml</subject>

<body>This is my first message for xml</body>

</note>
```

2. XML Code: note with attribute, without child tag
   - Note element with attribute to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>

<note to="Ramesh" from="Krishna" subject="This is my first XML" body="This is my first
    message for xml" >
</note>
```

3. **XML Code: notes data with multiple note**

- Notes with child multiple note
- Note element with child element to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>
<notes>
 <note>
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <body>This is my first message for xml</body>
 </note>
 <note>
 <to>Sita</to>
 <from>Gita</from>
 <subject>This is my second xml</subject>
 <body>This is my second message for xml</body>
 </note>
</notes>
```

4. **XML Code: Using Element & Attribute together**

- Notes with child multiple note
- Note element with child element to, from, subject, body
- Note element have attribute id for each note

```
<?xml version="1.0" encoding="UTF-8"?>
<notes>
 <note id="1">
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <body>This is my first message for xml</body>
 </note>
 <note id="2">
 <to>Sita</to>
 <from>Gita</from>
 <subject>This is my second xml</subject>
 <body>This is my second message for xml</body>
 </note>
 </notes>
```

5 **XML Code: with internal DTD for single note**

- Note element with child element to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
 <!ELEMENT note (to,from,subject, body ) >
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <body >This is my first message for xml</body>
</note>
```

6 **XML Code: With external DTD with multiple note**

- Notes with child multiple note
- Note element with child element to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE notes SYSTEM "5.notes.dtd">
<notes>
 <note>
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <message>This is my first message for xml</message>
 </note>
 <note>
 <to>Sita</to>
 <from>Gita</from>
 <subject>This is my first xml</subject>
 <message>This is my second message for xml</message>
 </note>
</notes>
```

- <! DOCTYPE notes SYSTEM "5.notes.dtd"> is used to link DTD file to xml file.

DTD File

```
<!ELEMENT notes (note+)>
<!ELEMENT note (to,from,subject,message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

- Note with "+" xml file contain one or more note.

7. XML Code: With external DTD with multiple note with other parameters

- Notes with child multiple note
- Note element with child element to, from, subject, body or message
- Note also have type attribute which is required
- Single note can be sent to multiple recipient

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE notes SYSTEM "6.notes.dtd">
<notes>
 <note type="sms">
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <message>This is my first message for xml</message>
 </note>
 <note type="email">
 <to>Sita</to>
 <to>Ramesh</to>
 <from>Gita</from>
 <subject>This is my second xml</subject>
 <body>This is my second message for xml</body>
 </note>
</notes>
```

DTD file:

```
<!ELEMENT notes (note+)>

<!ELEMENT note (to+,from,subject, ( message | body )?)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT subject (#PCDATA)>

<!ELEMENT body (#PCDATA)>

<!ELEMENT message (#PCDATA)>

<!ATTLIST note type CDATA #REQUIRED>
```

- Note can be sent to multiple user at single time(to+)
- Optional (?) element (message | body)
- Note have type attribute which is required

XML Code: list of books with attribute

- Library has multiple books with single book
- Single book have isbn attribute
- Book also have name, author, page and price element
- Books can be written by multiple author with author_name, email
- Price have currency attribute with npr & usd choice with default npr value

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE books SYSTEM "7.books.dtd">
<books>
 <book isbn="548745">
  <name>Web Tech</name>
  <author>
   <author_name>ABCD</author_name>
   <email>info@abcd.com</email>
  </author>
  <page>200</page>
  <price currency="npr">450</price>
 </book>
 <book isbn="548745">
  <name>HTML CSS &amp;amp; Wev TecTech</name>
  <author>
   <author_name>ABCD</author_name>
   <email>info@abcd.com</email>
  </author>
  <author>
   <author_name>Ramesh</author_name>
   <email>info@ramesh.com</email>
  </author>
  <page>900</page>
  <price currency="npr">1450</price>
 </book>
 <book isbn="327845">
  <name>HTML CSS &amp;amp; Wev TecTech1</name>
  <author>
   <author_name>Ramesha</author_name>
   <email>info@ramesha.com</email>
  </author>
  <page>1020</page>
  <price currency="usd">15</price>
 </book>
</books>
```

DTD File:

```
<!ELEMENT books (book+)>
<!ELEMENT book (name,author+,page,price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (author_name,email)>
<!ELEMENT author_name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT page (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ATTLIST price currency (npr | usd) "npr">
```

9. XML Code: Validate note using XML Schema for single note

- Note element with child element to, from, subject, body

```
<?xml version="1.0" encoding="UTF-8"?>
<note
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="9.note_with_schema.xsd">
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <body >This is my first message for xml</body>
</note>
```

Schema Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="note">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="subject" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

- To link schema file we need following code into xml: where note is root tag

```
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="9.note_with_schema.xsd">
```

XML Code: Validate notes using XML Schema for multiple note

```
Notes with child multiple note
Note element with child element to, from, subject, body
<?xml version="1.0" encoding="UTF-8"?>
<notes
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="9.multiplenote_with_schema.xsd">
<note>
<to>Ramesh</to>
<from>Krishna</from>
<subject>This is my first xml</subject>
<body >This is my first message for xml</body>
</note>
<note>
<to>Gita</to>
<from>Rita</from>
<subject>This is my second xml</subject>
<body >This is my second message for xml</body>
</note>
</notes>
```

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="notes" >
  <xs:complexType>
   <xs:sequence>
    <xs:element name="note" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="to" type="xs:string"/>
       <xs:element name="from" type="xs:string"/>
       <xs:element name="subject" type="xs:string"/>
       <xs:element name="body" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

11. XML Code: Validate notes using XML Schema for multiple note and ID attribute

- Notes with child multiple note
- Note element with child element to, from, subject, body
- Note also have ID attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<notes
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="10.notewithattribute.xsd">
<note ID="1">
 <to>Ramesh</to>
 <from>Krishna</from>
 <subject>This is my first xml</subject>
 <body >This is my first message for xml</body>
</note>
<note ID="2">
 <to>Gita</to>
 <from>Rita</from>
 <subject>This is my second xml</subject>
 <body >This is my second message for xml</body>
</note>
</notes>
```

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="notes" >
   <xs:complexType>
    <xs:sequence>
     <xs:element name="note" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="subject" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
       </xs:sequence>
       <xs:attribute name="ID" type="xs:integer" use="optional"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

12. **XML Schema: Phone Entries with multiple entries and attribute, option element**

- A contact book has phoneNumbers as root element
- phoneNumbers has phone entries
- entries has multiple phone entry
- entry contain name, phone, city
- name have first, middle, last name
- city has option element

```
<?xml version="1.0" encoding="UTF-8"?>
<phoneNumbers
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="11.phone.xsd">
<title>Test</title>
<entries>
 <entry>
 <name gender="male">
  <first>test</first>
  <middle>dsfds</middle>
  <last>sdfdsf</last>
 </name>
 <phone>987545454</phone>
 <city>Chitwan,Kathmandu</city>
 </entry>
 <entry>
 <name gender="female">
  <first>test</first>
  <middle>dsfds</middle>
  <last>sdfdsf</last>
 </name>
 <phone>987545454</phone>
 </entry>
</entries>
</phoneNumbers>
```

Xml Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="phoneNumbers">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="title" type="xs:string"  maxOccurs="unbounded"></xs:element>
    <xs:element name="entries">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="entry" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
       <xs:sequence>
       <xs:element name="name" >
       <xs:complexType>
        <xs:sequence>
         <xs:element name="first" type="xs:string" ></xs:element>
         <xs:element name="middle" type="xs:string" minOccurs="0"
      maxOccurs="1"></xs:element>
          <xs:element name="last" type="xs:string"></xs:element>
        </xs:sequence>
        <xs:attribute name="gender" type="xs:string"></xs:attribute>
       </xs:complexType>
       </xs:element>
       <xs:element name="phone" type="xs:string"></xs:element>
       <xs:element name="city" type="xs:string"  minOccurs="0"
      maxOccurs="1"></xs:element>
       </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     </xs:complexType>
     </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

13. **XML Schema: with various restriction and pattern**

- City has list of city name :Kathmandu, Pokhara, Chitwan, Butwal, Biratnagar
- Gender has one of two value (male, female)
- Area code has 3 digit value
- Phone has xxx-xxxx digit format

```
<?xml version="1.0" standalone="no" ?>
<phoneNumbers
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="12.restriction.xsd">
<title>Phone Numbers</title>
<entries>
 <entry>
  <name>
   <first>Ram</first>
   <last>Thapa</last>
  </name>
  <phone areaCode="319">335-0055</phone>
  <city>Pokhara</city>
 </entry>
<entry>
 <name gender="male">
  <first>Hari</first>
  <last>Sharma</last>
 </name>
 <phone>354-9876</phone>
 <city>Kathmandu</city>
</entry>
<entry>
 <name gender="female">
  <first>Ramesh</first>
  <middle>Prasad</middle>
  <last>Shah</last>
 </name>
 <phone areaCode="319">335-4582</phone>
 <city>Chitwan</city>
</entry>
<entry>
 <name gender="female">
  <first>Ritu</first>
  <last>Shrestha</last>
 </name>
 <phone>337-5967</phone>
</entry>
</entries>
</phoneNumbers>
```

Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="cityType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Kathmandu"/>
            <xs:enumeration value="Pokhara"/>
            <xs:enumeration value="Chitwan"/>
            <xs:enumeration value="Butwal"/>
            <xs:enumeration value="Biratnagar"/>
        </xs:restriction>
 </xs:simpleType>
 <xs:complexType name="nameType">
        <xs:sequence>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string" minOccurs="0"/>
        <xs:element name="last" type="xs:string"/>
        </xs:sequence>
 <xs:attribute name="gender">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                    <xs:pattern value="male|female"/>
            </xs:restriction>
        </xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:simpleType name="phoneType">
        <xs:restriction base="xs:string">
            <xs:pattern value="\d{3}-\d{4}"/>
        </xs:restriction>
</xs:simpleType>
<xs:complexType name="entryType">
       <xs:sequence>
            <xs:element name="name" type="nameType"/>
            <xs:element name="phone">
                    <xs:complexType>
                            <xs:simpleContent>
                                    <xs:extension base="phoneType">
                                            <xs:attribute name="areaCode">
                                                    <xs:simpleType>
                                                            <xs:restriction base="xs:string">
```

```
                                                    <xs:pattern
value="\d{3}"/>
                                                    </xs:restriction>
                                                </xs:simpleType>
                                            </xs:attribute>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="city" type="cityType" minOccurs="0"/>
                        </xs:sequence>
                    </xs:complexType>
                    <xs:element name="phoneNumbers">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="title" type="xs:string"/>
                                <xs:element name="entries">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="entry" type="entryType"
minOccurs="0" maxOccurs="unbounded"/>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:schema>
```

Discussion Exercise

1. What is XML? Describe use of XML in web technology.
2. What is XML element and attribute, Which one is used mostly and why?
3. Write rules of writing XML.
4. What is a namespace? describe use of namespace into xml with example.
5. Difference between well formed and valid XML document.
6. What are different technique to validate XML document?

7. What is a DTD? Describe different types of DTD with example.
8. What are different occurrence indicator used into DTD?
9. What is XML schema? Describe advantage of schema with dtd.
10. Difference between simple types and complex types of element.
11. How to define attribute and element into xml schema.
12. How to define default value, fixed value into xml attribute.
13. Describe different occurrence indicator with example.
14. What is facets? Describe different restriction used into xml schema.
15. What is XSL ? Describe XSLT with example.
16. Describe the following topics:
    - XPath
    - Xquery
    - SAX
    - DOM
    - XML Parser

❑❑❑

Chapter 6

SERVER SIDE SCRIPTING USING PHP

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- PHP Syntax, Variables, Data Types, Strings, Constants, Operators, Control structure, Functions, Array, Creating Class and Objects, PHP Forms, Accessing Form Elements, Form Validation, Events, Cookies and Sessions, Working with PHP and MySQL, Connecting to Database, Creating, Selecting, Deleting, Updating Records in a table, Inserting Multiple Data, Introduction to Codelgniter, Laravel, Wordpress etc.

Basics of PHP

PHP: Hypertext Preprocessor (or simply PHP) is a general purpose programming language originally designed for web development. PHP originally stood for Personal Home Page but now it stands for the recursive acronym PHP i.e. Hypertext Preprocessor

PHP is a programming language used mostly for building websites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computer

PHP is a programming language. Something in the web server computer reads your PHP programs, which are instructions written in this programming language and figures out what to do. The PHP engine follows the instruction.

PHP is called sever side scripting language because it runs on a web server. A language such as JavaScript can be used as a client-side language because, embedded in a web browser, it can cause that browser, while running on your PC to do something such as pop up a new window

Advantages of PHP

The following are the advantages of PHP:

PHP is Open Source

We don't have to pay anyone to use PHP. Whether you run the PHP engine on a beat-up 10-year-old PC in your basement or in a room full of million-dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

PHP is Cross Platform

PHP can be used with a web server computer that runs Windows, Mac OS , Linux and many other version of Unix.

PHP is Widely used

PHP is used on different websites from countless tiny personal homepage to giant's websites

PHP hides its complexity

Powerful ecommerce websites can be build using PHP as well as simple sites. Simpler and complex features can be used simultaneously which doesn't get in the way with each other's

PHP is Built for Web Programming

Unlike other programming languages, PHP was created for generating web pages.

Syntax

Unlike other programming language do have its own syntax PHP also has its own syntax
Start Tag: The less than sign(<) followed by ? symbol and the word 'php' i.e : <?php any thing written inside this tag is considered as PHP code
End Tag: The ? symbol followed by greater than sign(>) i.e ?>

So the syntax for PHP is:

```
<?php
//statements or blocks
?>
```

PHP can support multiple start and end tags.Each start tag must be closed by its own end tags.

Comments in PHP

1. Single Line comment

   The two backslash (//) or the hash (#) is used to post a single line comment in PHP

2. Multiline comments

   For a multiline comment, start the comment with/* and end it with*/. Everything between the /*and */ is treated as a comment by the PHP engine. Multiline comments are useful for temporarily turning off a small block of code.

Basic Rules of PHP Programs

1. Start and End Tags
2. Whitespace and Case-Sensitivity
3. Comments

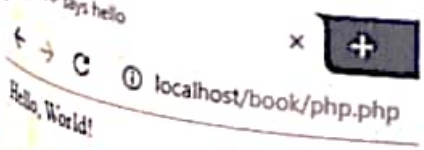
Printing statement in PHP

To print the statement in PHP we use **print** or **echo**

Example 1: Printing Hello world

```
<html>
<head>
<title>PHP says hello</title>
</head>
<body>
<?php print "Hello, World!";
?>
</body>
</html>
```

Output:



Variables

Variables hold the data that your program manipulates while it runs, such as user information that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by a (dollar sign) $ followed by the variable's name. To assign a value to a variable, use an equals sign (=). This is known as the assignment operator.

Variable names may only include:

- Uppercase or lowercase Basic Latin letters (A-Z and a-z)
- Digits (0-9)
- Underscore (_)
- Any non-Basic Latin character (such as ç), if you're using a character encoding such as UTF-8 for your program file
- Additionally, the first character of a variable name is not allowed to be a digit. Table 2-2lists some allowable variable names.

Following are the list of some allowed variable names:

$size
$drinkSize
$SUPER_BIG_DRINK
$_d_r_i_n_k_y
$drink4you2
$напиток

The following are the list of some disallowed variable names and reason

Variable name	Reason
$2hot4u	Begins with a number
$drink-size	Unacceptable character: -
$drinkmaster@example.com	Unacceptable characters: @ and .
$drink!lots	Unacceptable character: !
$drink+dinner	Unacceptable character: +

Variable names are case-sensitive. This means that variables named $dinner, $Dinner, and $DINNER are separate and distinct, with no more in common than if they were named $breakfast, $lunch, and $supper. In practice, you should avoid using variable names that differ only by letter case. They make programs difficult to read and debug.

Example 2: Defining the variables

```
<?php
$name;
$roll;
?>
```

Text

When they're used in computer programs, pieces of text are called strings. This is because they consist of individual items, strung together. Strings can contain letters, numbers, punctuation, spaces, tabs, or any other characters. A string can even contain the contents of a binary file, such as an image or a sound. The only limit to the length of a string in a PHP program is the amount of memory your computer has.

Defining Text Strings

There are a few ways to indicate a string in a PHP program. The simplest is to surround the string with single quotes or double quotes. The single quotes aren't part of the string. They are delimiters, which tell the PHP engine where the start and end of the string is. If you want to include a single quote inside a string surrounded with single quotes, put a backslash (\) before the single quote inside the string.The backslash tells the PHP engine to treat the following character as a literal single quote instead of the single quote that means "end of string." This is called escaping, and the backslash is called the escape character. An escape character tells the system to do something special with the character that comes after it. Inside a single-quoted string, a single quote usually means "end of string." Preceding the single quote with a backslash changes its meaning to a literal single quote character. To concatenate strings and variables we use "." Known concatenating operator

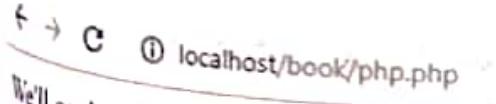
Example 3: Defining Strings

```
<?php
print('Hello, World!')
?>
```

Example 4: Defining strings with backslash

```
<?php
print ('We\'ll each have a bowl of soup.');
?>
```

Output:



We'll each have a bowl of soup.

The biggest difference between single-quoted and double-quoted strings is that when you include variable names inside a double-quoted string, the value of the variable is substituted into the string, which doesn't happen with single-quoted strings. For example, if the variable $user holds the value Bill, then 'Hi $user' is just that: Hi $user. However, "Hi $user" is Hi Bill. "Variables" gets into this in more detail.

Validating Strings

Validation is the process of checking that input coming from an external source conforms to an expected format or meaning.

The **trim()** function removes whitespace from the beginning and end of a string. **strlen()**,which tells the length of a string, **rtrim()** removes the whitespace from the right side of the string whereas **ltrim()** removes the whitespace from the left side of the string

Syntax:

trim(variable name or strings)

ltrim(variable name or strings)

rtrim(variable name or strings)

strlen(variable name or strings)

To compare strings without paying attention to case, use **strcasecmp()**. It compares two strings while ignoring differences in capitalization. If the two strings you provide to **strcasecmp()** are the same independent of any differences between upper- and lowercase letters, it returns 0. The **strcmp()** function is used to compare the strings without ignoring the differences in capitalization.

Syntax:

strcasecmp(string1, string2)

strcmp(string1, string2)

The **ucwords()** function uppercases the first letter of each word in a string. This is useful when combined with **strtolower()** to produce nicely capitalized names when they are provided to you in all uppercase. The **strtoupper()** converts the string into uppercase. The **substr()** function, you can extract just part of a string.The three arguments to substr() are the string to work with, the starting position of the substring to extract, and the number of bytes to extract. The beginning of the string is position 0, not 1. When you give substr() a negative number for a start position, it counts back from the end of the string to figure out where to start. A start position of -4 means "start four bytes from the end." The **str_replace()** function changes parts of a string. It looks for a substring andreplaces the substring with a new string

Syntax:

ucwords(string)

strtolower(string)

strtoupper(string)

substr(string,start,length)

str_replace(find,replace,string)

Data Types

Data Types defines the type of data a variable can store. PHP allows eight different types of data types. All of them are discussed below. The first five are called simple data types and the last three are compound data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- NULL
- Array
- Object
- Resource

Constants

PHP constants are name or identifier that can't be changed during the execution of the script. PHP constants can be defined by 2 ways:

1. Using define() function
2. Using const keyword

PHP constants follow the same PHP variable rules. For example, it can be started with letter or underscore only. Conventionally, PHP constants should be defined in uppercase letters.

PHP constant: define()

It is case sensitive by default.

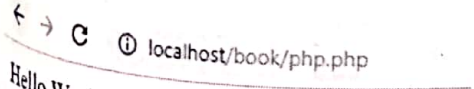
Syntax:

define(name,value)

Example: define example

```
<?php
define("MESSAGE","Hello World!");
echo MESSAGE;
?>
```

Output:



Hello World!

PHP constant: const keyword

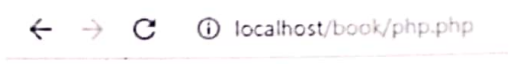
The const keyword defines constants at compile time. It is a language construct not a function.

1. It is bit faster than define().
2. It is always case sensitive.

Example : Const example

```
<?php
const MESSAGE="Hello const by PHP";
echo MESSAGE;
?>
```

Output:



Hello const by PHP

PHP Operator

PHP Operator is a symbol i.e. used to perform operations on operands. PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Logical Operators
- String Operators
- Incrementing/Decrementing Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators
- Assignment Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- Unary Operators: works on single operands such as ++, -- etc.
- Binary Operators: works on two operands such as binary +, -, *, / etc.
- Ternary Operators: works on three operands such as "?:".

Control Structure

PHP If Else

PHP if else statement is used to test condition. There are various ways to use if statement in PHP.

- if
- if-else
- if-else-if
- nested if

PHP if statement

PHP if statement is executed if condition is true.

Syntax:

```
if(condition){
//code to be executed
}
```

PHP If-else Statement

PHP if-else statement is executed whether condition is true or false.

Syntax:

```
if(condition){
//code to be executed if true
}
else{
//code to be executed if false
}
```

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement

Syntax

```
switch(expression){
case value1:
//code to be executed
break;
case value2:
//code to be executed
break;
...
default:
code to be executed if all cases are not matched;
}
```

PHP For loop

PHP for loop can be used to traverse set of code for the specified number of times. It should be used if number of iteration is known otherwise use while loop.

Syntax

```
for(initialization; condition; increment/decrement){
//code to be executed
}
```

Example 8: For Loop

```
<?php
for($n=1;$n<=10;$n++){
echo "$n<br/>";
}
?>
```

Output

1
2
3
4
5
6
7
8
9
10

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop.In case of inner or nested for loop, nested for loop is executed fully for one outer for loop. If outer for loop is to be executed for 3 times and inner for loop for 3 times, inner for loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example: Nested For loop

```
<?php
for($i=1;$i<=3;$i++){
for($j=1;$j<=3;$j++){
echo "$i   $j<br/>";
}
}
?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP For Each Loop

PHP for each loop is used to traverse array elements. We will be covering more in Array

Syntax:

```
foreach($array_name as $key => $data )
```

PHP While Loop

PHP while loop can be used to traverse set of code like for loop. It should be used if number of iteration is not known.

Syntax:

```
while(condition){
//code to be executed
}
```

PHP do while loop

PHP do while loop can be used to traverse set of code like php while loop. The PHP do-while loop is guaranteed to run at least once. It executes the code at least one time always because condition is checked after executing the code.

Syntax:

```
do{
//code to be executed
}while(condition);
```

PHP Functions

When you're writing computer programs, laziness is a virtue. Reusing code, you've already written makes it easier to do as little work as possible. Functions are the key to code reuse. A function is a named set of statements that you can execute just by invoking the function name instead of retyping the statements. This saves time and prevents errors. Plus, functions make it easier to use code that other people have written.

Advantage of PHP Functions

- **Code Reusability:** PHP functions are defined only once and can be invoked many times, like in other programming languages.
- **Less Code:** It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.
- **Easy to understand:** PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

Creating a Function

While creating a user defined function we need to keep few things in mind:

- Any name ending with an open and closed parenthesis is a function.
- A function name always begins with the keyword function.
- To call a function we just need to write its name followed by the parenthesis
- A function name cannot start with a number. It can start with an alphabet or underscore.
- A function name is not case-sensitive.

Syntax:

```
function function_name(){
 // executable code;
}
```

Calling a Function

Functions can be defined before or after they are called. The PHP engine reads the entire program file and takes care of all the function definitions before it runs any of the commands in the file.

Example: Creating a Function

```
<?php
function sum() {
    echo 'sum of 10 and 20 is: 30';
}
?>
```

Example: Defining functions before or after calling them

```
<?php
function greet()
{
   echo 'Hello';
}
   greet();
   print "Welcome, to the class of PHP ";
   message();
function message()
{
   echo 'Thanks for coming';
}
?>
```

Passing Arguments to Function

The information or variable, within the function's parenthesis, are called parameters. These are used to hold the values executable during runtime. A user is free to take in as many parameters as he wants, separated with a comma (,) operator. These parameters are used to accept inputs during runtime. While passing the values like during a function call, they are called arguments. An argument is a value passed to a function and a parameter is used to hold those arguments. In common term, both parameter and argument mean the same. We need to keep in mind that for every parameter, we need to pass its corresponding argument.

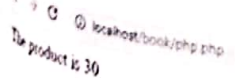
Syntax:

```
function function_name($first_parameter, $second_parameter) {
  executable code;
}
```

Example: Passing arguments to function

```
<?php
// function along with three parameters
function product($num1,  $num2,  $num3)
{
   $product = $num1 * $num2 * $num3;
   echo "The product is $product";
}
// Caling the function
// Passing three arguments
product (2,  3,  5);
?>
```

Output:



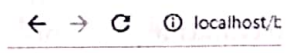
Default values for function parameter

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

Example: Default values for function parameter

```
<?php
function sayHello($name="John"){
echo "Hello $name<br/>";
}
sayHello("Romeo");
sayHello();//passing no value
sayHello("Johny");
?>
```

Output:



Hello Romeo
Hello John
Hello Johny

Returning Values from Functions

Functions can also return values to the part of program from where it is called. The *return* keyword is used to return value back to the part of program, from where it was called. The returning value may be of any type including the arrays and objects. The return statement also marks the end of the function and stops the execution after that and returns the value.

Example: Returning values from Function

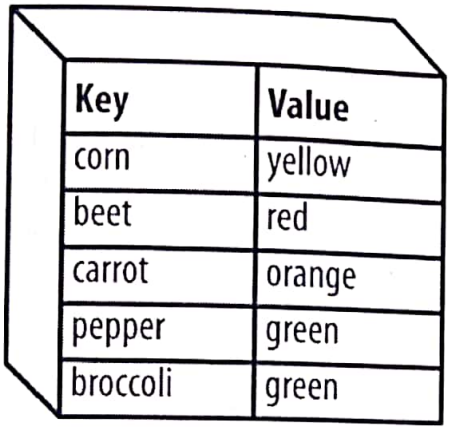
```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

Output

Cube of 3 is: 27

PHP Array

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key. An array is made up of elements. Each element has a key and a value. For example, an array holding information about the colors of vegetables has vegetable names for keys and colors for values, as shown



Key	Value
corn	yellow
beet	red
carrot	orange
pepper	green
broccoli	green

An array can only have one element with a given key. In the vegetable color array, there can't be another element with the key corn even if its value is blue. However, the same value can appear many times in one array. You can have green peppers, green broccoli, and green celery.

Any string or number value can be an array element key, such ascorn, 4, -36, or SaltBaked Squid. Arrays and othernonscalar1 values can't bekeys, but they can be element values. An element value can be anything: astring, a number, true, false, or a non-scalar type such as another array.

Creating an Array

To create an array, use the array() language construct. Specify a comma-delimited list of key/value pairs, with the key and the value separated by =>

Example: Creating an array

```
<?php
$fruit_name = array("Apple","Banana");
?>
```

A shortcut for the array() language construct is a pair of square brackets (called the short array syntax)

Example: Using short array syntax

```
<?php
$fruit_name = ["Apple","Banana"];
?>
```

Types of Array

There are basically three types of arrays in PHP:

- **Indexed or Numeric Arrays**: An array with a numeric index where values are stored linearly.
- **Associative Arrays**: An array with a string index where instead of linear storage, each value can be assigned a specific key.
- **Multidimensional Arrays**: An array which contains single or multiple array within it and can be accessed via multiple indices.

Numeric Array

PHP provides some shortcuts for working with arrays that have only numbers as keys. If you create an array with [] or array() by specifying only a list of values instead of key/value pairs, the PHP engine automatically assigns a numeric key to each value. The keys start at 0 and increase by one for each element

Example: Numeric Array

```
<?php
$season=array("summer","winter","spring","autumn");
?>
```

To print the array we use print_r() function.

Syntax: print_r(array_name)

Example: Printing array

```
<?php
$season=array("summer","winter","spring","autumn");
print_r($season);
?>
```

Output:

Array ([0] => summer [1] => winter [2] => spring [3] => autumn)

Associative Array

These type of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

Example: Associative Array

```
<?php
$associative_array = array(
    "Zack"=>"Zara",
    "Anthony"=>"Any",
    "Ram"=>"Rani",
    "Salim"=>"Sara",
    "Raghav"=>"Ravina"
    );
?>
```

Output:
Array ([Zack] = > Zara [Anthony] = > Any [Ram] = > Rani [Salim] = > Sara [Raghav] => Ravina)

Multidimensional Array

Multi-dimensional arrays are such arrays which stores an another array at each index instead of single element. In other words, we can define multi-dimensional arrays as array of arrays. As the name suggests, every element in this array can be an array and they can also hold other sub-arrays within. Arrays or sub-arrays in multidimensional arrays can be accessed using multiple dimensions.

Syntax:
$array_name = array(array());

Example: Creating multidimensional arrays with array() and []

```
<?php
$meals = array('breakfast'  => ['Walnut Bun','Coffee'],
               'lunch'      => ['Cashew Nuts', 'White Mushrooms'],
               'snack'      => ['Dried Mulberries','Salted Sesame Crab']);
               print_r($meals);
               echo '<br>';

$lunches = [ ['Chicken','Eggplant','Rice'],
             ['Beef','Scallions','Noodles'],
             ['Eggplant','Tofu'] ];
             print_r($lunches);
               echo '<br>';

$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard',
                                     'pepper-salty' => 'prickly ash'));
          print_r($flavors);
          echo '<br>';
?>
```

Output:

```
Array
(
    [breakfast]    =>  Array
       (
            [0]    => Walnut Bun
            [1]    => coffee

       )
    [lunch]=>Array
            [0]    => Walnut Bun
            [1]    => coffee
       )
```

```
        [snack]=>Array
                [0]     => Dried Mulberries
                [1]     => Salted Sesame crab
            )
)
Array
(
        [0] => Array
            (
                [0]     => Chicken
                [1]     => Eggplant
                [2]     => Rice
        [1] => Array
            (
                [0]     => Beef
                [1]     => Scallions
                [2]     => Scallions
        [2] => Array
            (
                [0]     => Eggplant
                [1]     => Tofu
            )
)
Array
(
        [Janapese]  => Array
            (
              [jot] => wasabi
             [Salty] => Soy sauce

            )
        [Chinese] =>Array
            (
                    [hot] = wasabi
                    [pepper-salty] =>prickly ash
        )
)
```

Looping through arrays

To loop array, we use foreach loop;

Syntax

```
<?php
   foreach ($array_name as $key => $value) {
      # code...

   }
   //or
   foreach ($variable as $value) {
      # code...

   }
?>
```

Example: Looping with foreach

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
'dinner' => 'Eggplant with Chili Sauce');
foreach ($meal as $key => $value) {
   echo "Key is ".$key. " and value is " .$value;
}
?>
```

Output:

Key is breakfast and value is Walnut Bun

Key is lunch and value is Cashew Nuts and White Mushrooms

Key is snack and value is Dried Mulberries

Key is dinner and value is Eggplant with Chili Sauce

Some Array Function

The **count()** or **sizeof()** function tells you the number of elements in an array

Syntax:

count(array_name) or sizeof(array_name)

To check for an element with certain key, use **array_key_exists()**. This function returns true if an element with the provided key exists in the provided array.

Syntax:

array_key_exists(key,array_name)

To check for an element with certain key, use **in_array()**. This function returns true if it finds an element with the given value. It is case-sensitive when it compares strings.

Syntax:

in_array(value,array_name)

The **array_search()** function is similar to in_array(), but if it finds an element, it returns the element key instead of true

Syntax:

array_search(value,array_name)

To remove an element from an array, use **unset()**. Removing an element with unset() is different than just setting the element value to 0 or the empty string. When you use unset(), the element is no longer there when you iterate through the array or count the number of elements in the array. Using unset() on an array that represents a store's inventory is like saying that the store no longer carries a product. Setting the element's value to 0 or the empty string says that the item is temporarily out of stock.

Syntax:

unset(array[key])

To convert string from array, use **implode()** function.It makes a string by combining all the values in an array, putting a string delimiter between each value.

Syntax:

implode(separator,array)

Example: Making a string from an array with implode()

```
<?php
$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');
$menu = implode(', ', $dimsum);
echo $menu;
?>
```

Output:

Chicken Bun, Stuffed Duck Web, Turnip Cake

The counterpart to **implode()** is called **explode()**. It breaks a string apart into an array. The delimiter argument to **explode()** is the string it should look for to separate array elements.

Syntax:

explode(separator,string,limit)

Example 22: Turning a string into an array with explode()

```
<?php
    $fish = 'Bass, Carp, Pike, Flounder';
    $fish_list = explode(', ', $fish);
    print_r($fish_list);
?>
```

Output:

```
Array ( [0] => Bass [1] => Carp [2] => Pike [3] => Flounder )
```

Sorting Arrays

There are several ways to sort arrays. Which function to use depends on how you want to sort your array and what kind of array it is. The following are the types of sorting techniques:

- sort()
- asort()
- ksort()
- rsort()
- arsort()
- krsort()

The sort() function sorts an array by its element values in ascending order. It should only be used on numeric arrays, because it resets the keys of the array when it sorts. The rsort() sorts an array element values in descending order

Example: Sorting with sort()

```
<?php
$dinner = array('Sweet Corn and Asparagus',
'Lemon Chicken',
'Braised Bamboo Fungus');
echo "before sorting </br>";
print_r($dinner);
echo "</br>";
sort($dinner);
echo "After sorting </br>";
print_r($dinner);
?>
```

Output:

before sorting
Array [0] => Sweet Corn and Asparagus [1] => Lemon Chicken [2]=>Braised Bamboo Fungus)
After. Sorting
Array ([0] => Braised Bamboo Fungus [1] => Lemon Chicken [2]=> Sweet Corn and Asparagus)

To sort an associative array by element value in ascending order, use **asort()**. This keeps keys together with their values. The **arsort()** sorts array by element value in descending order. The values are sorted in the same way with **asort()** as with **sort()**, but this time, the keys stick around.

Example 24: sorting with asort()

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
'dinner' => 'Eggplant with Chili Sauce');
echo "before sorting </br>";
print_r($meal);
echo "</br>";
asort($meal);
echo "After sorting </br>";
print_r($meal);
?>
```

Output:

before soring

Array ([breakfast] => Walnut Bun [Lunch] > Cashew Nuts and White Mushrooms [snack]=> Dried Mulberries [dinner]=> Eggplant with Chili Sauce)

Array ([lunch] = > Cashew Nuts and White Mushrooms [snack] => Dried Mulberries [dinner] => Eggplant with Chili Sauce [breakfast] = Walnut Bun)

To sort an array by keys in ascending order, use **ksort()**. This keeps key/value pairs together, but orders them by key. The **krsort()** sorts array by keys in descending order

Example Sorting with ksort()

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
'dinner' => 'Eggplant with Chili Sauce');
echo "before sorting </br>";
print_r($meal);
echo "</br>";
ksort($meal);
echo "After sorting </br>";
print_r($meal);
?>
```

Output:

before sorting

Array ([breakfast => Walnut Bun [lunch] => Cashew Nuts and White Mushroom [snack] => Dried Mulberries [dinner] = > Eggplant with Chili Sauce]

After sorting

Array ([breakfast] = > Walnut Bun [dinner] = > Eggplant with Chili Sauce [launch] = > Cashew Nuts and White Mushrooms [snack] = > Dried Mulberries)

Creating Class and Objects in PHP

Like C++ and Java, PHP also supports object oriented programming

1. Classes are the blueprints of objects. One of the big differences between functions and classes is that a class contains both data (variables) and functions that form a package called an: 'object'.
2. Class is a programmer-defined data type, which includes local methods and local variables.
3. Class is a collection of objects. Object has properties and behavior.

Syntax: We define our own class by starting with the keyword 'class' followed by the name you want to give your new class.

```
<?php
   class person {

   }
?>
```

Objects in PHP

An object is an individual instance of the data structure defined by a class. We define a class once and then make many objects that belong to it. Objects are also known as instances.

Creating an Object:

Following is an example of how to create object using **new** operator.

```
<?php
 class Books {
   // Members of class Books
 }

 // Creating three objects of Books
 $physics = new Books;
 $maths = new Books;
 $chemistry = new Books;
 ?>
```

PHP Forms

Form processing is an essential component of almost any web application. Forms are how users communicate with your server: signing up for a new account, searching a forum for all the posts about a particular subject, retrieving a lost password, finding a nearby restaurant or shoemaker, or buying a book.

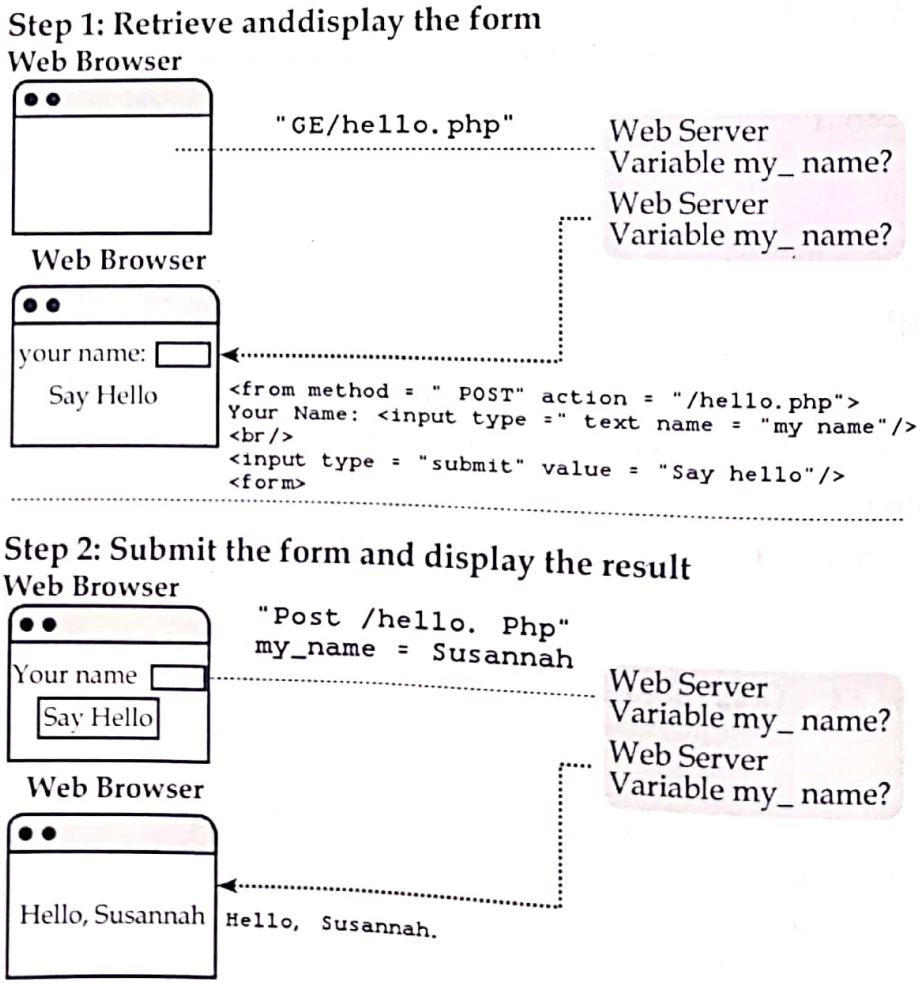
Using a form in a PHP program is a two-step activity. Step one is to display the form. This involves constructing HTML that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons.

When a user sees a page with a form in it, she inputs the requested information into the form and then clicks a button or hits Enter to send the form information back to your server. Processing that submitted form information is step two of the operation.

Accessing Form Elements

At the beginning of every request, the PHP engine sets up some auto-global arrays that contain the values of any parameters submitted in a form or passed in the URL. URL and form parameters from GET method forms are put into $_GET. Form parameters from POST method forms are put into $_POST.

The following figure indicates displaying and processing a simple form



The $_SERVER auto-global array holds a variety of information about your server and the current request the PHP engine is processing. The PHP_SELF element of $_SERVER holds the pathname part of the current request's URL, which is shown in the example below.

Example Form processing in PHP with $_SERVER and PHP_SELF using here document

```
<?php
if ('POST' == $_SERVER['REQUEST_METHOD']) {
   print "Hello, ". $_POST['my_name'];
} else {
   print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name" >
<br>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

Output:

Your name:
Say Hello

When Say Hello button is pressed the page is processed by $_SERVER which holds the current request URL's using PHP_SELF and redirected to the page and the $_SERVER ['REQUEST_METHOD'] checks the method in the form tag and the form parameters with name having value my_name is put into S_POST['my_name'] for futher processing.

Useful Server Variables

In addition to **PHP_SELF** and **REQUEST_METHOD**, the **$_SERVER** auto-global array contains a number of useful elements that provide information on the web server and the current request. The following are some of the entries that **$_SERVER** holds

Element	Description
QUERY_STRING	The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL *http://www.example.com/catalog/store.php?category=kitchen&price=5.*
PATH_INFO	Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example PATH_INFO shown is for the URL *http://www.example.com/catalog/store.php/browse.*
SERVER_NAME	The name of the website on which the PHP engine is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.

DOCUMENT_ROOT	The directory on the web server computer that holds the documents available on the website. If the document root is */usr/local/htdocs* for the website *http://www.example.com*, then a request for *http://www.example.com/catalog/store.php* corresponds to the file */usr/local/htdocs/catalog/store.php*.
REMOTE_ADDR	The IP address of the user making the request to your web server.
REMOTE_HOST	If your web server is configured to translate user IP addresses into hostnames, this is the hostname of the user making the request to your web server. Because this address-to-name translation is relatively expensive (in terms of computational time), most web servers do not do it.
HTTP_REFERER	If someone clicked on a link to reach the current URL, HTTP_REFERER contains the URL of the page that contained the link. This value can be faked, so don't use it as your sole criterion for giving access to private web pages. It can, however, be useful for finding out who's linking to you.
HTTP_USER_AGENT	The web browser that retrieved the page. The example value is the signature of Firefox 37 running on OS X. Like with HTTP_REFERER, this value can be faked, but is useful for analysis.

Example Accessing Form parameters using $_POST

```
<html>
<head>
    <title>Form</title>
</head>
<body>
<form method="POST" action="process.php">
Name:<input type="text" name="name">
Email: <input type="email" name="email">
<input type="submit" name="submit">
</form>
</body>
</html>
```

Output:

Name: John Email: John@gmail.com Submit

After the form is submitted it will redirect to the process.php as the action="process.php" in the form

The process.php page looks like:

```
<?php
echo "The name and emails are:";
echo "Name: " . $_POST['name'];
echo "Email: ". $_POST['email'];
?>
```

And the output is displayed as:

The name and emails are: Name: John Email: John@ gmail.com

Form Validation

Data validation is one of the most important parts of a web application. Weird, wrong, and damaging data shows up where you least expect it. Users can be careless, malicious, and fabulously more creative (often accidentally) than you may ever imagine when you are designing your application. Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows:

- Client-Side Validation – Validation is performed on the client machine web browsers.
- Server Side Validation – After submitted by data, the data has sent to a server and perform validation checks in server machine.

Server Side Validation

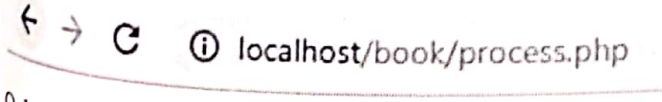
Text validation:

For validating text we use regular expression i.e**preg_match()**. This function searches string for pattern, returns true if pattern exists, otherwise returns false. Usually search starts from beginning of subject string. The optional parameter offset is used to specify the position from where to start the search.

Example Text Validation with preg_match()

```
<?php
$name = $_POST['name'];
if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
   echo "Only letters and white space allowed";
}
else{
   echo "The name is". $name;
}
?>
```

Output:



Only letters and white space allowed

In the above example the preg_match() function searches in string given through a variable $name for the pattern of a-zA-z.

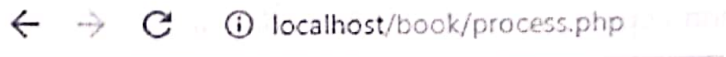
Email validation

To validate the email, use **FILTER_VALIDATE_EMAIL**

Example Email validation using FILTER_VALIDATE_EMAIL

```
<?php
// Variable to check
$email = $_POST['email'];
// Validate email
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Output:



john.doe@example.com is a valid email address

Number Validation

To validate the integer, use **FILTER_VALIDATE_INT** andto validate float, use **FILTER_VALIDATE_FLOAT**

Some useful validating functions

- **htmlspecialchars**:converts some predefined characters to HTML entities.
- **stripslashes**: function removes backslashes
- **trim**: removes the white spaces

Cookies and Session

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the **setcookie()** function.

Syntax

setcookie(name, value, expire, path, domain, secure, httponly);

Only the name parameter is required. All other parameters are optional.

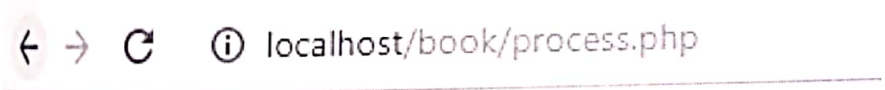
PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer). We then retrieve the value of the cookie "user" (using the global variable **$_COOKIE**). We also use the **isset()** function to find out if the cookie is set.

Example Creating and retrieving a cookie

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); //8640
0 = 1 day
if(!isset($_COOKIE[$cookie_name])) {
echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
echo "Cookie '" . $cookie_name . "' is set!<br>";
echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
<html>
<body>
```

Output:



Cookie 'user' is set!

Value is : John Doe

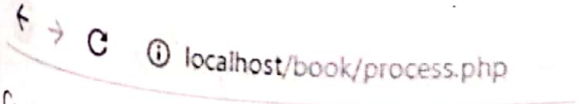
Deleting a cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example Deleting a cookie

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
echo "Cookie 'user' is deleted.";
?>
```

Output:



Cookie 'user' is deleted

Session

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state. Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser. So; Session variables hold information about one single user, and are available to all pages in one application.

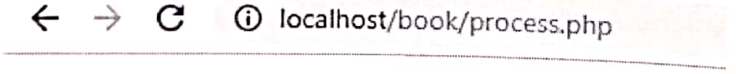
Starting PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page. Session variables are stored in associative array called **$_SESSION[].** These variables can be accessed during lifetime of a session. The following example starts a session then register a variable called counter that is incremented each time the page is visited during the session. Make use of **isset()** function to check if session variable is already set or not.

Example Starting session and setting session values

```
<?php
// Start the session
session_start();
// Set session variables
$_SESSION["username"] = "john";
$_SESSION["email"] = "ducky@gmail.com";
echo "Session variables are set.";
?>
```

Output:



Session variable are set

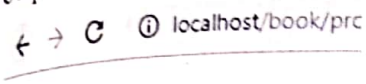
Getting PHP Session Variable values

Next, we create another example. From this example, we will access the session information we set on the previous example. Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Example Getting session variables

```
<?php
session_start();
// Echo session variables that were set on previous page
echo "Username is " . $_SESSION["username"] . ".<br>";
echo "Email is " . $_SESSION["email"] . ".";
?>
```

Output:



Username is john.

Email isducky@gmail.com

Destroy a PHP Session

To remove all global session variables and destroy the session, use **session_unset()** and session_destroy()

Example Destroying a session

```
<?php
session_start();
?>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
?>
```

Working with Databases

With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP. The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows. Databases are useful for storing information categorically. There are three ways of working with MySQl and PHP

1. MySQLi (object-oriented)
2. MySQLi (procedural)
3. PDO

Above the three ways we will be doing the procedural approach in this section

Example: Creating Connection using object-oriented Approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Creating connection
$conn = new mysqli($servername, $username, $password);
// Checking connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

In the above example, the variable name $servername is the name of the server, the variable $username is the username of the server and the variable $password is the password used to connect the server. The **new mysqli()** function takes the parameters and the creates the connection, the **connect_error()** returns the error if there is any error in the process of being connected.

Example Creating Connection using procedural approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Creating connection
$conn = mysqli_connect($servername, $username, $password);
// Checking connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

In MySQLi procedural approach instead of creating an instance we can use the **mysqli_connect()** function available in PHP to establish a connection. This function takes the information as arguments such as host, username , password , database name etc. This function returns MySQL link identifier on successful connecction or FALSE when failed to establish a connection.

The some of the mysqli function used in procedural approach are:

- **mysqli_connect:** Ifopens a new connection to the MySQL server.
- **mysqli_connect_error:** If generates the error while creating the connection
- **mysqli_connect_errno:** Ifgenerates the error number while creating the connection
- **mysqli_query:** performs a query against the database
- **mysqli_error:** If returns the last error description for the most recent function call
- mysqli_errno: Ifreturns the last error code for the most recent function call
- mysqli_num_rows: Ifreturns the number of rows in a result set.
- mysqli_close: If closes the connection

Example Creating Database using procedural approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
// Create database
$sql = "CREATE DATABASE webTech";
if (mysqli_query($conn, $sql)) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Output:

Database created successfully

Inserting Multiple Data

Multiple SQL statements must be executed with the **mysqli_multi_query()** function.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "web";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO student (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO student (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO student (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
   echo "Records inserted successfully";
} else {
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Output

Records inserted successfully.

Introduction to PHP Framework

Codeigniter Framework

CodeIgniter is an Application Development Framework - a toolkit - for people who build web sites using PHP. Its goal is to enable you to develop projects much faster than you could if you were writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. CodeIgniter lets you creatively focus on your project by minimizing the amount of code needed for a given task.

Who is Code Igniter For?

Code Igniter is right for you if:

- You want a framework with a small footprint.
- You need exceptional performance.
- You need broad compatibility with standard hosting accounts that run a variety of PHP versions and configurations.
- You want a framework that requires nearly zero configuration.
- You want a framework that does not require you to use the command line.
- You want a framework that does not require you to adhere to restrictive coding rules.
- You are not interested in large-scale monolithic libraries like PEAR.
- You do not want to be forced to learn a templating language (although a template parser is optionally available if you desire one).
- You eschew complexity, favoring simple solutions.
- You need clear, thorough documentation.

To use codeigniter just go to https://codeload.github.com/bcit-ci/CodeIgniter/zip/3.1.11 and download the codeigniter 3.1.11 which is the current version

Installation Instructions

CodeIgniter is installed in four steps:

- Unzip the package.
- Upload the CodeIgniter folders and files to your server. Normally the *index.php* file will be at your root.
- Open the *application/config/config.php* file with a text editor and set your base URL. If you intend to use encryption or sessions, set your encryption key.
- If you intend to use a database, open the *application/config/database.php* file with a text editor and set your database settings.

Laravel Framework

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked.

Installing Laravel

Laravel utilizes Composer to manage its dependencies. So, before using Laravel, make sure you have Composer installed on your machine.

Via Laravel Installer

First, download the Laravel installer using Composer:

```
composer global require laravel/installer
```

Make sure to place Composer's system-wide vendor bin directory in your $PATH so the laravel executable can be located by your system. This directory exists in different locations based on your operating system; however, some common locations include:

- macOS and GNU / Linux Distributions: $HOME/.config/composer/vendor/bin
- Windows: %USERPROFILE%\AppData\Roaming\Composer\vendor\bin

Once installed, the laravel new command will create a fresh Laravel installation in the directory you specify. For instance, laravel new blog will create a directory named blog containing a fresh Laravel installation with all of Laravel's dependencies already installed:

```
laravel new blog
```

Via Composer Create-Project

Alternatively, you may also install Laravel by issuing the Composer create-project command in your terminal:

```
composer create-project --prefer-distlaravel/laravel blog
```

Local Development Server

If you have PHP installed locally and you would like to use PHP's built-in development server to serve your application, you may use the serve Artisan command. This command will start a development server at http://localhost:8000:

```
php artisan serve
```

Introduction to Wordpress

WordPress is a free and open source content management system (CMS) based on PHP and MySQL. It is the most widely used CMS software in the world, and as of May 2019, it powers more than 30% of the top 10 million websites and has an estimated 60% market share of all websites built using a CMS.

WordPress started as a simple blogging system in 2003, but it has evolved into a full CMS with thousands of plugins, widgets, and themes. It is licensed under the General Public License (GPLv2 or later).

DISCUSSION EXERCISE

1. What is server side scripting language? Why we need web server in order to develop web application?
2. What is PHP? Describe awesome feature provided by PHP.
3. What is Variable? Write rules to create variable in PHP.
4. Describe importance of comment in programming? Describe PHP comments.
5. Describe different operators in PHP with example.
6. What is array in PHP? Describe different types of array in PHP with example.
7. Difference between for and for each loop also with coding example.
8. What is Global variable? Describe different global variable/array used by PHP.
   a. $_POST
   b. $_GET
   c. $_REQUEST
   d. $_SERVER
9. Explain how we can achieve file inclusion in PHP with different PHP functions.
10. Describe how we can upload file using PHP.
11. Describe different file handling function used by PHP.
12. What is session? Describe how session works with real time example.
13. Explain briefly advantages of session over cookie.
14. Describe setcookie() function with example.
15. Difference between echo and print.
16. What is the difference between $message and $message?
17. Describe different PHP data types with example.
18. What is function? Describe different types of user defined function in PHP.
19. Explain following function with example:
    a. ucfirst(), strtolower(), strtoupper(), strlen(),strrev(),ucwords(),lcfirst(),explode(),implode(),str_replace(),strcmp(),substr(),strpos()
    b. array_chunk(),array_columns(),array_keys(),array_values(),array_pop(),array_push(),array_shift(),array_unshift(),array_slice(),array_splice(),sort(),asort()

❑❑❑