ESTD. 2001

PRATHYUSHA ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGULATION R2021

III YEAR - V SEMESTER

CCS375 - WEB TECHNOLOGIES

CCS375 - WEB TECHNOLOGIES

COURSE OBJECTIVES:

- To understand different Internet Technologies
- To learn java-specific web services architecture
- To Develop web applications using frameworks

UNIT I WEBSITE BASICS, HTML 5, CSS 3, WEB 2.0 7

Web Essentials: Clients, Servers and Communication – The Internet – World wide web – HTTP Request Message – HTTP Response Message – Web Clients – Web Servers – HTML5 – Tables – Lists – Image – HTML5 control elements – Drag and Drop – Audio – Video controls - CSS3 – Inline, embedded and external style sheets – Rule cascading – Inheritance – Backgrounds – Border Images – Colors – Shadows – Text – Transformations – Transitions – Animations. Bootstrap Framework

UNIT II CLIENT SIDE PROGRAMMING 6

Java Script: An introduction to JavaScript–JavaScript DOM Model-Exception Handling-Validation- Built-in objects-Event Handling- DHTML with JavaScript- JSON introduction – Syntax – Function Files.

UNIT III SERVER SIDE PROGRAMMING 5

Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies- DATABASE CONNECTIVITY: JDBC. 121

UNIT IV PHP and XML 6

An introduction to PHP: PHP- Using PHP- Variables- Program control- Built-in functions-Form Validation. XML: Basic XML- Document Type Definition- XML Schema, XML Parsers and Validation, XSL ,

UNIT V INTRODUCTION TO ANGULAR and WEB APPLICATIONS FRAMEWORKS 6

Introduction to AngularJS, MVC Architecture, Understanding ng attributes, Expressions and data binding, Conditional Directives, Style Directives, Controllers, Filters, Forms, Routers, Modules, Services; Web Applications Frameworks and Tools – Firebase- Docker- Node JS-React- Django- UI & UX.

COURSE OUTCOMES:

CO1: Construct a basic website using HTML and Cascading Style Sheets
CO2: Build dynamic web page with validation using Java Script objects and by applying different event handling mechanisms.
CO3: Develop server side programs using Servlets and JSP.
CO4: Construct simple web pages in PHP and to represent data in XML format.
CO5: Develop interactive web applications.

TEXTBOOKS
1. Deitel and Deitel and Nieto, Internet and World Wide Web - How to Program, Prentice Hall, 5th Edition, 2011.
2. Jeffrey C and Jackson, Web Technologies A Computer Science Perspective, Pearson Education, 2011.
3. Angular 6 for Enterprise-Ready Web Applications, Doguhan Uluca, 1st edition, Packt Publishing

REFERENCES:
1. Stephen Wynkoop and John Burke "Running a Perfect Website", QUE, 2nd Edition,1999.
2. Chris Bates, Web Programming – Building Intranet Applications, 3rd Edition, Wiley Publications, 2009.

CCS375 WEB TECHNOLOGIES
UNIT I WEBSITE BASICS, HTML 5, CSS 3, WEB 2.0 9
Web Essentials: Clients, Servers and Communication – The Internet – Basic Internet protocols – World wide web – HTTP Request Message – HTTP Response Message – Web Clients – Web Servers – HTML5 – Tables – Lists – Image – HTML5 control elements – Semantic elements – Drag and Drop – Audio – Video controls - CSS3 – Inline, embedded and external style sheets – Rule cascading – Inheritance – Backgrounds – Border Images – Colors – Shadows – Text – Transformations – Transitions – Animations.

WEB ESSENTIALS

Web Essentials:
Server:
The software that distributes the information and the machine where the
information and softwarereside is called the server.
• provides requested service to client
• e.g., Web server sends requested Web page
Client:
The software that resides on the remote machine, communicates with the
server, fetches theinformation, processes it, and then displays it on the remote
machine is called the client.
• initiates contact with server (―speaks first‖)
• typically requests service from server
• Web: client implemented in browser
Web server:
Software that delivers Web pages and other documents to browsers using the HTTP protocol
Web Page:
A web page is a document or resource of information that is suitable for the World
Wide Web andcan be accessed through a web browser.
Website:
A collection of pages on the World Wide Web that is accessible from the same URL
and typicallyresiding on the same server.
URL:
Uniform Resource Locator, the unique address which identifies a resource on
the Internet forrouting purposes.
Client-server paradigm:
IThe Client-Server paradigm is the most prevalent model for distributed computing protocols. It is
the basis of all distributed computing paradigms at a higher level of abstraction. It is
service-oriented, andemploys a request-response protocol.
A server process, running on a server host, provides access to a service. A client
process, running on a client host, accesses the service via the server process.The
interaction of the process proceedsaccording to a protocol.
The primary idea of a client/server system is that you have a central repository of
information—some kind of data, often in a database—that you want to distribute on

demand to some set ofpeople or machines.

The Internet:

• Medium for communication and interaction in inter connected network.

• Makes information constantly and instantly available to anyone with a connection.

Web Browsers:

• User agent for Web is called a browser:o Internet Explorer

o Firefox

Web Server:

• Server for Web is called Web server:

o Apache (public domain)

o MS Internet Information Server

Protocol:

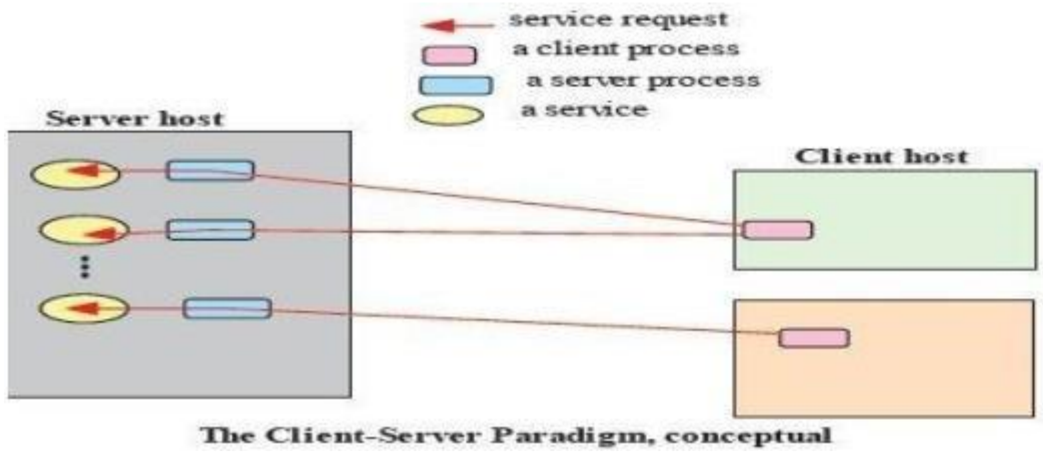
Protocols are agreed formats for transmitting data between devices.The protocol determines:

i. The error checking required

ii. Data compression method used

iii. The way the end of a message is signalled

iv. The way the device indicates that it has received the message



The Client-Server Paradigm, conceptual

Internet Protocol:

There are many protocols used by the Internet and the WWW:

o TCP/IP

o HTTP

o FTP

o Electronic mail protocols IMAP

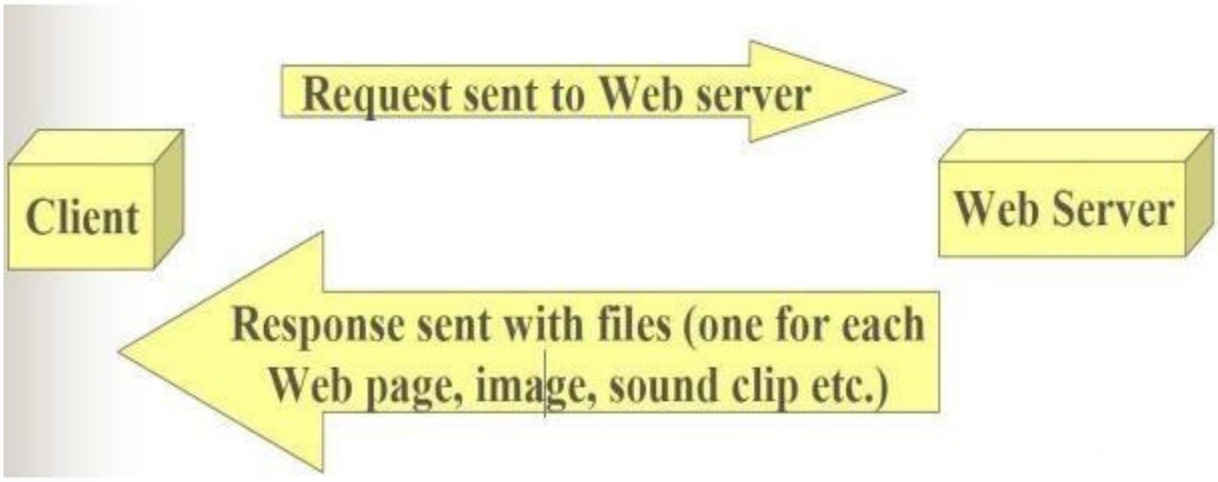
o POP

TCP/IP

The Internet uses two main protocols (developed by Vincent Cerf and Robert Kahn) Transmission control protocol (TCP):Controls disassembly of message into packets at the originreassembles at the destination

Internet protocol (IP):Specifies the addressing details for each packet Each packet is labelled withits origin and destination.

Hypertext Transfer Protocol (HTTP)
• The hypertext transfer protocol (HTTP) was developed by Tim Berners-Lee in 1991
• HTTP was designed to transfer pages between machines
• The client (or Web browser) makes a request for a given page and the Server is responsible forfinding it and returning it to the client
• The browser connects and requests a page from the server
• The server reads the page from the file system, sends it to the client andterminates the connection.


Request sent to Web server
Client
Web Server
Response sent with files (one for each Web page, image, sound clip etc.)

Electronic Mail Protocols:
• Electronic mail uses the client/server model
• The organisation has an email server devoted to handling emailo Stores and forwards email messages
• Individuals use email client software to read and send emailo (e.g. Microsoft Outlook, or Netscape Messenger)
• Simple Mail Transfer Protocol (SMTP)
o Specifies format of mail messages
• Post Office Protocol (POP) tells the email server to:
o Send mail to the user’s computer and delete it from the server
o Send mail to the user’s computer and do not delete it from the server
o Ask whether new mail has arrived.
Interactive Mail Access Protocol (IMAP)
Newer than POP, provides similar functions with additional features.
o e.g. can send specific messages to the client rather than all the messages. A user can view email message headers and the sender’s name before downloading the entire message.
Allows users to delete and search mailboxes held on the email server.
The disadvantage of POP: You can only access messages from one PC.
The disadvantage of IMAP : Since email is stored on the email server, there is a need for more and moreexpensive (high speed) storage space.
World Wide Web: comprises software (Web server and browser) and data (Web sites).
Internet Protocol (IP) Addresses:
- Every node has a unique numeric address
- Form: 32-bit binary number

- New standard, IPv6, has 128 bits (1998)
- Organizations are assigned groups of IPs for their computers
- **Domain names**
- Form: host-name. domain-names
- First domain is the smallest (Google)
- Last domain specifies the type of organization (.com)
- Fully qualified domain name - the host name and all of the domain names
- DNS servers - convert fully qualified domain names to IPs

HTTP:

□ □Hypertext Transfer Protocol (HTTP) is the communication protocol used by the Internet to transfer hypertext documents.

□ □A protocol to transfer hypertext requests and information between servers and browsers

•□Hypertext is text, displayed on a computer, with references (hyperlinks) to other text that the reader can immediately follow, usually by a mouse HTTP is behind every request for aweb document or graph, every click of a hypertext link, and every submission of a form.

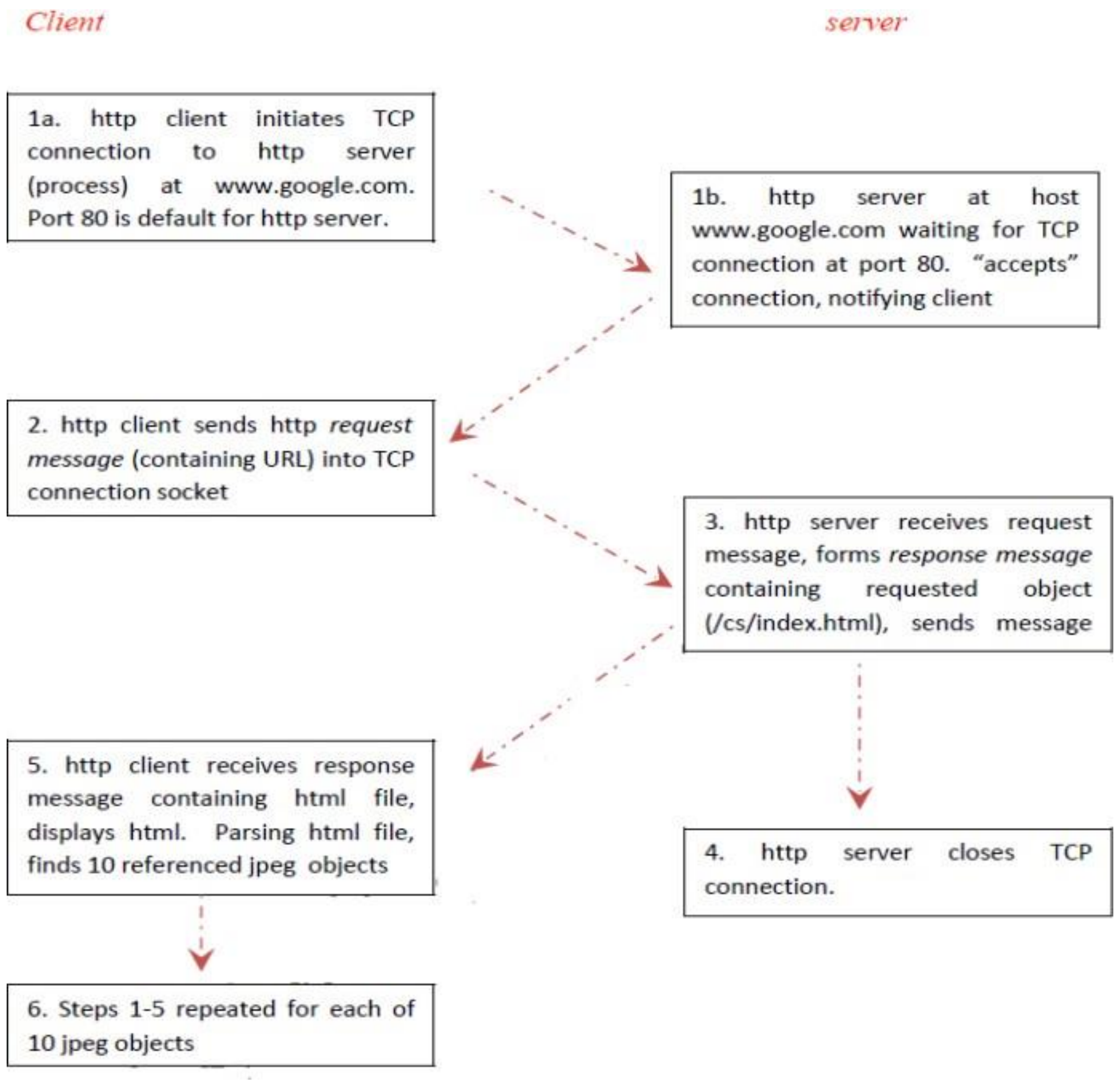
□□HTTP specifies how clients **request** data, and how servers **respond** to these requests.

□□The client makes a request for a given page and the server is responsible for finding it and returning it to the client.

□□The browser connects and requests a page from the server.

•□The server reads the page from the file system and sends it to the client and thenterminates the connection

HTTP Transactions



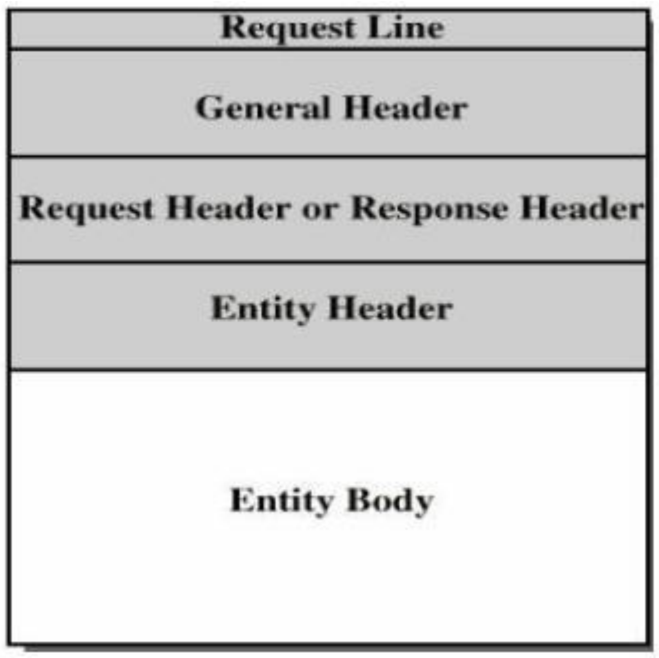
Client
server
1a. http client initiates TCP connection to http server (process) at www.google.com. Port 80 is default for http server.
1b. http server at host www.google.com waiting for TCP connection at port 80. "accepts" connection, notifying client
2. http client sends http *request message* (containing URL) into TCP connection socket
3. http server receives request message, forms *response message* containing requested object (/cs/index.html), sends message
4. http server closes TCP connection.
5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
6. Steps 1-5 repeated for each of 10 jpeg objects

HTTP Message:

HTTP message is the information transaction between the client and server.

Two types of HTTP Message:

1. Requests
a. Client to server
2. Responses
a. Server to client



Fields

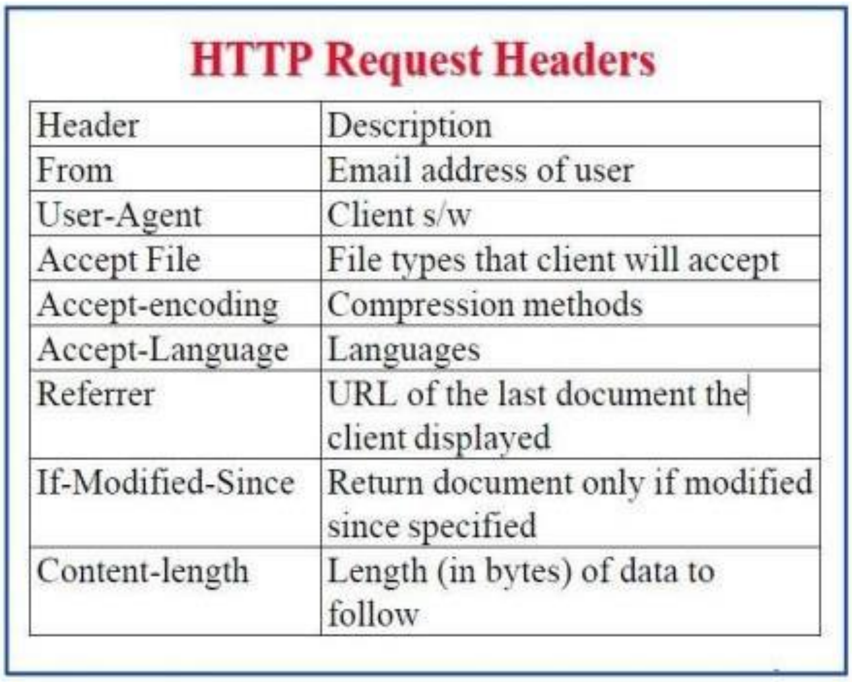
· Request line or Response line
· General header
· Request header or Response header
· Entity header
· Entity body

.10 Request Message:

Request Line:

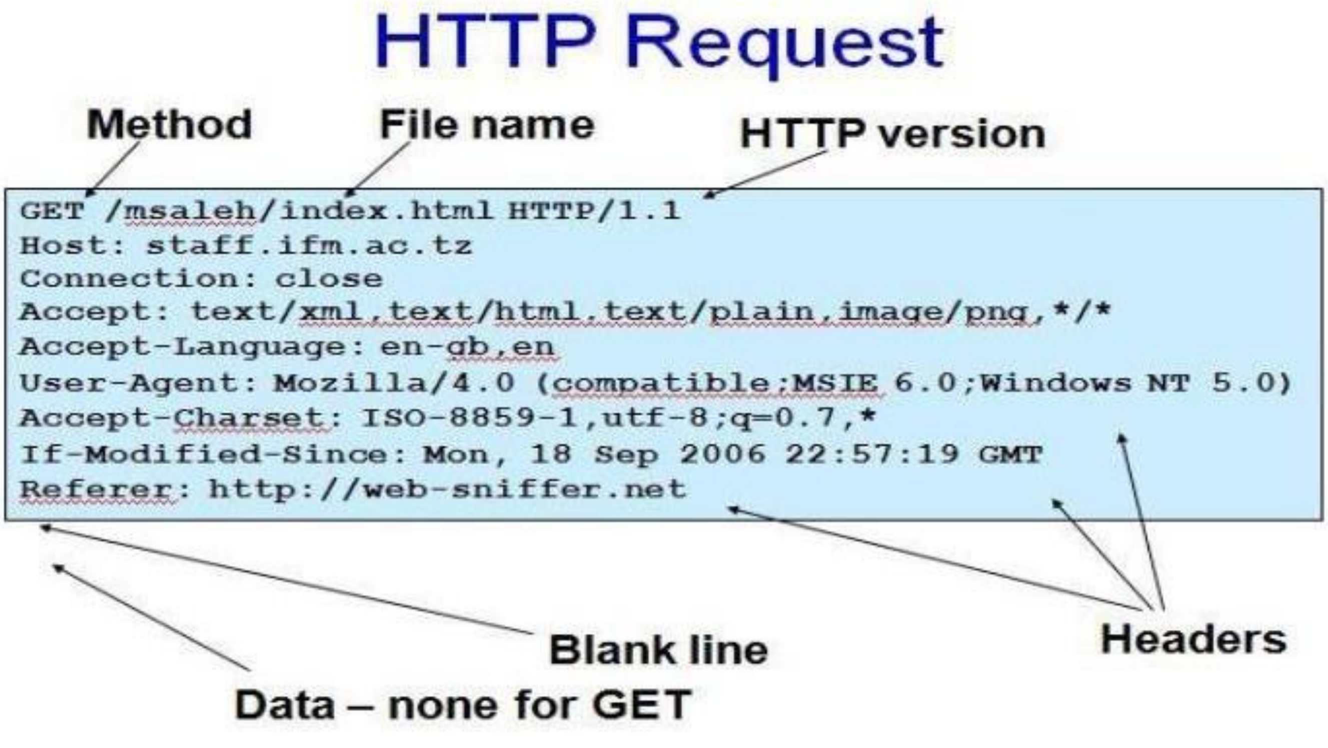
• A request line has three parts, separated by spaceso a *method* name
o the local path of the requested resource
o the version of HTTP being used
• A typical request line is:
o GET /path/to/file/index.html HTTP/1.1
• Notes:
o **GET** is the most common HTTP method; it says "give me this resource". Othermethods include **POST** and **HEAD.** Method names are always uppercase
o The path is the part of the URL after the host name, also called the *request URI*
o The HTTP version always takes the form "**HTTP/x.x**", uppercase.

Request Header:



HTTP Request Headers

Header	Description
From	Email address of user
User-Agent	Client s/w
Accept File	File types that client will accept
Accept-encoding	Compression methods
Accept-Language	Languages
Referrer	URL of the last document the client displayed
If-Modified-Since	Return document only if modified since specified
Content-length	Length (in bytes) of data to follow



```
GET /msaleh/index.html HTTP/1.1
Host: staff.ifm.ac.tz
Connection: close
Accept: text/xml,text/html,text/plain,image/png,*/*
Accept-Language: en-gb,en
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*
If-Modified-Since: Mon, 18 Sep 2006 22:57:19 GMT
Referer: http://web-sniffer.net
```

.11 Response Message:

Response Line:

• A request line has three parts, separated by spaceso the HTTP version,

o a *response status code* that gives the result of the request, and

o an English *reason phrase* describing the status code

• Typical status lines are:

o HTTP/1.0 200 OK or

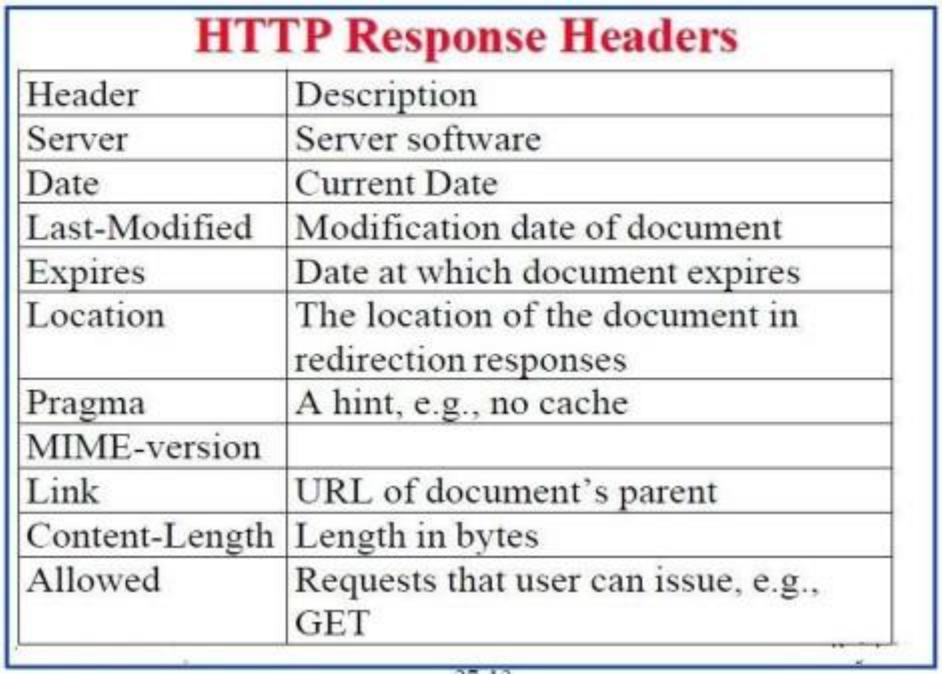
o HTTP/1.0 404 Not Found

• Notes:

o The HTTP version is in the same format as in the request line, "**HTTP/x.x**".

o The status code is meant to be computer-readable; the reason phrase is meant to behuman-readable, and may vary.

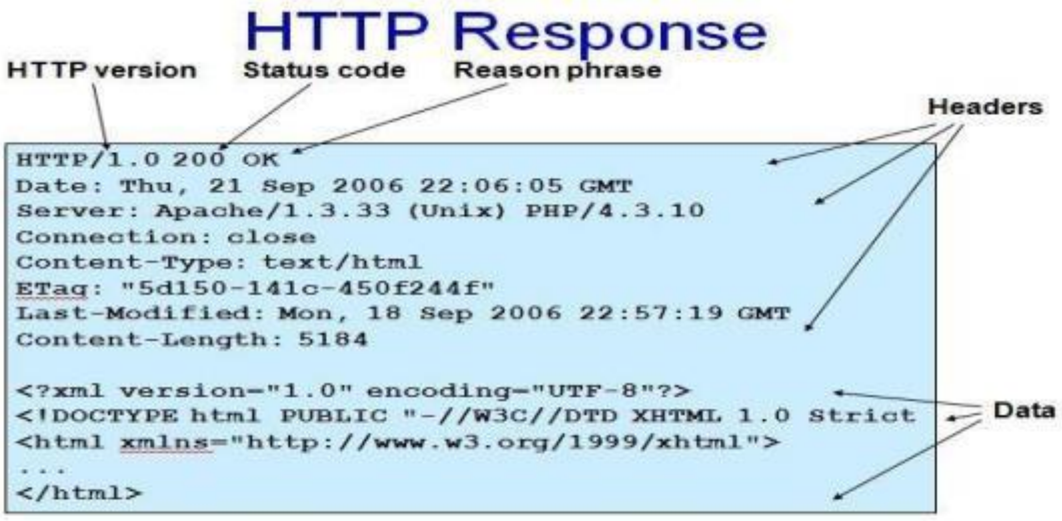
HTTP Request Header:



HTTP Response Headers

Header	Description
Server	Server software
Date	Current Date
Last-Modified	Modification date of document
Expires	Date at which document expires
Location	The location of the document in redirection responses
Pragma	A hint, e.g., no cache
MIME-version	
Link	URL of document's parent
Content-Length	Length in bytes
Allowed	Requests that user can issue, e.g., GET

EXAMPLE



```
HTTP/1.0 200 OK
Date: Thu, 21 Sep 2006 22:06:05 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.10
Connection: close
Content-Type: text/html
ETag: "5d150-141c-450f244f"
Last-Modified: Mon, 18 Sep 2006 22:57:19 GMT
Content-Length: 5184

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

HTTP Method:

• HTTP method is supplied in the request line and specifies the operation that the client hasrequested.

Some common methods:

• Options
• Get
• Head
• Post
• Put
• Move

• Delete

Two methods that are mostly used are the GET and POST:

o **GET** for queries that can be safely repeated

o **POST** for operations that may have side effects (e.g. ordering a book from an on-line store).

The GET Method

• It is used to retrieve information from a specified URI and is assumed to be a safe, repeatableoperation by browsers, caches and other HTTP aware components

• Operations have no side effects and GET requests can be re-issued.

• For example, displaying the balance of a bank account has no effect on the account and can besafely repeated.

• Most browsers will allow a user to refresh a page that resulted from a **GET**, without displayingany kind of warning

• Proxies may automatically retry **GET** requests if they encounter a temporary network connectionproblem.

• GET requests is that they can only supply data in the form of parameters encoded in the URI(known as a **Query String**) – [downside]

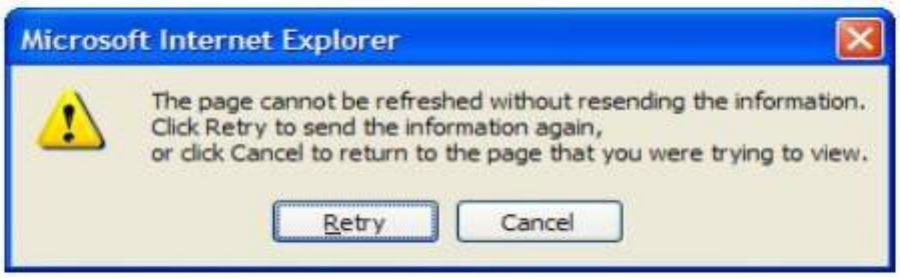
• Cannot be unused for uploading files or other operations that require large amounts of data to besent to the server.

The POST Method

• Used for operations that have side effects and cannot be safely repeated.

• For example, transferring money from one bank account to another has side effects and shouldnot be repeated without explicit approval by the user.

• If you try to refresh a page in Internet Explorer that resulted from a **POST**, it displays thefollowing message to warn you that there may



Microsoft Internet Explorer

The page cannot be refreshed without resending the information.
Click Retry to send the information again,
or click Cancel to return to the page that you were trying to view.

The **POST** request message has a content body that is normally used to sendparameters and data

• The IIS server returns two status codes in its response for a **POST** request

o The first is **100 Continue** to indicate that it has successfully received the **POST** request

o The second is **200 OK** after the request has been processed.

HTTP response status codes

• Informational (1xx)

• Successful (2xx)

• Redirection (3xx)

o 301: moved permanently

• Client error (4xx)

o 403 : forbidden

o 404: Not found

• Server error (5xx)
o 503: Service unavailable
o 505: HTTP version not supported

1.12 HTTP

❖ **Features**

• Persistent TCP Connections: Remain open for multiple requests
• Partial Document Transfers: Clients can specify start and stop positions
• Conditional Fetch: Several additional conditions
• Better content negotiation
• More flexible authentication.

Web Browsers :

• Mosaic - NCSA (Univ. of Illinois), in early 1993 First to use a GUI, led to Explosion of Web use Initially for X-Windows, under UNIX, but was ported to other platforms by late 1993
• Browsers are clients - always initiate, servers react (although sometimes servers require responses)
• Most requests are for existing documents, using Hypertext Transfer Protocol
• (HTTP) But some requests are for program execution, with the output being returned as a document.

Browser: A web browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.

Web Servers:

- Provide responses to browser requests, either existing documents or dynamically Built documents.
- Browser-server connection is now maintained through more than one request- Response cycle
- All communications between browsers and servers use Hypertext Transfer Protocol
- Web servers run as background processes in the operating system.
- Monitor a communications port on the host, accepting HTTP messages when they appear All current Web servers came from either

1. The original from CERN
2. The second one, from NCSA

- Web servers have two main directories:

1. Document root (servable documents)
2. Server root (server system software)

- Document root is accessed indirectly by clients
- Its actual location is set by the server Configuration file
- Requests are mapped to the actual location
- Virtual document trees
- Virtual hosts
- Proxy servers
- Web servers now support other Internet protocols
- Apache (open source, fast, reliable)
- IIS
- Maintained through a program with a GUI interface.

HTML 5

- ✓ HTML is the main markup language for describing the structure of web pages.
- ✓ HTML stands for **Hypertext Markup Language**.
- ✓ HTML is the basic building block of World Wide Web.
- ✓ Hypertext is text displayed on a computer or other electronic device with references to other text that theuser can immediately access, usually by a mouse click or key press.
- ✓ Apart from text, hypertext may contain **tables, lists, forms, images**, and other presentational elements. It isan easy-to-use and flexible format to share information over the Internet.
- ✓ Markup languages use sets of markup tags to characterize text elements within a document, which givesinstructions to the web browsers on how the document should appear.
- ✓ HTML was originally developed by **Tim Berners-Lee** in 1990.
- ✓ He is also known as the father of the web. In1996, the World Wide Web Consortium (W3C) became the authority to maintain the HTML specifications. HTML also became an international standard (ISO) in 2000. HTML5 is the latest version of HTML.
- ✓ HTML5 provides a faster and more robust approach to web development.

HTML Tags and Elements

HTML is written in the form of HTML elements consisting of markup tags.
These markup tags arethe fundamental characteristic of HTML.
Every markup tag is composed of a keyword, surroundedby angle brackets, such as <html>, <head>, <body>, <title>, <p>, and so on.
HTML tags normally come in pairs like <html> and </html>.
The first tag in a pair is often called the **opening tag (or start tag),** and the second tag is called the **closing tag (or end tag).**
An opening tag and a closing tag are identical, except a slash (/) after the opening angle bracket ofthe closing tag, to tell the browser that the command has been completed.

HTML5 IMAGE:

Inserting Images into Web Pages

Images enhance visual appearance of the web pages by making them more interesting and colorful.
The <img> tag is used to insert images in the HTML documents. It is an empty element and containsattributes only. The syntax of the <img> tag can be given with:

<img src="*url*" alt="*some_text*">

Attributes	**Description**
src	Specifes the path to the image
alt	Specifies an alternate text
height	Specifies the height of an image
width	Specifies the width
ismap	Specifies an image as a server-side image map
usemap	Define a valid map name

The following example inserts three images on the web page:
Example:

```
<html><head></head>
<body>
<img src="c.jpg" height=="160"  width="130"alt="C++ how to Program">
<img src="java.jpg" height="150" width="130"alt="Java how to program">
</body></html>
```

OUTPUT:

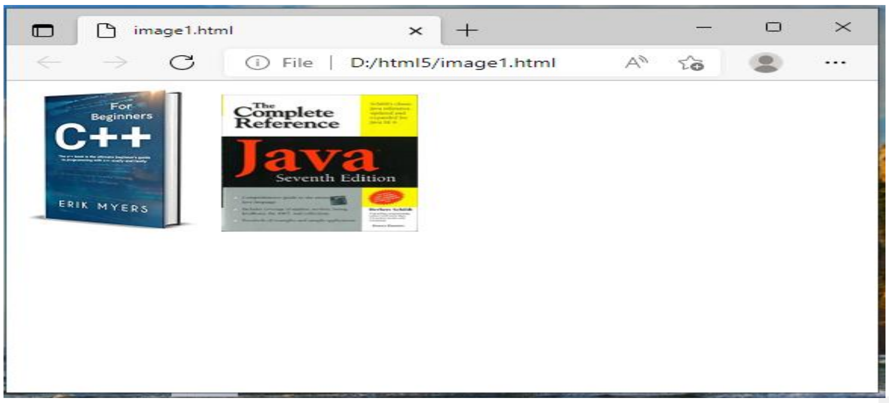


image1.html
File | D:/html5/image1.html
For Beginners C++
ERIK MYERS
The Complete Reference Java Seventh Edition

Each image must carry at least two attributes: **the src attribute, and an alt attribute.** The src attribute tells the browser where to find the image. Its value is the URL of the image file. Whereas, the alt attribute provides an alternative text for the image, if it is unavailable or cannot bedisplayed for some reason. Its value should be a meaningful substitute for the image.

- **Using Image as Hyperlink:**

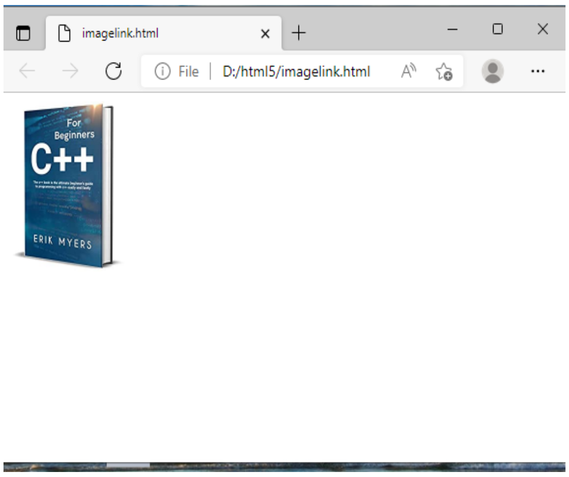
By using image as hyperlinks, web developers can create graphical web pages that link to other resources.
Tag <a> anchor-We can use hyperlinks by using attributes.
Href → specifies the URL.By default links will appear as follows in all browser.

```
<html> <head > </head>
<body>
<a href ="link.html">
<img src="c.jpg" height=="160"  width="130"alt="C++ how to Program">
</a>  </body> </html>
```

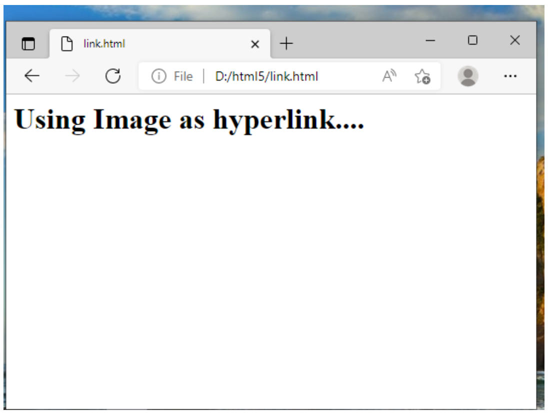
Link.html

```
<html>   <head>  </head>
<body> <h1> Using Image as hyperlink....</h1>
</body>  </html>
```

Output:


imagelink.html

D:/html5/imagelink.html


link.html

D:/html5/link.html

Using Image as hyperlink....

Hyperlink

Link act as a pointer to some web page or documents and image. Both text and image can be acts as hyperlinks.

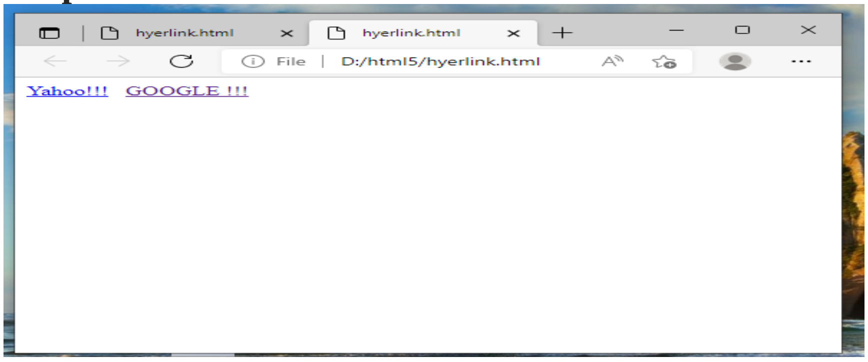
Links are created using the (a) anchor element. Attribute used is href →specifies the URL.

- An unvisited link is blue.
- A visited link is purple.
- An active link is red.

Example:

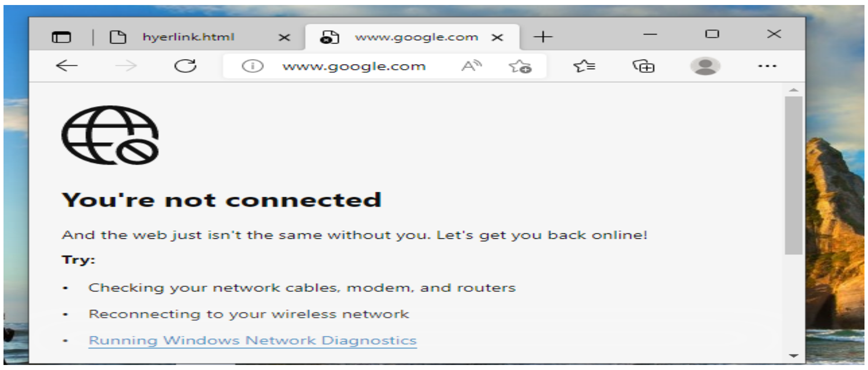
```
<html>
<head></head>
<body>
<a href ="http://www.yahoo.com">Yahoo!!!</a>&nbsp&nbsp
<a href="http://www.google.com">GOOGLE !!!</a>
</body></html>
```

Output:


hyerlink.html

D:/html5/hyerlink.html

Yahoo!!! GOOGLE !!!


hyerlink.html

www.google.com

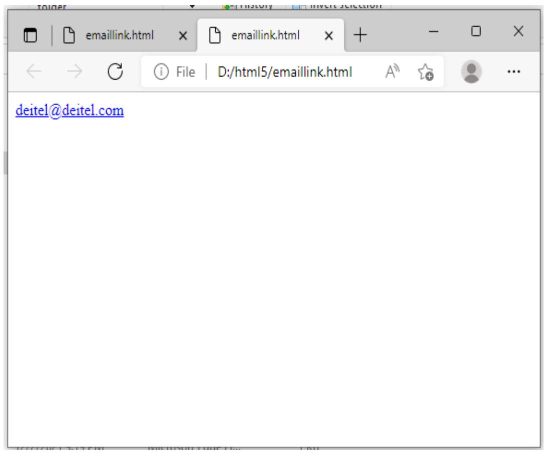
You're not connected

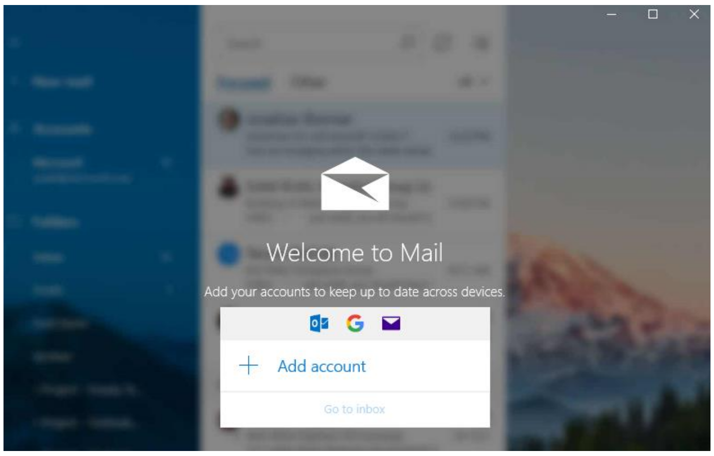
And the web just isn't the same without you. Let's get you back online!

Try:

- Checking your network cables, modem, and routers
- Reconnecting to your wireless network
- Running Windows Network Diagnostics

- Hyperlink to an E-Mail address:Anchor can link to e-mail address using a **mailto:url**
  Format : <a href ="mailto:deitel@deitel.com">deitel@deitel.com</a>


deitel@deitel.com



HTML Lists
HTML lists are used to present list of information in well formed and semantic way. There are threedifferent types of list in HTML and each one has a specific purpose and meaning.

- **Unordered list** — Used to create a list of related items, in no particular order.
- **Ordered list** — Used to create a list of related items, in a specific order.
- **Description list** — Used to create a list of terms and their descriptions.

✓ **HTML Unordered Lists**

An unordered list created using the <ul> element, and each list item starts with the <li> element.The list items in unordered lists are marked with bullets.

<ul>… </ul>	Specifies an unordered list
<li>…</li>	Specifies list item
<ul type=”circle” >	Display the circular bullets
<ul type=”disc”>	Display the solid round bullets
<ul type=”square”>	Display the squared bullets

Here's an example:
```
<ul type=”disc”>
   <li >Chocolate Cake</li>
   <li>Black Forest Cake</li>
   <li>Pineapple Cake</li>
</ul>
```
— The output of the above example will look something like this:

- Chocolate Cake
- Black Forest Cake
- Pineapple Cake

✓ HTML Ordered Lists

An ordered list created using the <ol> element, and each list item starts with the <li> element.Ordered lists are used when the order of the list's items is important.
The list items in an ordered list are marked with numbers.

<ol>… </ol>	Specifies an ordered list
<li>…</li>	Specifies list item
<ul type="A" >	Display the list in the following A,B...
<ul type="I">	Display the list in the following I,II…
<ul type="i">	Display the list in the following i, ii..
<ul type="1">	Display the list in the following 1,2…

Example:

```
<ol type="1">
   <li>Fasten your seatbelt</li>
   <li>Starts the car's engine</li>
   <li>Look around and go</li>
</ol>
```

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

1. Fasten your seatbelt
2. Starts the car's engine
3. Look around and go

✓ HTML Definition list:

Specifically used for lists in which each element is labeled with a word rather than a bullet or number.

Tag	Description
<dl>..</dl>	Specifies a description list
<dt>…</dt>	Specifies the term in a description list
<dd>..</dd>	Specifies description of term a description list

Ex:

```
  <h1> Abbrevation</h1>
 <dl>
  <dt>HTML </dt>
   <dd> Hypertext Markup Language….</dd>
  <dt>CSS </dt>
    <dd> Cascading Style sheet</dd>
 </dl>
```

Output:

Abbreviation

HTML

Hypertext Markup language

CSS

Cascading Style sheet.

HTML Tables

Creating Tables in HTML

HTML table allows you to arrange data into rows and columns. They are commonly used to displaytabular data like product listings, customer's details, financial reports, and so on. You can create a table using the <table> element. Inside the <table> element, you can use the <tr> elements to create rows, and to create columns inside a row you can use the <td> elements.You can also define a cell as a header for a group of table cells using the <th> element.

The following example demonstrates the most basic structure of a table.

Example

```
<table>
    <tr>
        <th>No.</th>
        <th>Name</th>
        <th>Age</th>
     </tr>
     <tr>
        <td>1</td>
        <td>Peter Parker</td>
        <td>16</td>
     </tr>
     <tr>
        <td>2</td>
        <td>Clark Kent</td>
        <td>34</td>
    </tr>
</table>
```

Tables do not have any borders by default. You can use the CSS border property to add borders tothe tables. Also, table cells are sized just large enough to fit the contents by default. To add more space around the content in the table cells you can use the CSS padding property.

Rowspan and colspan:

Used to categorize the information properly in sub rows and sub columns.

Rowspan → Used to extend the row vertically.

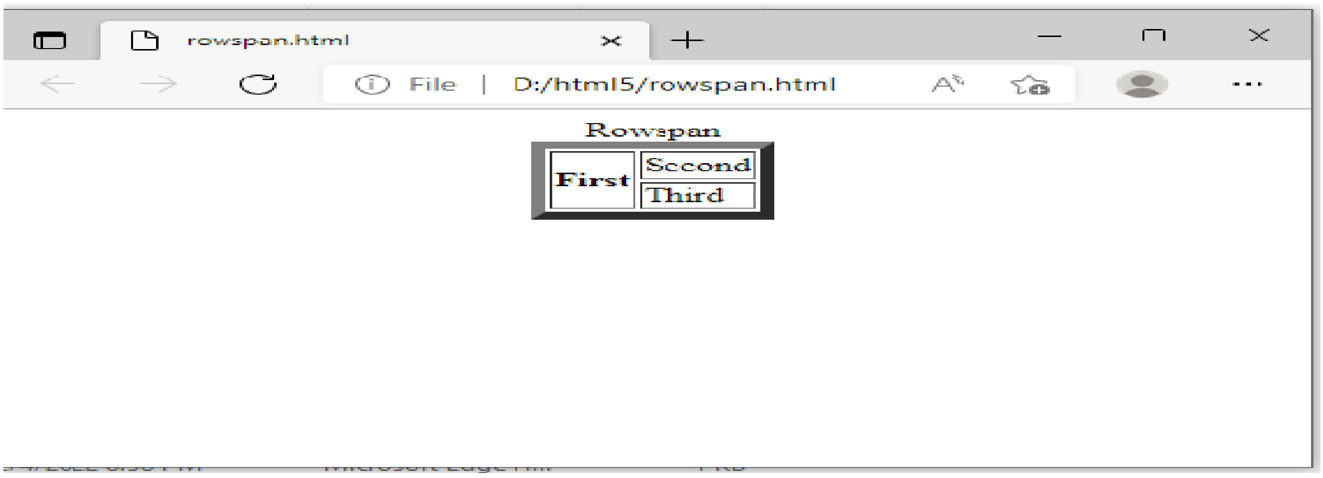
Colspan →Used to extend the column horizontally.

When rowspan =2 then the row can be extended vertically by 2 cells.

Example:

```
<html>
<head>
</head>
<body><center>
<table border="6">
 Rowspan
 <tr>
   <th rowspan="2">First</th>
    <td>Second</td> </tr>
    <tr> <td>Third</td> </tr>
  </table>
</center>
</body>
</html>
```

OUTPUT:



Rowspan

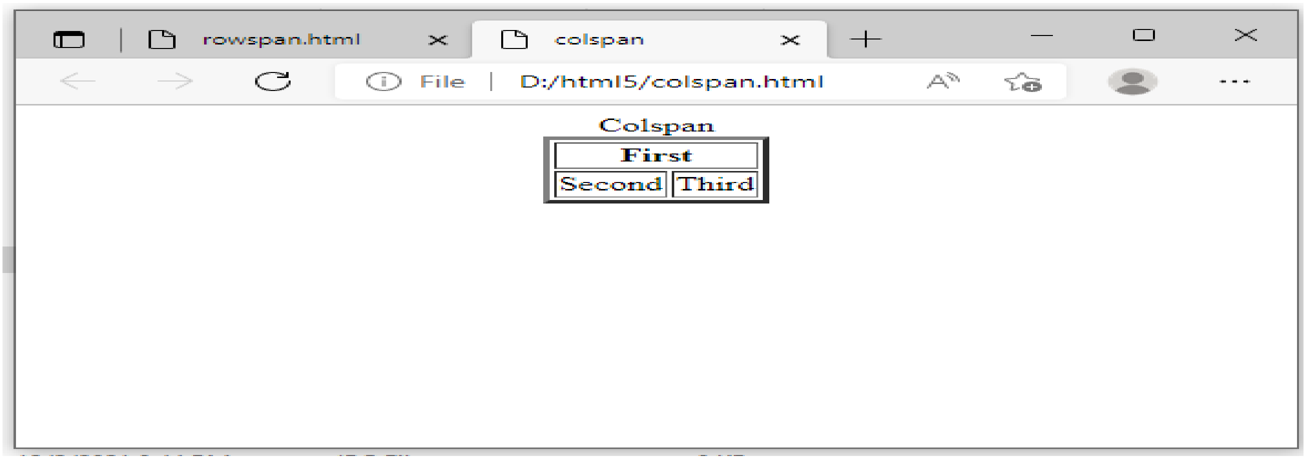
First	Second
	Third

When colspan =2 then the row can be extended horizontally by 2 cells.

Example:

```
<html>
 <head>
  <title> colspan </title>
  </head>
 <body>
  <center>
   <table border="3">
  Colspan
    <tr>
     <th colspan="2">First</th>
    </tr>
    <tr>
     <td>Second</td>
     <td>Third</td>
  </tr>
 </table>
 </center>
 </body>
</html>
```

OUTPUT:



Colspan

First	
Second	Third

Defining a Table Header, Body, and Footer

HTML provides a series of tags <thead>, <tbody>, and <tfoot> that helps you to create morestructured table, by defining header, body and footer regions, respectively.

The following example demonstrates the use of these elements.

Example

```
<table>
   <thead>
      <tr>
         <th>Items</th>
         <th>Expenditure</th>
      </tr>
   </thead
         >
  <tbody>
     <tr>
         <td>Stationary</td>
         <td>2,000</td>
      </tr>
      <tr>
         <td>Furniture</td>
         <td>10,000</td>
     </tr>
   </tbody
         >
  <tfoot>
      <tr>
          <th>Total</th>
          <td>12,000</td>
      </tr>
  </tfoot>
</table>
```

HTML5 Image

HTML Images Syntax

In HTML, images are defined with the `<img>` tag.

The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.

The `src` attribute specifies the URL (web address) of the image:

```
<img src="url">
```

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Image</h2>
<img src="html5.gif" alt="HTML5 Icon" style="width:128px;height:128px;">
</body>
</html>
```

OUTPUT

HTML Image

HTML Form

HTML Forms are required to collect different kinds of user inputs, such as contact details likename, email address, phone numbers, or details like credit card information, etc.
Forms contain special elements called **controls** like ***inputbox, checkboxes, radio-buttons, submit buttons***, etc. Users generally complete a form by modifying its controls e.g. entering text, selectingitems, etc. and submitting this form to a web server for further processing.

The <form> tag is used to create an HTML form. Here's a simple example of a login form:

Example

```
<form>
   <label>Username: <input type="text"></label>
   <label>Password: <input type="password"></label>
   <input type="submit" value="Submit">
</form>
```

The following section describes different types of controls that you can use in your form.

Input Element

This is the most commonly used element within HTML forms.
It allows you to specify various types of user input fields, depending on the type attribute.
An inputelement can be of type *text field*, *password field*, *checkbox*, *radio button*, *submit button*, *reset button*, *file select box*, as well as several new input types introduced in HTML5.
The most frequently used input types are described below.

Text Fields

Text fields are one line areas that allow the user to input text.
Single-line text input controls are created using an <input> element, whose type attribute has a valueof text. Here's an example of a single-line text input used to take username:

Example

```
<form>
   <label for="username">Username:</label>
   <input type="text" name="username" id="username">
</form>
```

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

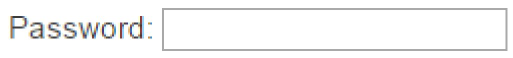
Password Field

Password fields are similar to text fields. The only difference is; characters in a password field aremasked, i.e. they are shown as asterisks or dots. This is to prevent someone else from reading the password on the screen. This is also a single-line text input controls created using
an <input> element whose type attribute has a value of password.

Example

```
<form>
   <label for="user-pwd">Password:</label>
   <input type="password" name="user-password" id="user-pwd">
</form>
```

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

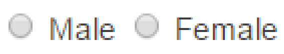


Radio Buttons
Radio buttons are used to let the user select exactly one option from a pre-defined set of options. Itis created using an `<input>` element whose `type` attribute has a value of `radio`.

Example
Try this code »

```
<form>
   <input type="radio" name="gender" id="male">
   <label for="male">Male</label>
   <input type="radio" name="gender" id="female">
   <label for="female">Female</label>
</form>
```

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

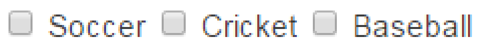


Checkboxes
Checkboxes allows the user to select one or more option from a pre-defined set of options. It iscreated using an `<input>` element whose `type` attribute has a value of `checkbox`.

Example

```
<form>
   <input type="checkbox" name="sports" id="soccer">
   <label for="soccer">Soccer</label>
   <input type="checkbox" name="sports" id="cricket">
```

```
    <label for="cricket">Cricket</label>
    <input type="checkbox" name="sports" id="baseball">
    <label for="baseball">Baseball</label>
</form>
```
— The output of the above example will look something like this:

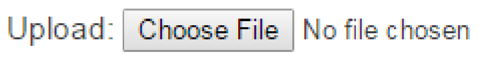


File Select box

The file fields allow a user to browse for a local file and send it as an attachment with the form data.Web browsers such as Google Chrome and Firefox render a file select input field with a Browse button that enables the user to navigate the local hard drive and select a file. This is also created using an <input> element, whose type attribute value is set to file.

Example

```
<form>
    <label for="file-select">Upload:</label>
    <input type="file" name="upload" id="file-select">
</form>
```
— The output of the above example will look something like this:

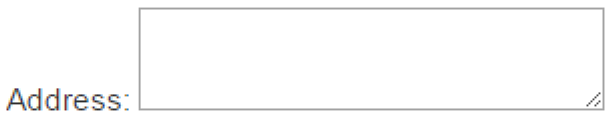


Textarea

Textarea is a multiple-line text input control that allows a user to enter more than one line of text.Multi-line text input controls are created using an <textarea> element.

Example

```
<form>
    <label for="address">Address:</label>
    <textarea rows="3" cols="30" name="address" id="address"></textarea>
</form>
```
— The output of the above example will look something like this:

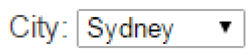


Select Boxes
A select box is a dropdown list of options that allows user to select one or more option from a pull-down list of options. Select box is created using the <select> element and <option> element.The <option> elements within the <select> element define each list item.

Example

```
<form>
   <label for="city">City:</label>
   <select name="city" id="city">
      <option value="sydney">Sydney</option>
      <option value="melbourne">Melbourne</option>
      <option value="cromwell">Cromwell</option>
   </select>
</form>
```

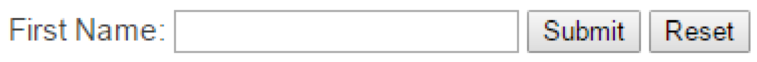
— The output of the above example will look something like this:



Submit and Reset Buttons
A submit button is used to send the form data to a web server. When submit button is clicked theform data is sent to the file specified in the form's action attribute to process the submitted data.
A reset button resets all the forms control to default values. Try out the following example by typingyour name in the text field, and click on submit button to see it in action.

Example

```
<form action="action.php" method="post">
   <label for="first-name">First Name:</label>
   <input type="text" name="first-name" id="first-name">
   <input type="submit" value="Submit">
   <input type="reset" value="Reset">
```



```
</form>
```

<u>HTML5 Colors</u>

```
<!DOCTYPE html>   <html>
<body>
<h1 style="background-color:Tomato;">Tomato</h1>
<h1 style="background-color:Orange;">Orange</h1>
<h1 style="background-color:DodgerBlue;">DodgerBlue</h1>
<h1 style="background-color:MediumSeaGreen;">MediumSeaGreen</h1>

<h1 style="background-color:Gray;">Gray</h1>
<h1 style="background-color:SlateBlue;">SlateBlue</h1>
<h1 style="background-color:Violet;">Violet</h1>
<h1 style="background-color:LightGray;">LightGray</h1>
</body>
</html>
```

OUTPUT

```
Tomato
Orange

DodgerBlue

MediumSeaGreen

Gray

SlateBlue

Violet


LightGray
```

<u>HTML5 Audio</u>

Embedding Audio in HTML Document

Inserting audio onto a web page was not easy before, because web browsers did not have a uniformstandard for defining embedded media files like audio.

Using the HTML5 audio Element

The newly introduced HTML5 `<audio>` element provides a standard way to embed audio in webpages. However, the audio element is relatively new but it works in most of the modern web browsers.

The following example simply inserts an audio into the HTML5 document, using the browserdefault set of controls, with one source defined by the `src` attribute.

Example

```
<audio controls="controls" src="media/birds.mp3">
  Your browser does not support the HTML5 Audio element.
</audio>
```

An audio, using the browser default set of controls, with alternative sources.

Example

```
<audio controls="controls">
  <source src="media/birds.mp3" type="audio/mpeg">
  <source src="media/birds.ogg" type="audio/ogg">
  Your browser does not support the HTML5 Audio element.
</audio>
```

<u>HTML5 Video</u>

Embedding Video in HTML Document

Inserting video onto a web page was not relatively easy, because web browsers did not have auniform standard for defining embedded media files like video.

Using the HTML5 video Element

The newly introduced HTML5 `<video>` element provides a standard way to embed video in webpages. However, the video element is relatively new, but it works in most of the modern web browsers.

The following example simply inserts a video into the HTML document, using the browser defaultset of controls, with one source defined by the `src` attribute.

Example

```
<video controls="controls" src="media/shuttle.mp4">
  Your browser does not support the HTML5 Video element.
</video>
```

A video, using the browser default set of controls, with alternative sources.

Example

```
<video controls="controls">
  <source src="media/shuttle.mp4" type="video/mp4">
  <source src="media/shuttle.ogv" type="video/ogg">
```

Your browser does not support the HTML5 Video element.
</video>

New HTML5 Elements

The most interesting new HTML5 elements are:

New **semantic elements** like <header>, <footer>, <article>, and <section>. New **attributes of form elements** like number, date, time, calendar, and range.
New **graphic elements**: <svg> and <canvas>.

New **multimedia elements**: <audio> and <video>.

What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: <div> and <span> - Tells nothing about its content.

Examples of **semantic** elements: <form>, <table>, and <article> - Clearly defines its content.

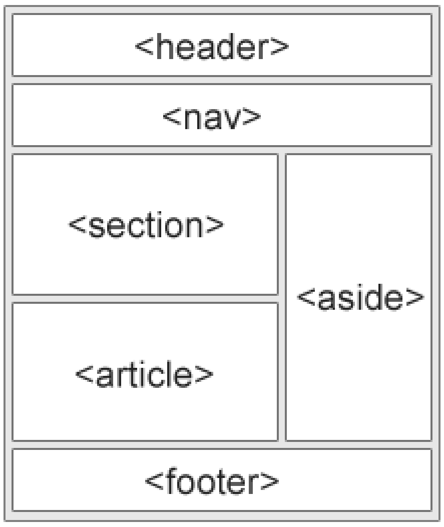
New Semantic Elements in HTML5

Many web sites contain HTML code like:

<div id="nav"> <div class="header"> <div id="footer">to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



HTML5 <section> Element

The <section> element defines a section in a document.

According to W3C's HTML5 documentation: "A section is a thematic grouping of content, typically with aheading."

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

Example
```
<section>
 <h1>WWF</h1>
 <p>The World Wide Fund for Nature (WWF) is      </p>
</section>
```
HTML5 <article> Element

The <article> element specifies independent, self-contained content.

An article should make sense on its own, and it should be possible to read it independently from the rest of theweb site.

Examples of where an <article> element can be used:

- Forum post
- Blog post
- Newspaper article

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

HTML5 <header> Element

The <header> element specifies a header for a document or section.

The <header> element should be used as a container for introductory content.

You can have several <header> elements in
one document. The following example defines
a header for an article:

Example

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

HTML5 <footer> Element

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

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

A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

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

Example

```
<footer>
  <p>Posted by: Hege Refsnes</p>
  <p>Contact information: <a
  href="mailto:someone@example.com">
  someone@example.com</a>.</p>
</footer>
```

HTML5 <figure> and <figcaption> Elements

The purpose of a figure caption is to add a visual explanation to an image.

In HTML5, an image and a caption can be grouped together in a <figure> element:

Example

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

OUTPUT

Places to Visit

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

Fig.1 - Trulli, Puglia, Italy.

Semantic Elements in HTML5

Below is an alphabetical list of the new semantic elements in HTML5. The links go to our complete HTML5 Reference.

Tag	Description
<article>	Defines an article
<aside>	Defines content aside from the page content

<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section
<header>	Specifies a header for a document or section
<main>	Specifies the main content of a document
<mark>	Defines marked/highlighted text
<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

HTML5 Drag and Drop

Drag the W3Schools image into the rectangle.

Drag and Drop
Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location. In HTML5, drag and drop is part of the standard: Any element can be draggable.

HTML Drag and Drop Example
The example below is a simple drag and drop example:

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function
 allowDrop(ev) {
 ev.preventDefault()
 ;
}

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

function drop(ev)
 {
 ev.preventDefau
 lt();
 var data = ev.dataTransfer.getData("text");
 ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true" ondragstart="drag(event)" width="336"
height="69">

</body>
</html>
```

OUTPUT
Drag the W3Schools image into the rectangle:

HTML5 <nav> Element

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

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

Example

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

What Is CSS3 And Why Is It Used?

To help build highly interactive online pages, CSS3 is invariably used due to its importance in providing greater options in the design process. When marketing products and services, web design plays a vital part; a site should be created in a manner that will draw potential customers to explore and revisit a website more often. Many **web design firm**s are developing and enhancing websites through the use of CSS3 as this is a great form of web development. This article will help define CSS3 and will point out its advantages.

Definition

The acronym CSS stands for Cascading Style Sheets which is used to augment the functionality, versatility.and efficient performance of site content. It allows for the creation of content-rich websites that do not require much weight or codes; this translates into more interactive graphics and animation, superior user- interface, and significantly more organization and rapid download time.

It is used with HTML to create content structure, with CSS3 being used to format structured content. It is responsible for font properties, colors, text alignments, graphics, background images, tables and other components. This tool provides extra capabilities such as absolute, fixed and relative positioning of various elements. The increasing popularity of CSS3 when used by **web design firm**s stimulates major browsers suchas Google Chrome, Firefox, Safari, and IE9 to adopt and embrace this programming language.

Advantages

Although CSS3 is not the only web development solution, it does allow provide greater advantages for severalreasons.

- **Customization** – A web page can be customized and alterations created in the design by simplychanging a modular file.

- **Bandwidth Requirements** – It decreases server bandwidth requirements, giving rapid download timewhen a site is accessed with desktop or hand-held devices, providing an improved user experience.
- **Consistency** – It delivers consistent and accurate positioning of navigational elements on the website.
- **Appealing** – It makes the site more appealing with adding videos and graphics easier.
- **Viewing** – It allows online videos to be viewed without the use of third-party plug-ins.
- **Visibility** – It delivers the opportunity to improve brand visibility by designing effective online pages.
- **Cost Effective** – It is cost-effective, time-saving, and supported by most browsers.

Since the introduction of CSS3, there is greater control of the presentation of content and various elements on a website; however it is not really responsible for overall design as it only specifies the structure and contentpresentation of certain web pages.

External, internal, and inline CSS styles

Cascading Style Sheets (CSS) are files with styling rules that govern how your website is presented onscreen. CSS rules can be applied to your website's HTML files in various ways. You can use
an **external stylesheet**, an **internal stylesheet**, or an **inline style**. Each method has advantages that suitparticular uses.

An **external stylesheet** is a standalone .css file that is linked from a web page. The advantage of external stylesheets is that it can be created once and the rules applied to multiple web pages. Should you need to make widespread changes to your site design, you can make a single change in the stylesheet and it will beapplied to all linked pages, saving time and effort.

An **internal stylesheet** holds CSS rules for the page in the **head** section of the HTML file. The rules only apply to that page, but you can configure CSS classes and IDs that can be used to style multiple elements inthe page code. Again, a single change to the CSS rule will apply to all tagged elements on the page.

Inline styles relate to a specific HTML tag, using a **style** attribute with a CSS rule to style a specific pageelement. They're useful for quick, permanent changes, but are less flexible than external and internal stylesheets as each inline style you create must be separately edited should you decide to make a design change.

Using external CSS stylesheets

An HTML page styled by an external CSS stylesheet must reference the .css file in the document head. Oncecreated, the CSS file must be uploaded to your server and linked in the HTML file with code such as:

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

You can name your stylesheet whatever you wish, but it should have a .css file extension.

Using internal CSS stylesheets

Rather than linking an external .css file, HTML files using an internal stylesheet include a set of rules intheir **head** section. CSS rules are wrapped in <style> tags, like this:

```
<head>
<style type="text/css">

   h1 {
         color:#fff
         margin-left: 20px;
      }

   p{
      font-family: Arial, Helvetica, Sans Serif;
     }
</style>
</head>
```

Using inline styles

Inline styles are applied directly to an element in your HTML code. They use the **style** attribute, followed byregular CSS properties.

For example:

```
<h1 style="color:red;margin-left:20px;">Today’s Update</h1>
```

Rule Cascading

Cascade and inheritance

Conflicting rules

- CSS stands for **Cascading Style Sheets**, and that first word *cascading* is incredibly important to understand

— the way that the cascade behaves is key to understanding CSS.

- At some point, we will find that the CSS have created two rules which could potentially apply to the same element.
- The **cascade**, and the closely-related concept of **specificity**, is mechanisms that control which ruleapplies when there is such a conflict.
- Which rule is styling your element may not be the one you expect, so you need to understand how these mechanisms work.
- Also significant here is the concept of **inheritance**, which means that some CSS properties by default inheritvalues set on the current element's parent element, and some don't. This can also cause some behavior that you might not expect.

The cascade

Stylesheets **cascade** — at a very simple level this means that the order of CSS rules matter; when two rulesapply that have equal specificity the one that comes last in the CSS is the one that will be used.

EXAMPLE

In the below example, we have two rules that could apply to the h1. The h1 ends up being colored blue — these rules have an identical selector and therefore carry the same specificity, so the last one in the sourceorder wins.

```
h1 {
   color: red;
   }
h1 {
   color: blue;
}
   <h1>This is my heading.</h1>
```

OUTPUT

This is my heading.

Specificity

Specificity is how the browser decides which rule applies if multiple rules have different selectors, but couldstill apply to the same element. It is basically a measure of how specific a selector's selection will be:

- An element selector is less specific — it will select all elements of that type that appear on a page — so willget a lower score.
- A class selector is more specific — it will select only the elements on a page that have a specific class attributevalue — so will get a higher score.

Example time! Below we again have two rules that could apply to the h1. The below h1 ends up being coloredred — the class selector gives its rule a higher specificity, and so it will be applied even though the rule withthe element selector appears further down in the source order.

EXAMPLE

```
main-heading {

   color: red;

}

      h1 {

   color: blue;

}
```

```
.   <h1 class="main-heading">This is my heading.</h1>
```

OUTPUT

```
This is my heading.
```

Inheritance

Inheritance also needs to be understood in this context — some CSS property values set on parent elementsare inherited by their child elements, and some aren't.

For example, if you set a color and font-family on an element, every element inside it will also be styled withthat color and font, unless you've applied different color and font values directly to them.

Some properties do not inherit — for example if you set a width of 50% on an element, all of its descendantsdo not get a width of 50% of their parent's width. If this was the case, CSS would be very frustrating to use!

```
body {
   color: blue;
}

span {
   color: black;
}
```

```
<p>As the body has been set to have a color of blue this is inherited through the
descendants.</p>

<p>We can change the color by targetting the element with a selector,
 such as this

<span>span</span>.</p>
```

OUTPUT

As the body has been set to have a color of blue this is inherited through the descendants.

We can change the color by targetting the element with a selector, such as this **span**.

BACKGROUND:

CSS provides control over the backgrounds of block-level elements. CSS can set a back- ground color or add background images to HTML5 elements.

- ***background-image Property***

The **background-image** property specifies the image URL for the image flower.png in the format url(*fileLocation*).

- ***background-position Property***

The **background-position** property places the image on the page. The keywords top, bottom, center, left and right are used individually or in combination for vertical and horizontal positioning. You can position an image using lengths by specifying the hor-izontal length followed by the vertical length.

For example, to position the image as *hori- zontally centered* (positioned at 50 percent of the distance across the screen) and 30 pixels from the top, use

background-position: 50% 30px;

- ***background-repeat Property***

✓ The **background-repeat** proper controls background image **tiling**, which place-es *multiple copies* of the image next to each other to fill the background. Here, we set the tiling to no-repeat to display only one copy of the background image.

✓ Other values in- clude repeat (the default) to tile the image *vertically and horizontally*,

✓ repeat-x to tile the image only *horizontally* or

✓ repeat-y to tile the image only *vertically*.

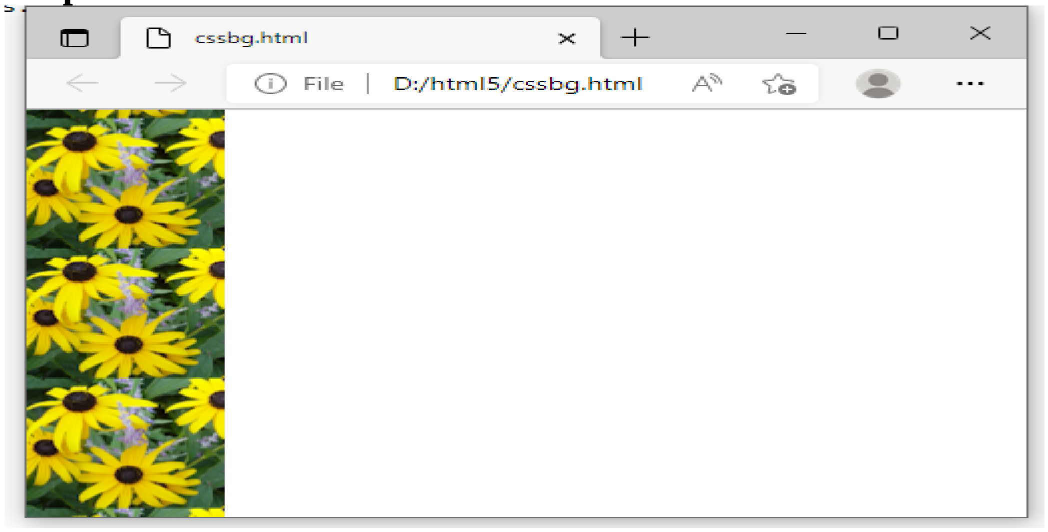
background-attachment: fixed Property

The next property setting, **background-attachment: fixed** fixes the image in the position specified by **background-position**.

Example:

```
<html>
  <head>
    <style type="text/css">
     body
      {
        background-image:url(yellowflowers.png);
        background-position:left;
        background-attachment:fixed;
        background-repeat:repeat-y;
     }
  </style>
 </head>
 <body>
 </body>
</html>
```

Output:



BORDER IMAGES:

The CSS3 **border-image property** uses images to place a border around *any* block-level element.

- Stretching an Image Border

The border-image property has six values:

border-image-source—the URL of the image to use in the border (in this case, url(border.png)).

border-image-slice—expressed with four space-separated values in pixels (in this case, 80808080). These values are the *inward offsets* from the top, right, bottom and left sides of the image. Since our original image is square, we used the same value for each.

- The border-image-slice divides the image into nine *regions*: four corners, four sides and middle, which is transparent unless other- wise specified. These regions may overlap.
- If you use values that are larger than the actual image size, the border-image-slice values will be interpreted as 100%. *You may not use negative values*.
- We could express the border-image-slice in *two* values—8080—in which case the first value would represent the top and bottom, and the second value the left and right.
- The border-image-slice may also be ex-pressed in percentages.

border-image-repeat—specifies how the regions of the border image are scaled and *tiled* (repeated). By indicating stretch just *once*, we create a border that will stretch the top, right, bottom and left regions to fit the area.

- You may specify *two* values for the border-image-repeat property.

Stretch ,repeat, the top and bottom regions of the image border would be *stretched*, and the right and left regions of the border would be repeated (i.e., *tiled*) to fit the area.

- Other possible values for the border-image-repeat property in- clude round and space.
- If you specify round, the regions are repeated using only whole tiles, and the border image is scaled to fit the area. If you specify space, the regions are repeated to fill the area using only whole tiles, and any excess space is distributed evenly around the tiles.

✓ **Repeating an Image Border**

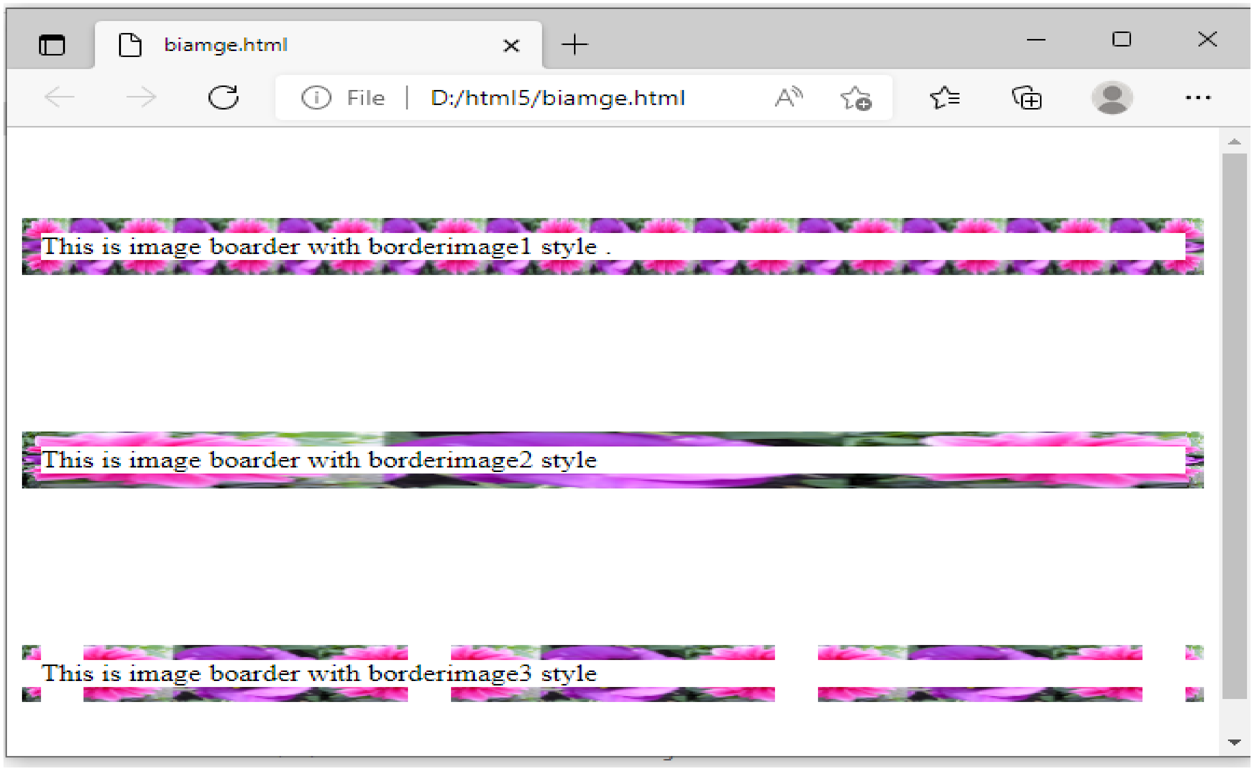
We create an image border by repeating the regions to fit the space. The border-image property

includes four values:
border-image-source—the URL of the image to use in the border (once again,url(border.png)).
border-image-slice—in this case, we provided *two* values expressed in percent-ages (34% 34%) for the top/bottom and left/right, respectively.
border-image-repeat—the value repeat specifies that the tiles are repeated to fit the area, using partial tiles to fill the excess space.

Example:

```
<html>
  <head>
    <style>
      #borderimg1 {
          border: 10px solid transparent;
        border-image-source: url(border.png);
        border-image-repeat: round;
        border-image-slice: 30;
        border-image-width: 10px;
      }
      #borderimg2 {
        border: 10px solid transparent;
        border-image-source: url(border.png);
        border-image-repeat: stretch;
        border-image-slice: 20;
        border-image-width:10px;
      }
      #borderimg3 {
        border: 10px solid transparent;
        border-image-source: url(border.png);
        border-image-repeat: space;
        border-image-slice: 30;
        border-image-width: 10px;
      }
    </style>
  </head>
  <body>
    <br> <p id = "borderimg1">This is image boarder with borderimage1 style  .</p> <br>
     <br><p id = "borderimg2">This is image boarder with borderimage2 style</p><br>
     <br><p id = "borderimg3">This is image boarder with borderimage3 style</p><br>
  </body>
</html>
```

Output:



This is image boarder with borderimage1 style .

This is image boarder with borderimage2 style

This is image boarder with borderimage3 style

COLORS:

CSS3 allows you to express color in several ways in addition to standard **color names** (suchas Aqua) or hexadecimal RGB values (such as #00FFFF for Aqua).

- ✓ **RGB** (Red, Green, Blue) or **RGBA** (Red, Green, Blue, Alpha) gives you greater control over the exact colors in yourweb pages.
    - The value for each color—red, green and blue—can range from 0 to 255. The *alpha* value—which represents *opacity*—can be any value in the range 0.0 (fully transparent) through 1.0(fully opaque). **For example**, if you were to set the backgroundcolor as follows:

background: rgba(**255**, **0**, **0**, **0.5**);

the resulting color would be a half-opaque red.

- Using RGBA colors gives you far more op- tions than using only the existing HTML color names—there are over 140 HTML colornames, whereas there are 16,777,216 different RGB colors (256 x 256 x 256) and varyingopacities of each.

- ✓ CSS3 also allows you to express color using **HSL (hue, saturation, lightness)** or **HSLA (hue, saturation, lightness, alpha)** values.
    - The ***hue*** is a color or shade expressed asa value from 0 to 359 representing the degrees on a color wheel (a wheel is 360 degrees).
    - The colors on the wheel progress in the order of the colors of the rainbow—red, orange, yellow, green, blue, indigo and violet.
    - The value for red, which is at the beginning of the wheel, is 0. Green hues have values around 120 and blue hues have values around 240. A hue value of 359, which is just left of 0 on the wheel, would result in a red hue.
    - The ***satu ration***—the intensity of the hue—is expressed as a percentage, where 100%is fully saturated(the full color) and 0% is gray**.** ***Lightness***—the intensity of light or luminance of the hue—is also expressed as a percentage.

- A lightness of 50% is the actual hue. If you *decrease* the amount of light to 0%, the color appears completely dark (black). If you *increase* the amount of light to 100%, the color appears completely light (white).

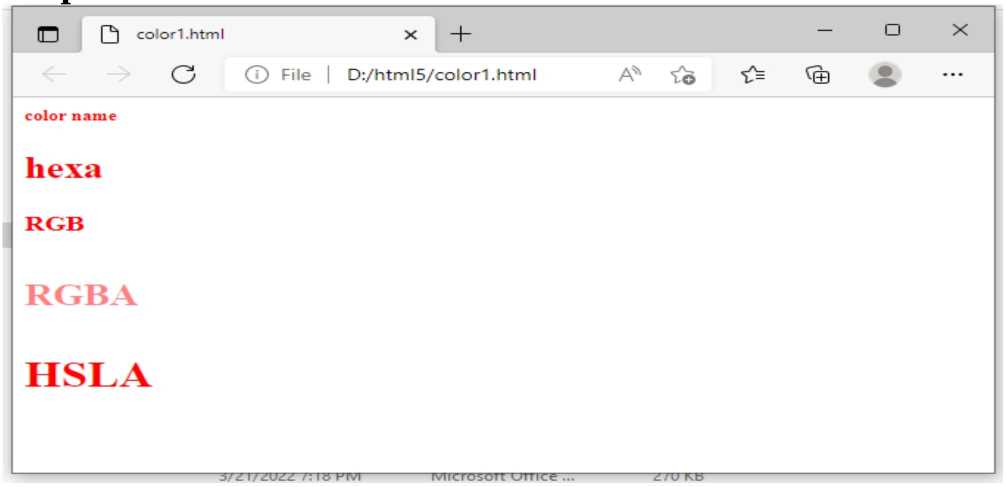
For example, if you wanted to use an hsla value to get the same color red as in our example of an rgba value, you would set the background property as follows:

```
    background: hsla(0, 100%, 50%, 0.5);
```

Example:

```
<!DOCTYPE html>
<html>
  <head>
   <style type="text/css">
     h5{color:red;}
     h2{color:FF0000;}
     h3{color:rgb(255,0,0);}
     h4{color:rgba(255,0,0,0.5);}
     h1{color:hsla(0,100%,50%,1.5);}
  </style>
  </head>
  <body>
     <h5> color name</h5>
     <h2 style="font-size:20pt"> hexa </h2>
     <h3>RGB </h3>
     <h4 style="font-size:20pt">RGBA </h4>
     <h1> HSLA</h1>
   </body>
</html>
```

Output:



```
color name
hexa
RGB
RGBA
HSLA
```

SHADOWS:

➢ **Text shadow:**

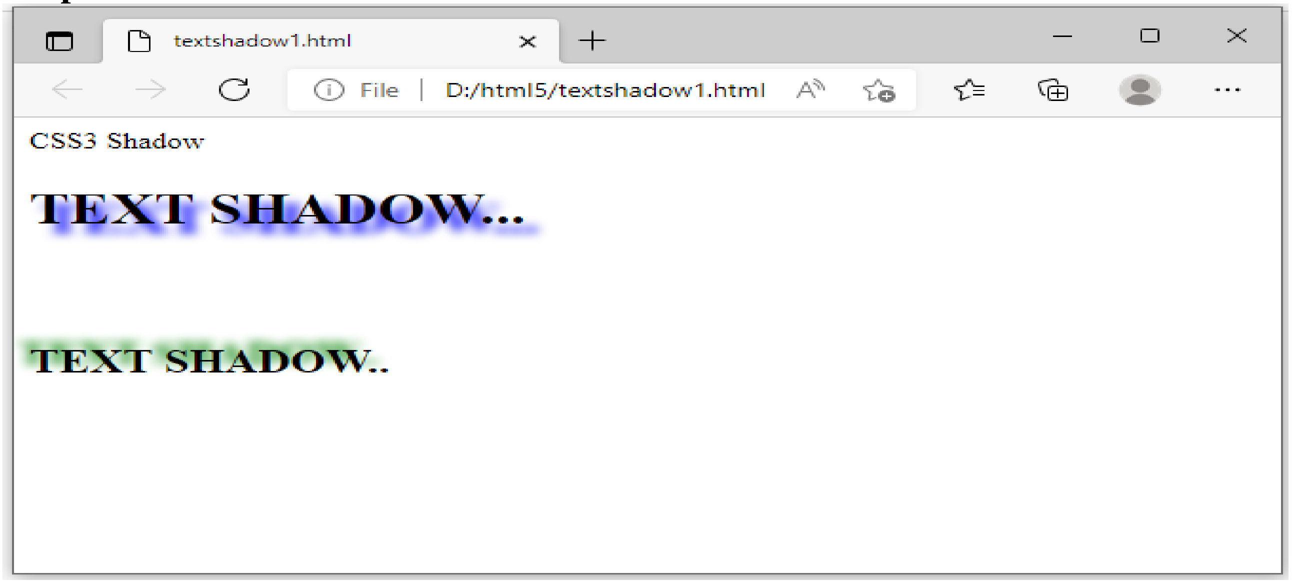
- The CSS3 **text-shadow property** makes it easy to add a **text shadow** effect to *any* text . First we add a text-shadow property to our styles . The property has four values: -4px, 4px, 6px and DimGrey, which represent:

- **Horizontal offset of the shadow**—the number of pixels that the text-shadow will appear to the *left* or the *right* of the text. In this example, the horizontal offset of the shadow is -4px. A *negative* value moves the text-shadow to the *left*; a *positive* value moves it to the *right*.
- **Vertical offset of the shadow**—the number of pixels that the text-shadow will be shifted *up* or *down* from the text. In this example, the vertical offset of the shadow is 4px. A *negative* value moves the shadow *up*, whereas a *positive* value moves it *down*.
- **blur radius**—the blur (in pixels) of the shadow. A blur-radius of 0px would result in a shadow with a sharp edge (no blur). The greater the value, the greater the blurring of the edges. We used a blur radius of 6px.
- **color**—determines the color of the text-shadow. We used dimgrey.

Example:

```
<!DOCTYPE html>
<html>
  <head>CSS3 Shadow
   <style type="text/css">
     h1
        {

         text-shadow:7px 7px 6px blue;
        }
      h2
        {
         text-shadow:-6px -6px 6px green;
        }
   </style>
 <head>
 <body>
   <h1> TEXT SHADOW...</h1> <br> <br>
   <h2>TEXT SHADOW..</h2>
  </body>
</html>
```

Output:



```
CSS3 Shadow

TEXT SHADOW...

TEXT SHADOW..
```

➢ **<u>Box shadow</u>:**

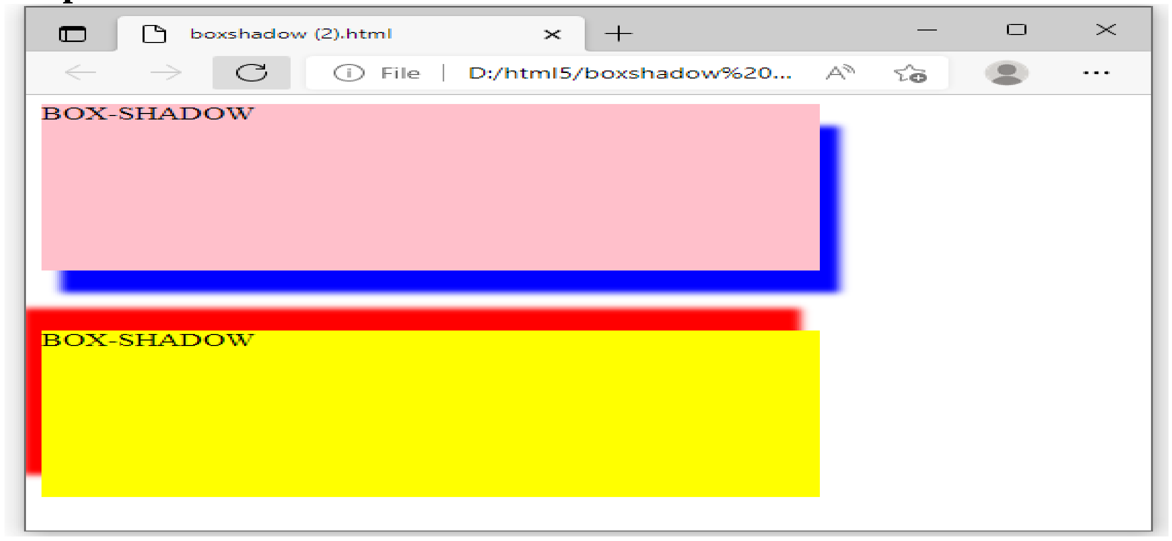
You can shadow *any* block-level element in CSS3. Next, we add the **box-shadow property** with four values :

- **Horizontal offset of the shadow (**25px)—the number of pixels that the box-shadowwill appear to the left or the right of the box. A *positive* value moves the box-shadow to the *right.A negative values moves the box-shadow to the left.*
- **Vertical offset of the shadow** (25px)—the number of pixels the box-shadowwill be shifted up or down from the box. A *positive* value moves the box-shadow*down. A negative values moves the box-shadow to the up.*
- **Blur radius**—A blur-radius of 0pxwould result in a shadow with a sharp edge (noblur). The greater the value, the more the edges of the shadow are blurred. Weused a blur radius of 10px.
- **Color**—the box-shadow's color .

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
  #bs1
  {
  width:400px;
  height:150px;
  background-color:pink;
  box-shadow:10px 20px 5px blue;
    }
  #bs2
    {
     width:400px;
    height:150px;
    background-color:yellow;
    box-shadow:-10px -20px 5px red;
     }
</style>
</head>
  <body>
   <div id="bs1"> BOX-SHADOW</div><br><br><br>
   <div id="bs2">BOX-SHADOW</div>
  </body>
</html>
```

Output:



TRANSFORMATION:

It is a property by which the object can be rotated, scaled or skewed.

The 2D transformation:

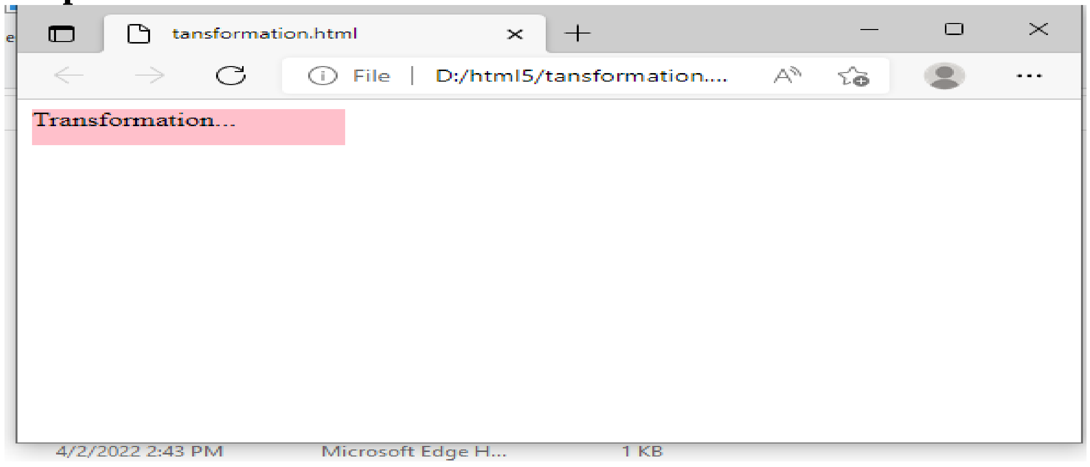
Translate Property:

- translate () : It moves an element from its current position.
  Syntax : transform: translate (30px,100px);
- rotate () : It is used to rotate the element in clockwise or anticlockwise manner to given degree.
  Syntax : transform : rotate(45deg);
- scale () : It used to increment or decrement the size of the element.
  scaleX () : Method increases or decreases the width of element.
  Syntax : scaleX(3);
  scaleY () : Method increases or decreases the height of element.
  Syntax : scaleY (4);
- skew () :Method skew the element along X and Y axis.
  Syntax : transform: skew (30deg,45deg);

Example:

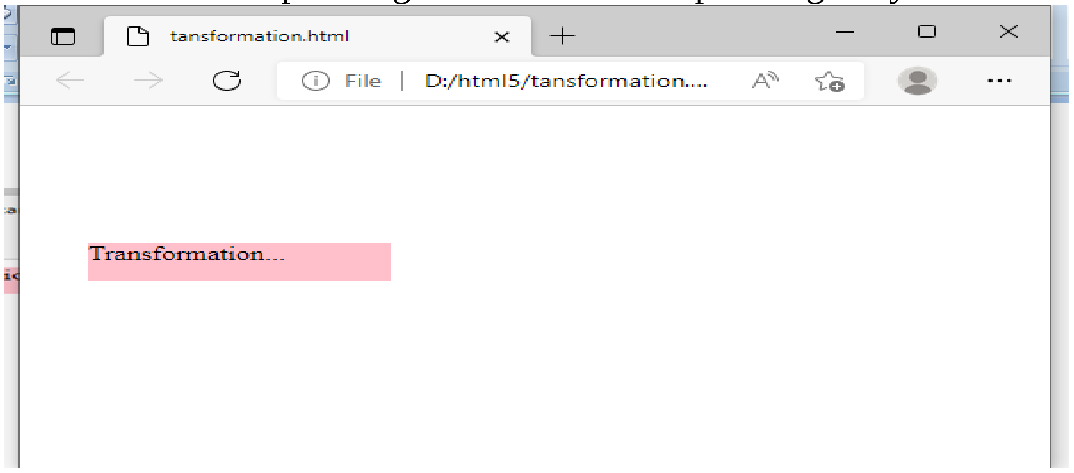
```
<!DOCTYPE html>
<html>
<head>
<style>
  div
  {
  width:170px;
  height:30px;
  background-color:pink;
     }
   div:hover
   {
    transform:translate(30px,100px);
   }
</style>
</head>
  <body>
   <div> Transformation...</div>
  </body>
</html>
```

Output:



Transformation...

-→ div is hover 30px along the x-axis and 170px along the y-axis.



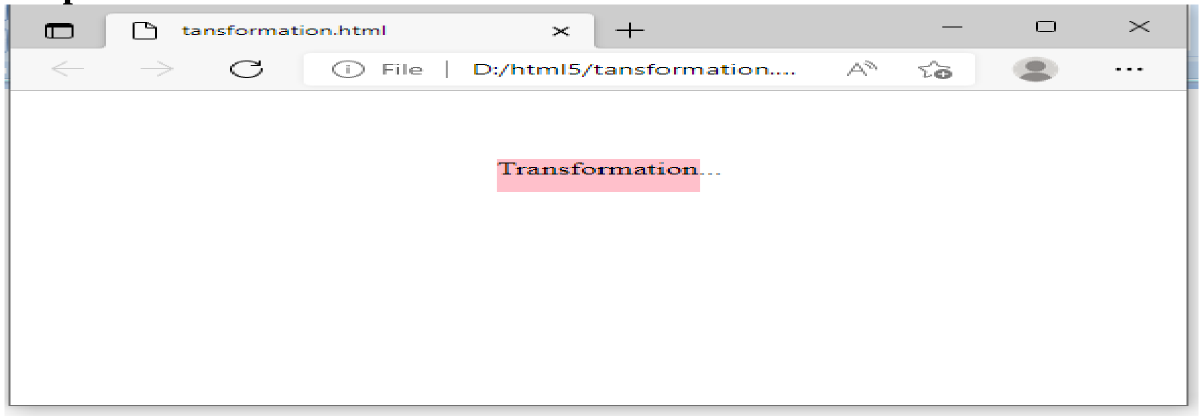
Transformation...

Example: scale()

```
<!DOCTYPE html>
<html>
<head>
<style>
  div
  {
  width:100px;
  height:30px;
  background-color:pink;
      }
```

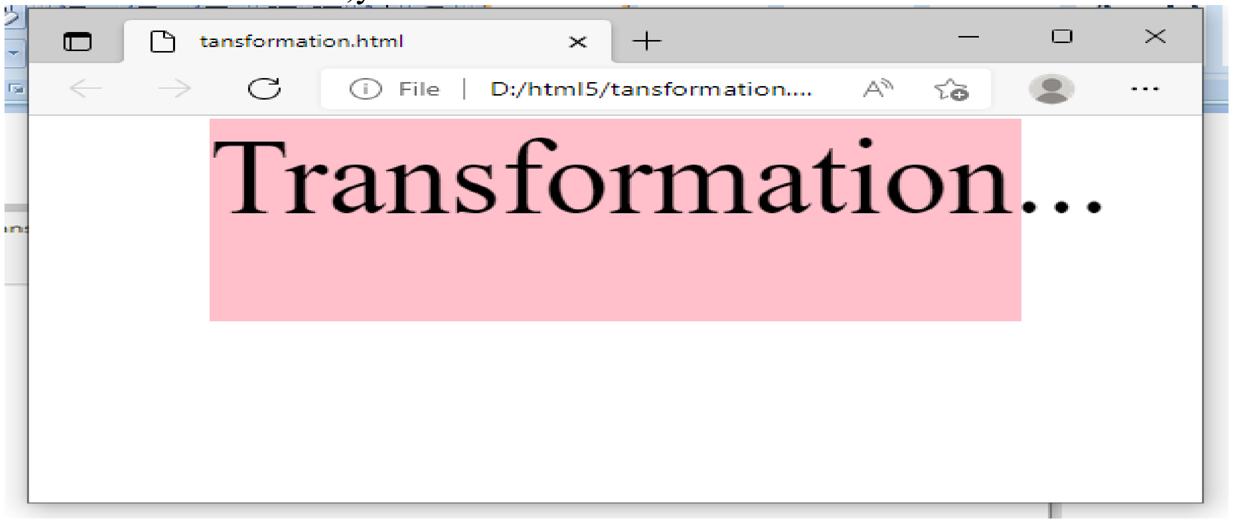
```
   div:hover
   {
    transform:scale(4,5); /*  scaled to x=4 and y=5 */
   }
</style>
</head>
  <body>
   <br> <br> <br><center>
    <div> Transformation...</div></center>
  </body>
</html>
```

Output:



Transformation...

→ div is scaled to x=4,y=5…



Transformation...

<u>TRANSITIONS:</u>

CSS Transitions is a module of CSS that lets you create gradual transitions between the values of specific CSS properties. The behavior of these transitions can be controlled by specifying their timing function, duration, and otherattributes.

Properties

- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function

transition –timing-function → Specifies the speed curve of the transition effect.

- ease – Specifies the transition effect slow start, then fast end slowly.
- linear – same speed from start to end.
- ease-in – slow start.
- ease-out – slow end.
- ease-in-out –slow start and end.

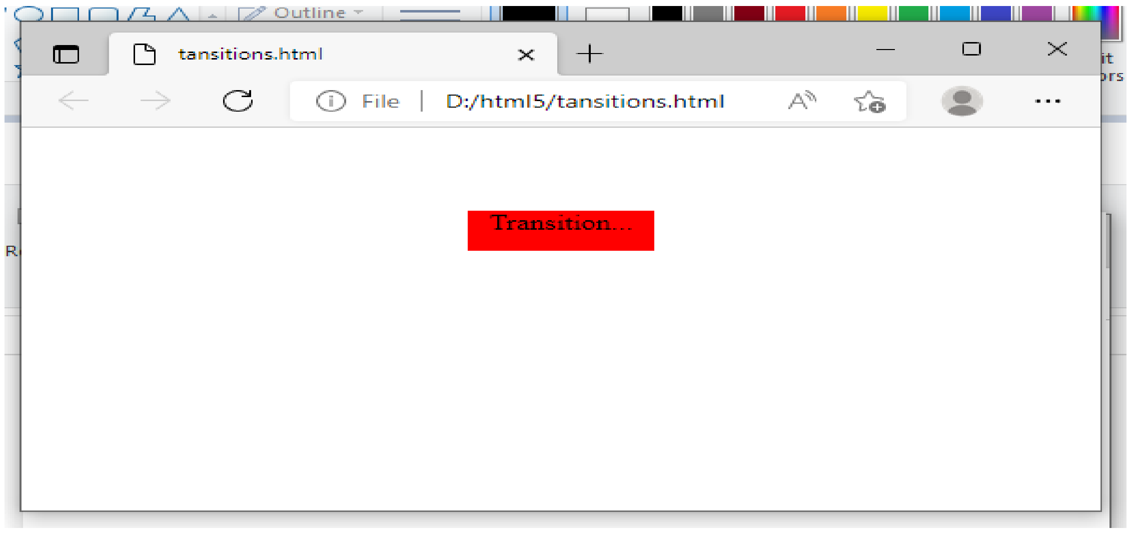
Example:

```
#div1
 { transition-timing-function: linear;}
#div2
{ transition-timing-function: ease;}
```

Example:

```
<! DOCTYPE html>
<html>
<head>
<style>
  div
  {
  width:100px;
  height:30px;
  background-color:red;
  transition:width 2s;
      }
   div:hover
   {
     width 300px;
   }
</style>
</head>
  <body>
   <br> <br> <br><center>
    <div> Transition...</div></center>
  </body>
</html>
```

OUTPUT:



ANIMATION:

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

What are CSS Animations?

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

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

CSS Animation Properties

The following table lists the @keyframes rule and all the CSS animation properties:

Property	**Description**
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties
animation-delay	Specifies a delay for the start of an animation
animation-direction	Specifies whether an animation should be played forwards,backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to completeone cycle
animation-fill-mode	Specifies a style for the element when the animation is notplaying (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused

animation-timing-function	Specifies the speed curve of the animation

Example:

```
/* The animation
code */@keyframes
example {
   from {background-
 color:red;} to {background-
 color: yellow;}
}

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

Note: The animation-duration property defines how long time an animation should take to complete. If
the animation-duration property is not specified, no animation will occur, because the default value is 0s (0seconds).

In the example above we have specified when the style will change by using the keywords "from" and "to"(which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

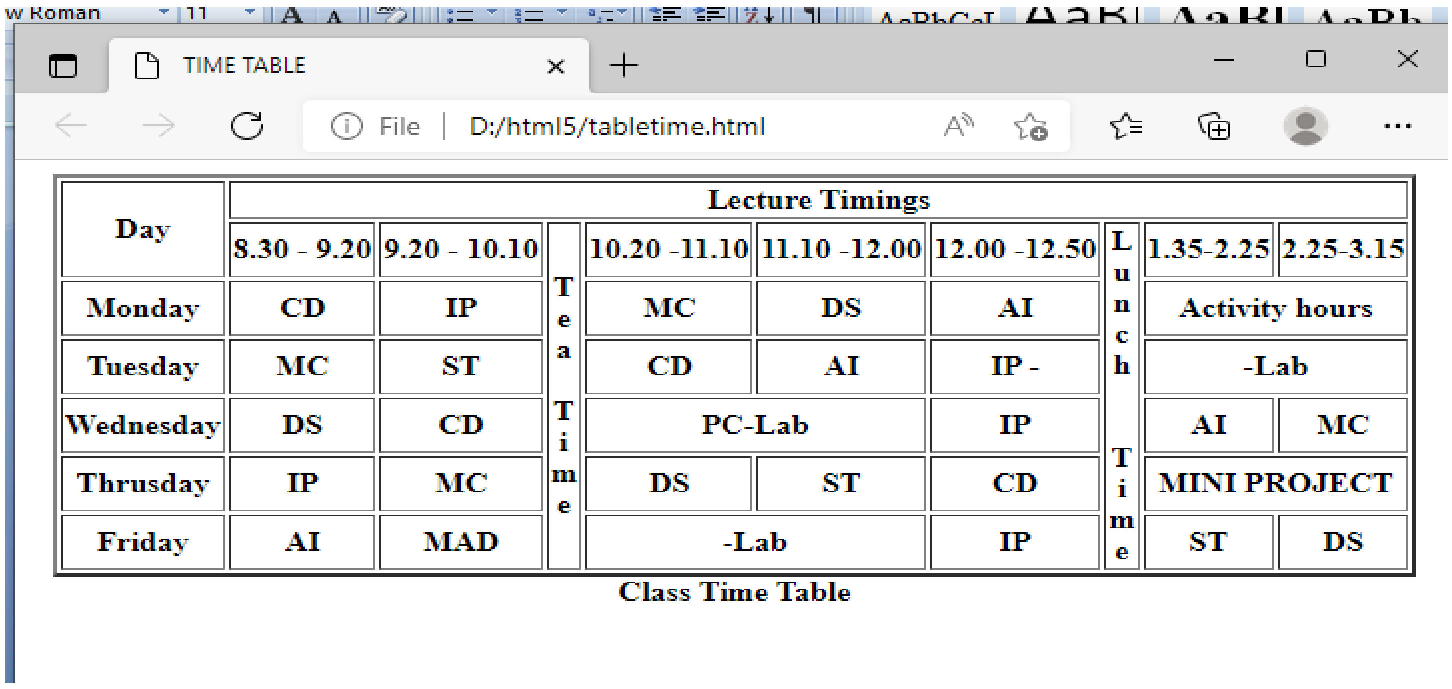
The following example will change the background-color of the <div> element when the animation is 25%complete, 50% complete, and again when the animation is 100% complete:

Example

```
/* The animation
code */@keyframes
example {
 0%  {background-color: red;}
 25%  {background-color: yellow;}
 50%  {background-color: blue;}
 100% {background-color: green;}
```

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

Example: Create the following time table using HTML tags.



Day	Lecture Timings								
	8.30 - 9.20	9.20 - 10.10		10.20 -11.10	11.10 -12.00	12.00 -12.50	Lunch Time	1.35-2.25	2.25-3.15
Monday	CD	IP	Tea Time	MC	DS	AI		Activity hours	
Tuesday	MC	ST		CD	AI	IP -		-Lab	
Wednesday	DS	CD		PC-Lab		IP		AI	MC
Thrusday	IP	MC		DS	ST	CD		MINI PROJECT	
Friday	AI	MAD		-Lab		IP		ST	DS

Class Time Table

TIME TABLE:

```
<html>
 <head>
  <title> TIME TABLE </title>
 </head>
 <body>
  <table border="2" align=center>
   <caption align=bottom>
   <b>Class Time Table</b>
   </caption>

   <tr algin=center>
    <th rowspan=2>Day</th>
    <th colspan=9>Lecture Timings</th>
   </tr>
   <tr>
     <th> 8.30 - 9.20</th>
     <th> 9.20 - 10.10</th>
    <th rowspan=6>T<br>e <br>a<br> <br>T<br> i<br>m<br> e<br>
       </th>
     <th>10.20 -11.10</th>
     <th>11.10 -12.00</th>
     <th>12.00 -12.50</th>
<th rowspan=6>L<br> u<br>n<br>c<br>h<br><br><br> T<br>i<br>m<br>e<br></th>
     <th>1.35-2.25</th>
     <th>2.25-3.15</th>
       </tr>

   <tr align=center>
     <th>Monday </th>
     <th>CD</th>
     <th>IP </th>
     <th>MC </th>
     <th>DS</th>
     <th>AI</th>
<th colspan=2> Activity hours
   </tr>

   <tr align=center>
     <th>Tuesday </th>
     <th> MC</th>
     <th>ST </th>
     <th>CD</th>
     <th>AI</th>
     <th>IP -</th>
    <th colspan=2>-Lab </th>
   </tr>

   <tr align=center>
     <th>Wednesday </th>
     <th> DS</th>
     <th>CD </th>
      <th colspan=2>PC-Lab</th>
      <th>IP</th>
      <th>AI</th>
      <th>MC</th>
   </tr>

<tr align=center>
     <th>Thrusday </th>
     <th> IP</th>
     <th>MC </th>
     <th>DS</th>
     <th>ST</th>
     <th>CD</th>
     <th colspan=2>MINI PROJECT</th>
   </tr>

<tr align=center>
     <th>Friday </th>
     <th> AI</th>
     <th> MAD</th>
     <th colspan=2>-Lab </th>
     <th>IP</th>
     <th>ST</th>
     <th>DS</th>
   </tr>
 </body>
</html>
```

UNIT-II CLIENT SIDE PROGRAMMING

An introduction to JavaScript – JavaScript DOM Model- Date and Objects - Regular Expressions- Exception Handling – Validation - Built-in objects - Event Handling - DHTML with JavaScript - JSON

INTRODUCTION

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

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

Client-side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

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

Limitations of JavaScript

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

JavaScript can be implemented using JavaScript statements that are placed within the**<script>... </script>** HTML tags in a web page.

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

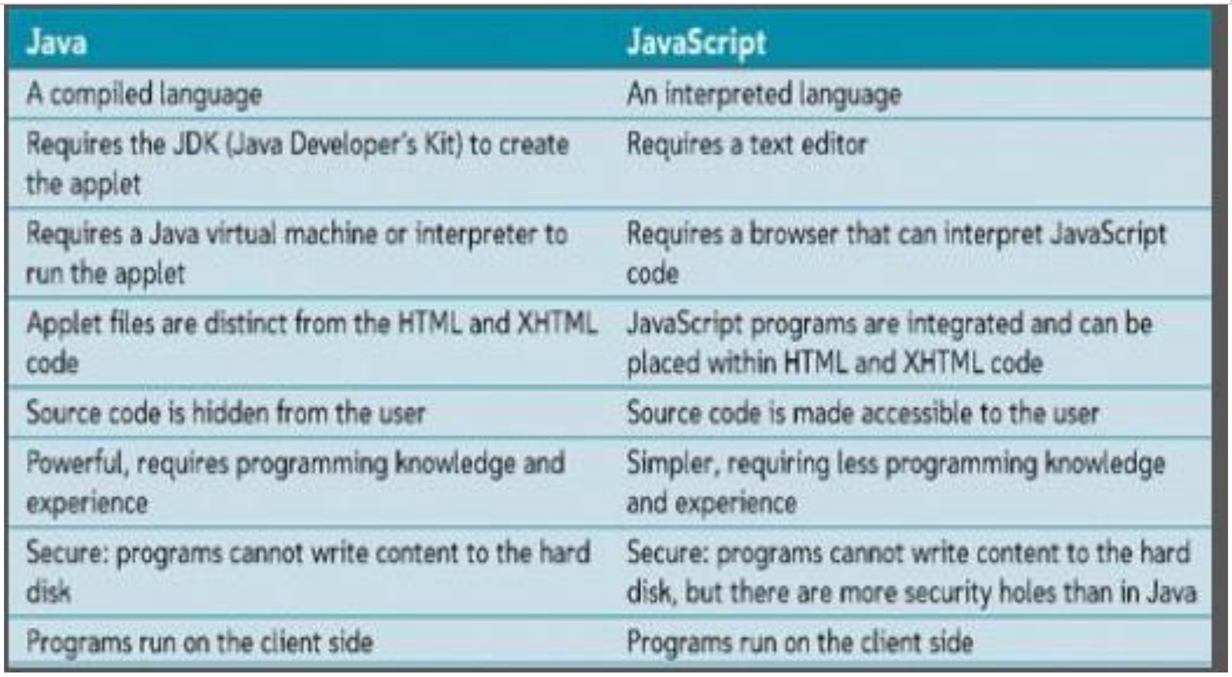
The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

The script tag takes two important attributes –

- **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like −

```
<scriptlanguage="javascript"type="text/javascript">
JavaScript code
</script>
```



Java	JavaScript
A compiled language	An interpreted language
Requires the JDK (Java Developer's Kit) to create the applet	Requires a text editor
Requires a Java virtual machine or interpreter to run the applet	Requires a browser that can interpret JavaScript code
Applet files are distinct from the HTML and XHTML code	JavaScript programs are integrated and can be placed within HTML and XHTML code
Source code is hidden from the user	Source code is made accessible to the user
Powerful, requires programming knowledge and experience	Simpler, requiring less programming knowledge and experience
Secure: programs cannot write content to the hard disk	Secure: programs cannot write content to the hard disk, but there are more security holes than in Java
Programs run on the client side	Programs run on the client side

INCLUDING JAVASCRIPT IN HTML FILE

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

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

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

JavaScript in <head>...</head> section

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

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

This code will produce the following results −

JavaScript in <body>...</body> section

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

```
         <html>

<head>
</head>

<body>

<script type="text/javascript">
<!--
```

```
document.write("Hello World")
        //-->
</script>
<p>This is web page body </p>
</body>
</html>
```

This code will produce the following results −

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows −

```
<html>
<head>
<script type="text/javascript">
<!--
FunctionsayHello() {
alert("Hello World")
           }
        //-->
</script>
</head>

<body>
<script type="text/javascript">
<!--
document.write("Hello World")
        //-->
</script>

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

</body>
</html>
```

This code will produce the following result −

JavaScript in External File

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

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

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

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

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

For example, you can keep the following content in **filename.js** file and then you can use**sayHello** function in your HTML file after including the filename.js file.

```
FunctionsayHello() {
alert("Hello World")
}
```

JAVASCRIPT OUTPUT

JavaScript does NOT have any built-in print or display functions.

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an alert box, using **window.alert()**.
- Writing into the HTML output using **document.write()**.
- Writing into an HTML element, using **innerHTML**.
- Writing into the browser console, using **console.log()**.

Using window.alert()

You can use an alert box to display data:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body></html>
```

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using console.log()

In your browser, you can use the **console.log()** method to display data.
Activate the browser console with F12, and select "Console" in the menu.
Example

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

SYNTAX OF JAVASCRIPT

JavaScript **syntax** is the set of rules, how JavaScript programs are constructed.
A **computer program** is a list of "instructions" to be "executed" by the computer.
In a programming language, these program instructions are called **statements**.JavaScript is a **programming language**.
JavaScript statements are separated by **semicolons**.

```
var x = 5;
var y = 6;
var z = x + y;
```

Statements

JavaScript statements are composed of:
Values, Operators, Expressions, Keywords, and Comments.
Values

The JavaScript syntax defines two types of values: Fixed values and variable values.
Fixed values are called **literals**. Variable values are called **variables**.

Variables

In a programming language, **variables** are used to **store** data values.
JavaScript uses the **var** keyword to **define** variables.
An **equal sign** is used to **assign values** to variables.
In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
var x;

x = 6;
```

Operators

JavaScript uses an **assignment operator** (=) to **assign** values to variables:

```
var x = 5;
var y = 6;
```

JavaScript uses **arithmetic operators** (+ - * /) to **compute** values:

```
        (5 + 6) * 10
```

Expressions

An expression is a combination of values, variables, and operators, which computes to a value.
The computation is called an evaluation.
For example, 5 * 10 evaluates to 50:

```
        5 * 10
```

Expressions can also contain variable values:

```
        x * 10
```

The values can be of various types, such as numbers and strings.
For example, "John" + " " + "Doe", evaluates to "John Doe":

```
        "John" + " " + "Doe"
```

Keywords

Keywords are used to identify actions to be performed.The **var** keyword tells the browser to create a new variable:

```
var x = 5 + 6;
var y = x * 10;
```

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

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

- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. 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 //-->.

Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, an underscore (_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

FUNCTIONS

- A function is a block of code designed to perform a particular task.
- 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.
- A JavaScript function is executed when "something" invokes it (calls it).

Syntax

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

```
function name(parameter1, parameter2, parameter3) {
  code to be executed
}
```

Function **parameters** are the **names** listed in the function definition.Function **arguments** are the real **values** received by the function when it is invoked.Inside the function, the arguments are used as local variables.

```
function myFunction(p1, p2) {
  return p1 * p2;     // The function returns the product of p1 and p2
}
```

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

```
<html><head>
<script type="text/javascript">
FunctionsayHello(name, age)
        {
document.write (name + " is " + age + " years old.");
        }
</script>

</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

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.

```
<html>
<head>
<script type="text/javascript">
function concatenate(first, last)
        {
var full;
full = first + last;
return full;
        }

functionsecondFunction()
        {
var result;
result = concatenate('Zara', 'Ali');
document.write (result );
        }
</script>
</head>
<body>
```

```
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
<form>

<p>Use different parameters inside the function and then try...</p>

</body>
</html>
```

Nested Functions

Function inside another function is known as Nested Functions.

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
function square(x) { return x*x; }
returnMath.sqrt(square(a) + square(b));
        }
functionsecondFunction(){
var result;
result = hypotenuse(1,2);
document.write( result );
        }
      //-->
</script>
</head>
<body>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
</body>
</html>
```

Function Constructors

The *function* statement is not the only way to define a new function; you can define your function dynamically using **Function()** constructor along with the **new** operator.

Note − Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

Syntax

Following is the syntax to create a function using **Function( )** constructor along with the**new** operator.

```
<script type="text/javascript">
varvariablename = new Function(Arg1, Arg2..., "Function Body");
</script>
```

The **Function()** constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the **Function()** constructor is not passed any argument that specifies a name for the function it creates. The **unnamed** functions created with the **Function()** constructor are called **anonymous** functions.

```
<html>
<head>
<script type="text/javascript">
<!--
varfunc = new Function("x", "y", "return x*y;");
functionsecondFunction(){
var result;
result = func(10,20);
document.write( result );
          }
        //-->
</script>
</head>
<body>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form></body></html>
```

Function literals

Function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

The syntax for a **function literal** is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.

```
<script type="text/javascript">
varvariablename = function(Argument List){
        Function Body
     };
```

</script>

Syntactically, you can specify a function name while creating a literal function as follows.

```
<script type="text/javascript">
varvariablename = function FunctionName(Argument List){
        Function Body
      };
</script>
```

But this name does not have any significance, so it is not worthwhile.

```
<html>
<head>
<script type="text/javascript">
varfunc = function(x,y){ return x*y };
functionsecondFunction(){
var result;
result = func(10,20);
document.write( result );
           }
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
</body></html>
```

OBJECTS

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

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

All cars have the same **properties**, but the property values differ from car to car.

All cars have the same **methods**, but the methods are performed at different times.

```
        var car = "Fiat";
```

Objects are variables too. But objects can contain many values.

```
        var car = {type:"Fiat", model:500, color:"white"};
```

Properties

The name:values pairs (in JavaScript objects) are called **properties**.

```
var person = {firstName:"XYZ", lastName:"ABC", age:20};
```

Example

```
<!DOCTYPE html><html>
<body>
<p id="demo"></p>
<script>
var person = {firstName:"XYZ", lastName:"ABC", age:20};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body></html>
```

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName[propertyName]

```
<!DOCTYPE html>
<html><body>
<p id="demo1"></p>
<p id="demo"></p>
<script>
var person = {
firstName: "XYZ",
lastName : "ABC",
id       :  5566
};
document.getElementById("demo").innerHTML =person.firs
person.lastName;
document.getElementById("demo1").innerHTML =person["firstName"] + " "
+ person.lastName;
</script>
</body></html>
```

OUTPUT

```
XYZ ABC

XYZ ABC
```

Methods

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

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

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

User-Defined Objects

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

The new Operator

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

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

The Object() Constructor

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

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

```
<html><head>
<title>User-defined objects</title>
<script type="text/javascript">
var book = new Object();   // Create the object
book.subject = "C++"; // Assign properties to the object
book.author  = "Ira Pohl";
</script>
</head>
<body>
<script type="text/javascript">
document.write("Book name is : " + book.subject + "<br>");
document.write("Book author is : " + book.author + "<br>");
</script>
</body></html>
```

OUTPUT:

```
Book name is : C++

Book author is : Ira Pohl
```

Example 2:

```
<html><head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
this.title = title;
this.author  = author;
        }
</script>
</head>
<body>
<script type="text/javascript">
varmyBook = new book("C++", "Ira Pohl");
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
</script>
</body></html>
```

OUTPUT:

```
Book name is : C++

Book author is : Ira Pohl
```

Defining Methods for an Object

```
<html>
<head>
<title>User-defined objects</title>

<script type="text/javascript">
        // Define a function which will work as a method
FunctionaddPrice(amount){
this.price = amount;
        }
function book(title, author){
this.title = title;
this.author  = author;
this.addPrice = addPrice; // Assign that method as property.
        }
</script>
</head>
<body>
<script type="text/javascript">
varmyBook = new book("C++", "Ira Pohl");
myBook.addPrice(100);
document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
</script></body></html>
```

OUTPUT:

```
Book name is : C++

Book author is : Ira Pohl

Book priceis : 100
```

NUMBER OBJECT

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

Syntax

The syntax for creating a **number** object is as follows −

```
    varval = 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).

Converting Variables to Numbers

There are 3 JavaScript functions that can be used to convert variables to numbers:

- ***The Number() method***
- ***The parseInt() method***
- ***The parseFloat() method***

These methods are not **number** methods, but **global** JavaScript methods.

The Number() Method - can be used to convert JavaScript variables to numbers:

```
x = true;
Number(x);        // returns 1
x = false;
Number(x);        // returns 0
x = new Date();
Number(x);        // returns 1404568027739
x = "10"
Number(x);        // returns 10
x = "10 20"
Number(x);        // returns NaN
```

The parseInt() Method - parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

```
parseInt("10");         // returns 10
parseInt("10.33");      // returns 10
parseInt("10 20 30");   // returns 10
parseInt("10 years");   // returns 10
parseInt("years 10");   // returns NaN
```

The parseFloat() Method - parses a string and returns a number. Spaces are allowed. Only the first number is returned:

```
parseFloat("10");        // returns 10
parseFloat("10.33");     // returns 10.33
parseFloat("10 20 30");  // returns 10
parseFloat("10 years");  // returns 10
parseFloat("years 10");  // returns NaN
```

The valueOf() Method - returns a number as a number.

```
var x = 123;
x.valueOf();          // returns 123 from variable x
(123).valueOf();       // returns 123 from literal 123
(100 + 23).valueOf();   // returns 123 from expression 100 + 23
```

Global Methods

JavaScript global functions can be used on all JavaScript data types.
These are the most relevant methods, when working with numbers:

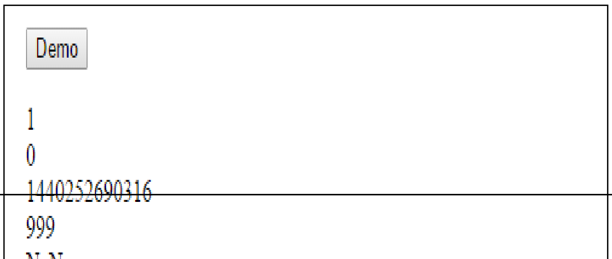
Method	Description
Number()	Returns a number, converted from its
parseFloat()	Parses its argument and returns a floating point
parseInt()	Parses its argument and returns an integer

Number Methods

JavaScript number methods are methods that can be used on numbers:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a string, with a number rounded and written using
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length
valueOf()	Returns a number as a number

```
<!DOCTYPE html><html>
<body>
<button onclick="myFunction()">Demo</button>
<p id="demo"></p>
<script>
FunctionmyFunction() {
var x1 = true;
var x2 = false;
var x3 = new Date();
var x4 = "999";
var x5 = "999 888";
```



```
1
0
1440252690316
999
```

```
var x6 = parseInt("40 years");
var x7 = parseFloat("36.00");
var x8 = isNaN("x6");
var n =
   Number(x1) + "<br>" +
   Number(x2) + "<br>" +
   Number(x3) + "<br>" +
   Number(x4) + "<br>" +
   Number(x5)+ "<br>"+
x6 +"<br>" + x7 + "<br>" + x8 + "<br>" + String(x3);
document.getElementById("demo").innerHTML = n;
}
</script>
</body></html>
```

BOOLEAN OBJECT

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.

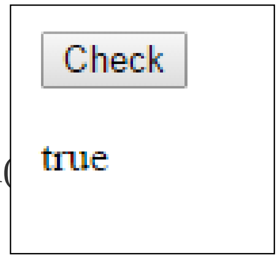
Syntax

Use the following syntax to create a **boolean** object.

```
varval = new Boolean(value);
```

Method	Description
toSource()	Returns a string containing the source of the Boolean object;
toString()	Returns a string of either "true" or "false" depending upon the value of the object.
valueOf()	Returns the primitive value of the Boolean object.

```
<!DOCTYPE html>
<html><body>
<button onclick="myFunction()">Check</button>
<p id="demo"></p>
<script>
functionmyFunction()
 {
document.getElementById("demo").innerHTML = Boolean(
}
</script>
```



true

</body></html>

ARRAY OBJECT

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

Syntax

```
        var fruits = new Array( "apple", "orange", "mango" );
        OR
         var fruits = [ "apple", "orange", "mango" ]; //easiest method  - literal
```

You will use ordinal numbers to access and to set values inside an array as follows.

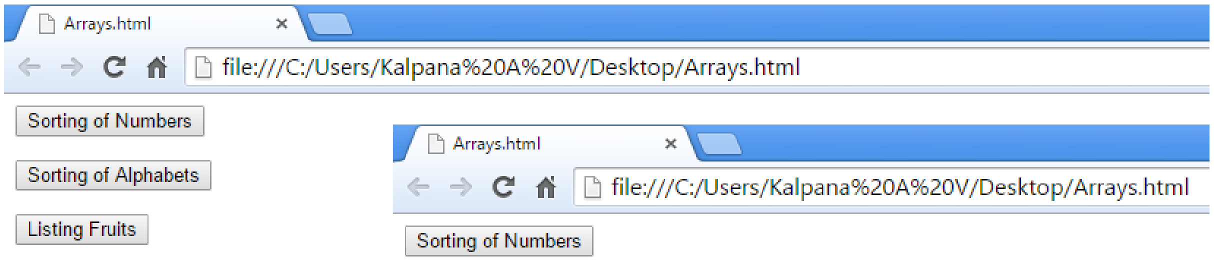
```
         fruits[0] is the first element
         fruits[1] is the second element
fruits[2] is the third element
```

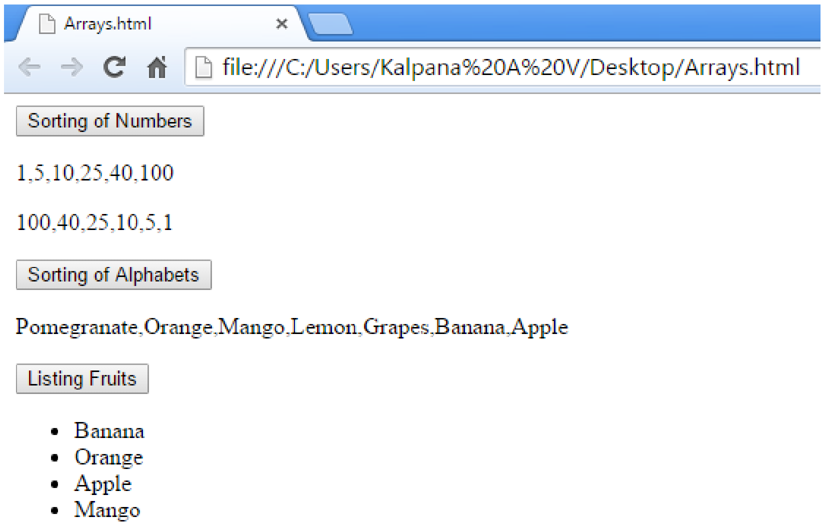
Method	**Description**
concat()	Joins two or more arrays, and returns a copy of the joined arrays
indexOf()	Search the array for an element and returns its position
join()	Joins all elements of an array into a string
lastIndexOf()	Search the array for an element, starting at the end, and returns its
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the

valueOf()	Returns the primitive value of an array

Example:

```
<!DOCTYPE html><html>
<body>
<button onclick="myNumber()">Sorting of Numbers</button>
<p id="demo"></p>
<p id="demo1"></p>
<button onclick="myFruits()">Sorting of Alphabets</button>
<p id="demo2"></p>
<button onclick="myList()">Listing Fruits</button>
<p id="demo3"></p>

<script>
functionmyFruits() {
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon"); //to add an element to the array
fruits[fruits.length] = "Pomegranate"
fruits[6]="Grapes"
document.getElementById("demo2").innerHTML = fruits;
fruits.sort();
fruits.reverse();
document.getElementById("demo2").innerHTML = fruits;
}

functionmyNumber() {
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
document.getElementById("demo").innerHTML = points;
points.sort(function(a, b){return b-a});
document.getElementById("demo1").innerHTML = points;

}

functionmyList()
{
var index;
var text = "<ul>";
var fruits = ["Banana", "Orange", "Apple", "Mango"];
for (index = 0; index <fruits.length; index++) {
text += "<li>" + fruits[index] + "</li>";
    }
text += "</ul>";
document.getElementById("demo3").innerHTML = text;
}
</script>
</body></html>
```





```
1,5,10,25,40,100

100,40,25,10,5,1

Pomegranate,Orange,Mango,Lemon,Grapes,Banana,Apple

• Banana
• Orange
• Apple
• Mango
```

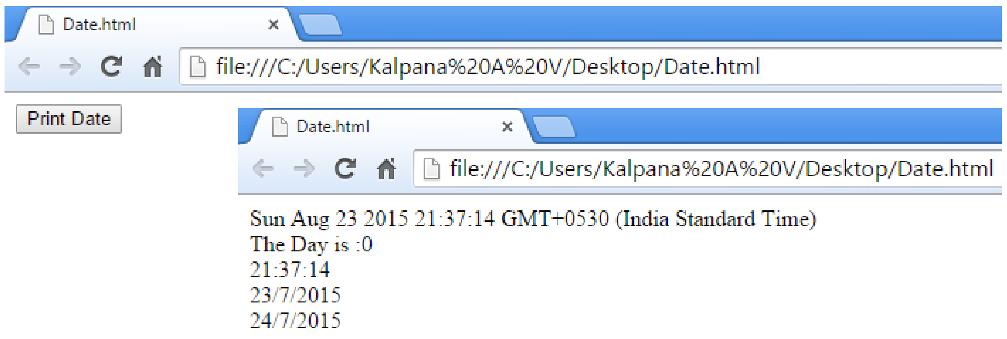
DATE OBJECT

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)

getTime()	Returns the number of milliseconds since midnight Jan
getTimezoneOffset()	Returns the time difference between UTC time and local
getUTCDate()	Returns the day of the month, according to universal
getUTCDay()	Returns the day of the week, according to universal time
parse()	Parses a date string and returns the number of
setDate()	Sets the day of the month of a date object
setFullYear()	Sets the year (four digits) of a date object
setHours()	Sets the hour of a date object
setMilliseconds()	Sets the milliseconds of a date object
setMinutes()	Set the minutes of a date object
setMonth()	Sets the month of a date object
setSeconds()	Sets the seconds of a date object
setTime()	Sets a date to a specified number of milliseconds after/before January 1, 1970
setUTCDate()	Sets the day of the month of a date object, according to
toDateString()	Converts the date portion of a Date object into a
toString()	Converts a Date object to a string
UTC()	Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time
valueOf()	Returns the primitive value of a Date object

```
<!DOCTYPE html><html>
<body>
<button onclick="myFunction()">Print Date</button>
<script>
FunctionmyFunction() {
```

```
var d = new Date();
var m = d.getMonth();
var day = d.getDay();
document.write(d + "<br> ");
document.write("The Day is :"+ day + "<br>");
document.write(d.getHours()+":" +d.getMinutes()+":"+d.getSeconds()+"<br>");
document.write(d.getDate() +"/" +d.getMonth()+"/"+d.getFullYear()+"<br>");
d.setDate(23);
document.write(d.getDate() +"/" +d.getMonth()+"/"+d.getFullYear()+"<br>");
}
</script></body></html>
```


Date.html
file:///C:/Users/Kalpana%20A%20V/Desktop/Date.html
Print Date
Date.html
file:///C:/Users/Kalpana%20A%20V/Desktop/Date.html

```
Sun Aug 23 2015 21:37:14 GMT+0530 (India Standard Time)
The Day is :0
21:37:14
23/7/2015
24/7/2015
```

MATH OBJECT

The Math object allows you to perform mathematical tasks on numbers.

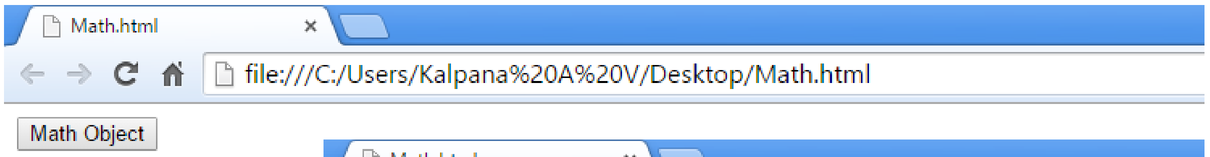
Constants

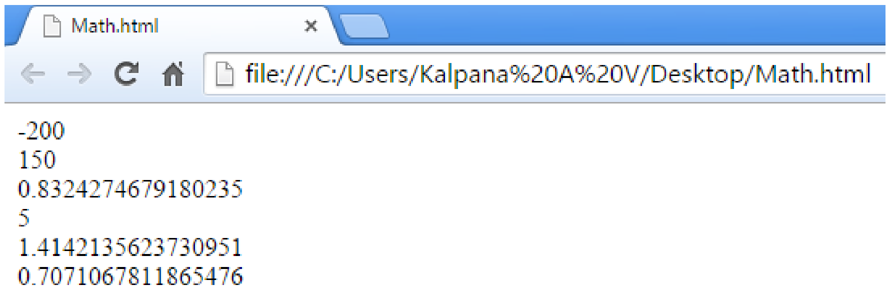
```
Math.E         // returns Euler's number
Math.PI        // returns PI
Math.SQRT2     // returns the square root of 2
Math.SQRT1_2   // returns the square root of 1/2
Math.LN2       // returns the natural logarithm of 2
Math.LN10      // returns the natural logarithm of 10
Math.LOG2E     // returns base 2 logarithm of E
Math.LOG10E    // returns base 10 logarithm of E
```

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians

atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

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

<button onclick="myFunction()">Math Object</button>

<script>
FunctionmyFunction() {
document.write(Math.min(0, 150, 30, 20, -8, -200));
document.write("<br>");
document.write(Math.max(0, 150, 30, 20, -8, -200));
document.write("<br>");
document.write(Math.random());
document.write("<br>");
document.write(Math.round(4.7888));
document.write("<br>");
document.write(Math.SQRT2);
document.write("<br>");
document.write(Math.SQRT1_2);
document.write("<br>");
}
</script>
</body></html>
```





```
-200
150
0.8324274679180235
5
1.4142135623730951
0.7071067811865476
```

STRING OBJECTS

A JavaScript string simply stores a series of characters like "John Doe".
A string can be any text inside quotes. You can use single or double quotes:

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

String Length

The length of a string is found in the built in property **length**:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Special Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
        var y = "We are the so-called "Vikings" from the north."
```

The string will be chopped to "We are the so-called ".
The solution to avoid this problem, is to use the \ **escape character**.
The backslash escape character turns special characters into string characters:

```
var x = 'It\'s alright';
var y = "We are the so-called \"Vikings\" from the north."
```

The escape character (\) can also be used to insert other special characters in a string.
This is the list of special characters that can be added to a text string with the backslash sign:

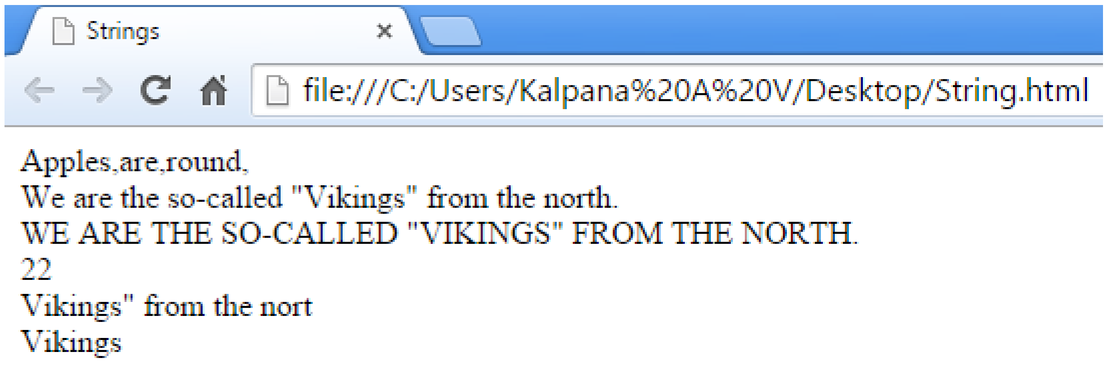
Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

String Methods

Method	**Description**
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified
lastIndexOf()	Returns the position of the last found occurrence of a specified value
localeCompar	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and
replace()	Searches a string for a value and returns a new string with the value
search()	Searches a string for a value and returns the position of the match

slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position through a number of
substring()	Extracts a part of a string between two specified positions
toLocaleLowe	Converts a string to lowercase letters, according to the host's locale
toLocaleUpper	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

```
<html>
<head>
<title>Strings</title>
</head>
<body>
<script type="text/javascript">
varstr = "Apples are round, and apples are juicy.";
varsplitted = str.split(" ", 3);
var y = "We are the so-called \"Vikings\" from the north.";
document.write(splitted );
document.write("<br>");
document.write(y);
document.write("<br>");
document.write(y.toUpperCase());
document.write("<br>");
document.write(y.search('V'));
document.write("<br>");
document.write(y.slice(22,-2));
document.write("<br>");
document.write(y.substring(22,29));
</script></body></html>
```



```
Apples,are,round,
We are the so-called "Vikings" from the north.
WE ARE THE SO-CALLED "VIKINGS" FROM THE NORTH.
22
Vikings" from the nort
Vikings
```

REGULAR EXPRESSIONS

- A regular expression is a sequence of characters that forms a **search pattern**.
- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

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

var pattern = /*pattern*/attibutes;
OR
var pattern = new RegExp(pattern, attributes);

Here is the description of the parameters –

- **pattern** − A string that specifies the pattern of the regular expression or another regular expression.
- **attributes** − An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.

[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from **b** through **v**.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, and $ flags all follow a character sequence.

Expression	**Description**
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing one or more p's.
p{**N**}	It matches any string containing a sequence of **N** p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples

Following examples explain more about matching characters.

Expression	**Description**
[^a-zA-Z]	It matches any string not containing any of the characters ranging from **a**through **z** and **A** through Z.
p.p	It matches any string containing **p,** followed by any character, in turn followed by another **p**.
^.{2}$	It matches any string containing exactly two characters.
<b>(.*)</b>	It matches any string enclosed within <b> and </b>.

p(hp)*	It matches any string containing a **p** followed by zero or more instances of the sequence **hp**.

Literal characters

Character	**Description**
Alphanumeric	Itself
\0	The NUL character (\u0000)
\t	Tab (\u0009)
\n	Newline (\u000A)
\v	Vertical tab (\u000B)
\f	Form feed (\u000C)
\r	Carriage return (\u000D)
\xnn	The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n
\uxxxx	The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t
\cX	The control character ^X; for example, \cJ is equivalent to the newline character \n

Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for a large sum of money using the '\d' metacharacter:**/([\d]+)000/**, Here **\d** will search for any string of numerical character.

The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

Character	**Description**
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character

\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[\b]	a literal backspace (special case).
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified

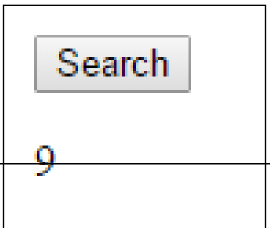
Modifiers
Several modifiers are available that can simplify the way you work with **regexps,** like case sensitivity, searching in multiple lines, etc.

Modifier	**Description**
i	Perform case-insensitive matching.
M	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of
G	Performs a global matchthat is, find all matches rather than stopping after the first match.

Search & Replace
In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().
The search() method uses an expression to search for a match, and returns the position of the match.
The replace() method returns a modified string where the pattern is replaced.

Using String search() With a Regular Expression

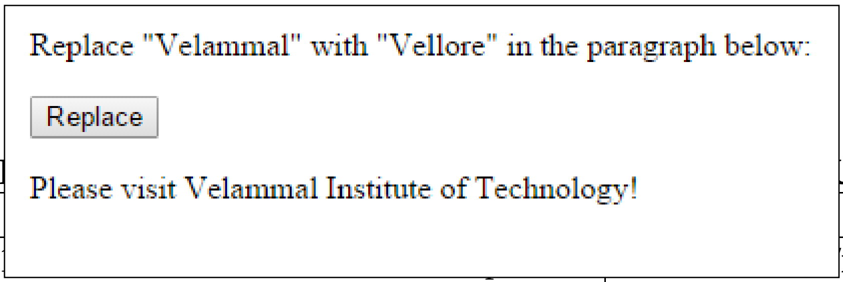
```
<!DOCTYPE html>
<html>
<body>
<button onclick="myFunction()">Search</button>
```



```
<p id="demo"></p>
<script>
FunctionmyFunction() {
varstr = "Velammal Institute of Technology";
var n = str.search(/Institute/i);
document.getElementById("demo").innerHTML = n;
}
</script>
</body></html>
```

Using String replace() With a Regular Expression

```
<!DOCTYPE html>
<html>
<body>
<p>Replace "velammal" with "Vellore" in the paragraph below:</p>
<button onclick="myFunction()">Replace</button>

<p id="demo">Please visit Velammal Institute of Technology!</p>
<script>
FunctionmyFunction()
{
varstr = document.getElementById("demo").innerHTML;
var txt = str.replace(/velammal/i,"Vellore");
document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```



Replace "Velammal" with "Vellore" in the paragraph below:

Please visit Velammal Institute of Technology!

ULAR EXPRESSIONS

	Pattern
numeric characters)	i
2. Email ID	**/^[a-z0-9._-]+@[a-z]+.[a-z.]{2,5}$/i**
3. Date of Birth	**/^[0-9]{1,2}-[0-9]{1,2}-[0-9]{4}$/i**
4. Mobile Number	**/^([+0-9]{1,3})?([0-9]{10,11})$/i**

5. Web Site URL	**/^[http://]+[www]?.[0-9a-z_.]+.[a-z.]{2,5}$/i**
6. Pincode	**/^[0-9]{6}$/i**

DOCUMENT OBJECT MODEL (THE DOM MODEL)

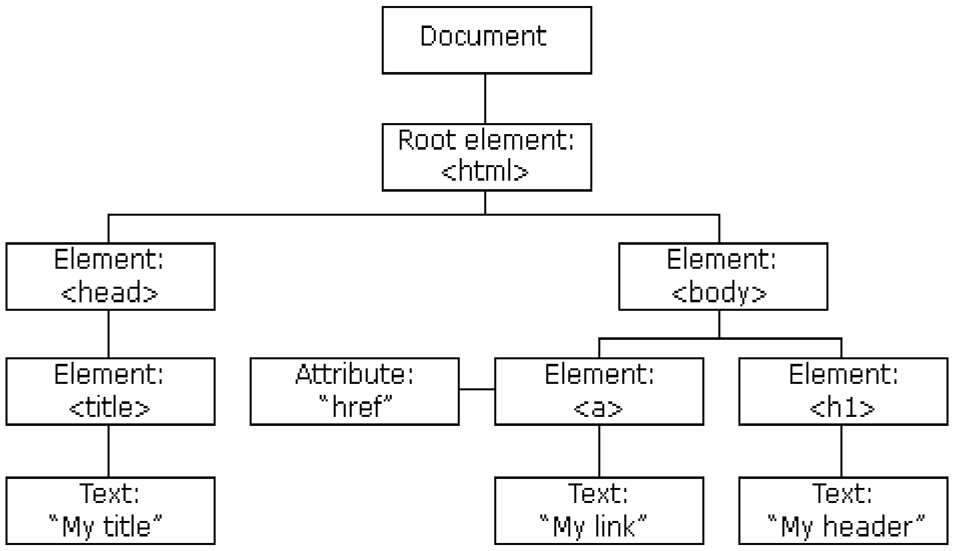
When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

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 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.

The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page

- JavaScript can create new HTML events in the page

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.
The DOM defines a standard for accessing documents:
"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**
HTML DOM methods are **actions** you can perform (on HTML Elements)
HTML DOM properties are **values** (of HTML Elements) that you can set or change

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).
In the DOM, all HTML elements are defined as **objects**.
The programming interface is the properties and methods of each object.
A **property** is a value that you can get or set (like changing the content of an HTML element).
A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<html>
<body>

<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, getElementById is a **method**, while innerHTML is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the **innerHTML** property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

The HTML DOM Document

In the HTML DOM object model, the document object represents your web page.

The document object is the owner of all other objects in your web page.

If you want to access objects in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	**Description**
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Changing HTML Elements

Method	**Description**
element.innerHTML=	Change the inner HTML of an element

element.attribute=	Change the attribute of an HTML element
element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

Adding and Deleting Elements

Method	**Description**
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output

Adding Events Handlers

Method	**Description**
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.
Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	**Description**	**DOM**
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1

document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3

document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

EVENT HANDLING

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

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

<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>

</body>
</html>
```

In this example, a function is called from the event handler:

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
    id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

In the example above, a function named *displayDate* is assigned to an HTML element with the id="myBtn".

The function will be executed when the button is clicked.

The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

```
<body onload="checkCookies()">
```

The onchange Event

The onchangeevent are often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" onchange="upperCase()">

<!DOCTYPE html>
<html>
<head>
<script>
FunctionmyFunction() {
var x = document.getElementById("fname");
x.value = x.value.toUpperCase();
}
</script>
</head>
<body>
```

```
Enter your name: <input type="text" id="fname" onchange="myFunction()">
<p>When you leave the input field, a function is triggered which transforms the
input text to upper case.</p>

</body>
</html>
```

The onmouseover and onmouseout Events

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

```
<!DOCTYPE html>
<html>
<body>
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>
<script>
FunctionmOver(obj) {
obj.innerHTML = "Thank You"
}

FunctionmOut(obj) {
obj.innerHTML = "Mouse Over Me"
}
</script>
</body></html>
```

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

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

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me</div>
<script>
FunctionmDown(obj) {
obj.style.backgroundColor = "#1ec5e5";
obj.innerHTML = "Release Me";
}

FunctionmUp(obj) {
obj.style.backgroundColor="#D94A38";
obj.innerHTML="Thank You";
}
</script>

</body>
</html>
```

The onfocus()

Change the background-color of an input field when it gets focus.

```
<!DOCTYPE html>
<html>
<head>
<script>
FunctionmyFunction(x) {
x.style.background = "yellow";
}
</script>
</head>
<body>
Enter your name: <input type="text" onfocus="myFunction(this)">
<p>When the input field gets focus, a function is triggered which changes the
background-color.</p>
</body>
</html>
```

Create a web page using two image files, which switch between one another as the mouse pointer moves over the images. Use the on Mouse Over and on Mouse Out

```
<!DOCTYPE html>
<html>
<head>
<script>
Functionlighton() {
document.getElementById('myimage').src = "bulbon.gif";
}
Functionlightoff() {
document.getElementById('myimage').src = "bulboff.gif";
}
</script>
</head>

<body>

<img id="myimage" onmousedown="lighton()" onmouseup="lightoff()"
src="bulboff.gif" width="100" height="180" />

<p>Click mouse and hold down!</p>

</body>
</html>
```

EXCEPTION HANDLING

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

Syntax Errors

Syntax errors, also called **parsing errors,** occur at compile time in traditional programming languages and at interpret time in JavaScript.

Runtime Errors

Runtime errors, also called **exceptions,** occur during execution (after compilation/interpretation).

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.

The try...catch...finally Statement

The **try** statement lets you test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

The **finally** statement lets you execute code, after try and catch, regardless of the result.

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.

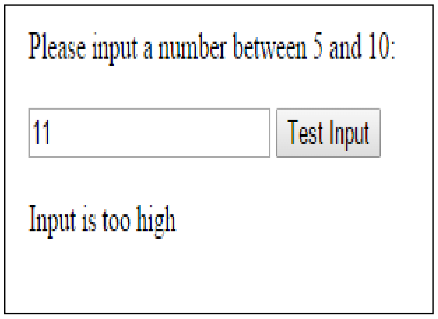
Here is the **try...catch...finally** block syntax −

```
<scripttype="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.

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

<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="message"></p>
<script>
FunctionmyFunction() {
var message, x;
message = document.getElementById("message");
message.innerHTML = "";
    x = document.getElementById("demo").value;
try {
if(x == "")  throw "empty";
if(isNaN(x)) throw "not a number";
        x = Number(x);
```


Please input a number between 5 and 10:

11

Input is too high

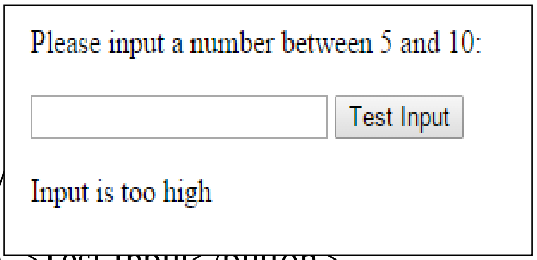
```
if(x < 5)    throw "too low";
if(x > 10)   throw "too high";
    }
catch(err) {
message.innerHTML = "Input is " + err;
    }
}
</script>
</body></html>
```

The finally Statement

The **finally** statement lets you execute code, after try and catch, regardless of the result:

try {
   Block of code to try
}
catch(err) {
   Block of code to handle errors
}
finally {
   Block of code to be executed regardless of the try / catch result
}

Example:



```
Please input a number between 5 and 10:
```

```
Input is too high
```

```
<!DOCTYPE html>
<html>
<body>
<p>Please input a number between 5 and 10:</
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>

<p id="message"></p>
<script>
FunctionmyFunction() {
var message, x;
message = document.getElementById("message");
message.innerHTML = "";
   x = document.getElementById("demo").value;
try {
if(x == "")  throw "is empty";
if(isNaN(x)) throw "is not a number";
        x = Number(x);
if(x > 10)   throw "is too high";
if(x < 5)    throw "is too low";
    }
catch(err) {
message.innerHTML = "Input " + err;
    }
finally {
document.getElementById("demo").value = "";
    }
}
</script>
</body></html>
```

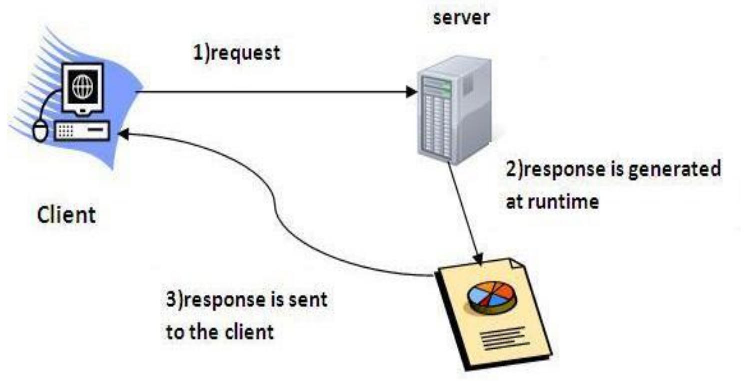
UNIT III SERVER SIDE PROGRAMMING

Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies- Installing and Configuring Apache Tomcat Web Server; DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example – **JSP:** Understanding Java Server Pages-JSP Standard Tag Library(JSTL)-Creating HTML forms by embedding JSP code.

SERVLETS

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet technology is *robust and scalable* because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there were many disadvantages of this technology.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

Servlet can be described in many ways, depending on the context.

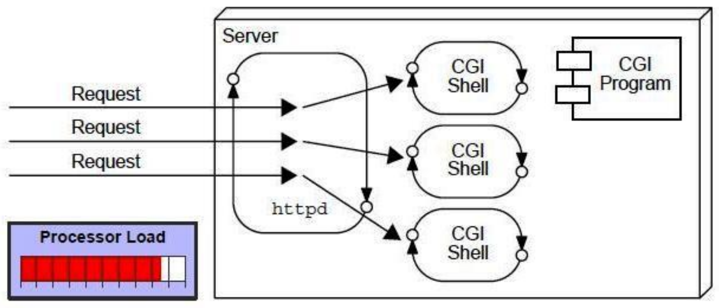
- Servlet is a technology i.e. used to create web application.
- Servlets are small Java programs that run inside servers.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming request. It can respond to any type of requests.

- Servlet is a web component that is deployed on the server to create dynamic web page.

CGI (Common Gateway Interface)

CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works :

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.

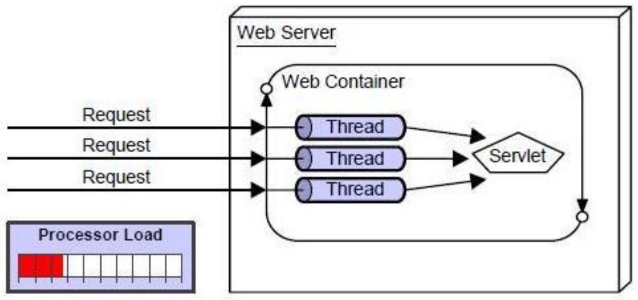


Drawbacks of CGI programs

- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Advantages of using Servlets



- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.

HTTP (Hyper Text Transfer Protocol)

1. Http is the protocol that allows web servers and browsers to exchange data over the web.
2. It is a request response protocol.
3. Http uses reliable TCP connections bydefault on TCP port 80.
4. It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user bydefault.

Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

HTTP Request	**Description**
GET	Asks to get the resource at the requested URL.

POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Container

It provides runtime environment for JavaEE (j2ee) applications.

Operations of Container:

- Life Cycle Management
- Communication Support
- Multithreaded support
- Security etc.

1. Life cycle management: Servlet and JSP are dynamic resources of java based web application. The Servlet or JSP will run on a server and at server side. A container will take care about life and death of a Servlet or JSP.
A container will instantiate, Initialize, Service and destroy of a Servlet or JSP. It means life cycle will be managed by a container.

2. Communication Support: If Servlet or JSP wants to communicate with server than its need some communication logic like socket programming. Designing communication logic is increase the burden on programmers, but container act as a mediator between a server and a Servlet or JSP and provides communication between them.

3. Multithreading: A container creates a thread for each request, maintains the thread and finally destroys it whenever its work is finished.

4. Security: A programmer is not required to write security code in a Servlet/JSP. A container will automatically provide security for a Servlet/JSP.

Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Example of Web Servers are: **Apache Tomcat** and **Resin**.

Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Example of Application Servers are:

1. **JBoss** Open-source server from JBoss community.
2. **Glassfish** provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic** provided by Oracle. It more secured.

4. **Websphere** provided by IBM.

Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

There are many content types:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- video/quicktime etc.

How a Servlet Application works

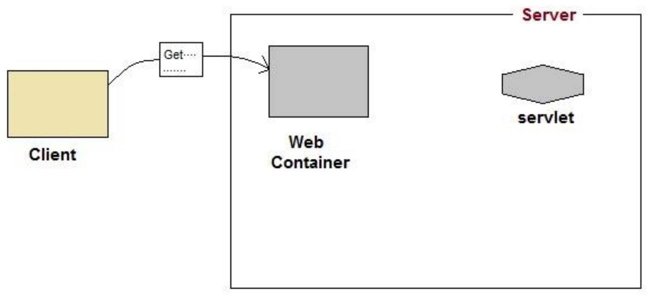
Web container is responsible for managing execution of servlets and JSP pages for Java EE application.

When a request comes in for a servlet, the server hands the request to the Web Container.

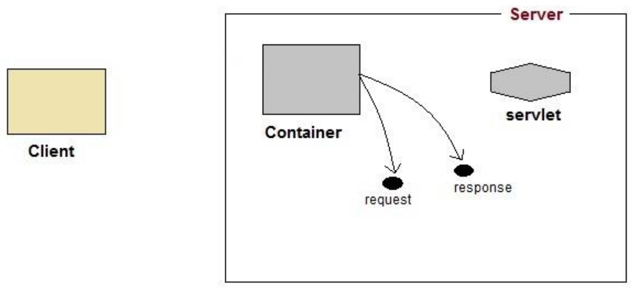
Web Container is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

Servlets don't have a main() method. Web Container manages the life cycle of a Servlet instance.

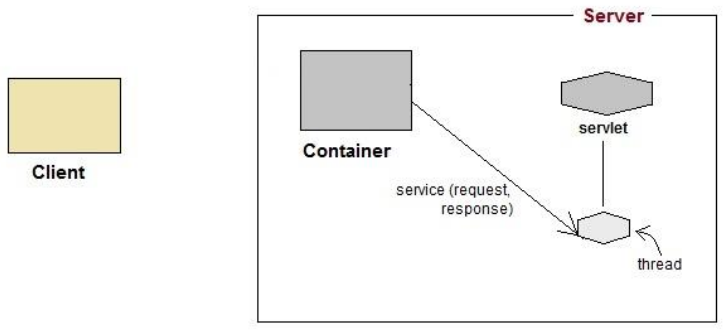
1. User sends request for a servlet by clicking a link that has URL to a servlet.



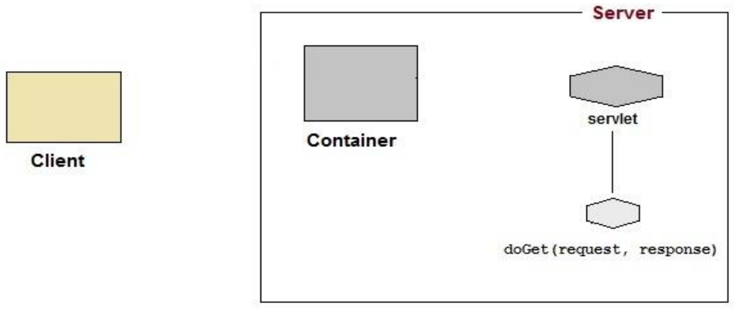
2. The container finds the servlet using **deployment descriptor** and creates two objects :
    a. **HttpServletRequest**
    b. **HttpServletResponse**



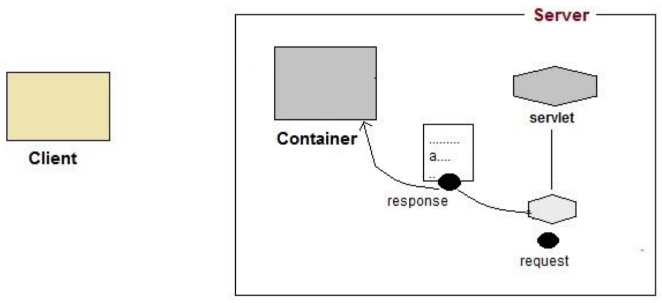
3. Then the container creates or allocates a thread for that request and calls the Servlet's service() method and passes the **request, response** objects as arguments.



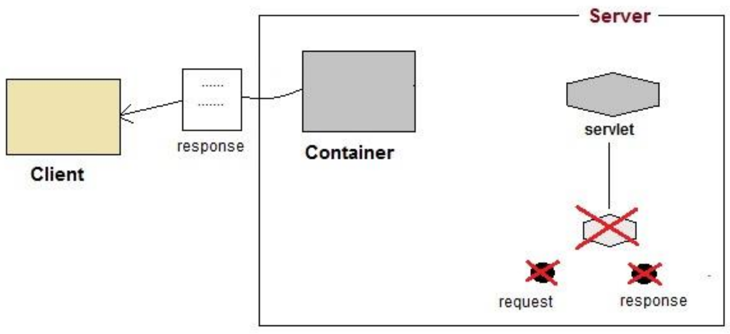
4. The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.



5. Then the Servlet uses response object to write the response back to the client.



6. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



Difference between Applet and Servlet

Applets

- Applets are applications designed to be transmitted over the network and executed by Java compatible web browsers.
- An Applet is a client side java program that runs within a Web browser on the client machine.
- An applet can use the user interface classes like AWT or Swing.

- Applet Life Cycle Methods: init(), stop(), paint(), start(), destroy()

Servlets

- Servlets are Java based analog to CGI programs, implemented by means of servlet container associated with an HTTP server.
- Servlet is a server side component which runs on the web server.
- The servlet does not have a user interface.
- Servlet Methods: doGet(), doPost()

SERVLET LIFE CYCLE

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
publicvoidinit()throwsServletException{// Initialization code...}
```

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
publicvoid service(ServletRequest request,ServletResponse response)

                          throwsServletException,IOException{

}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
publicvoiddoGet(HttpServletRequest request,

HttpServletResponse response)

throwsServletException,IOException{

// Servlet code}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
publicvoiddoPost(HttpServletRequest request,

HttpServletResponse response)

throwsServletException,IOException{

// Servlet code

}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
publicvoid destroy(){

// Finalization code...

}
```

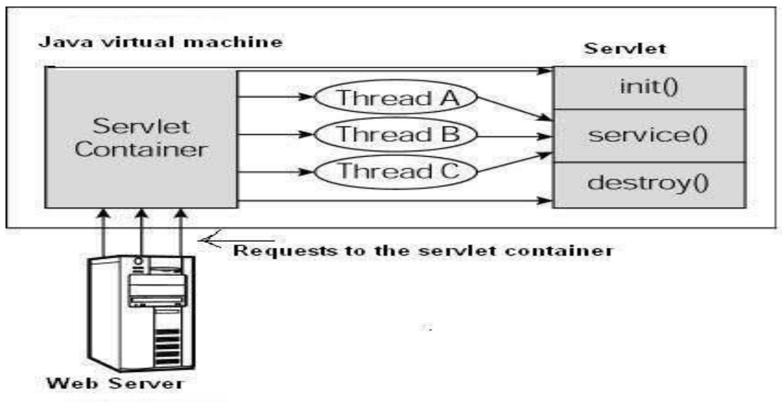
SERVLET ARCHITECTURE

Architecture Diagram:

The following figure depicts a typical servlet life-cycle scenario.

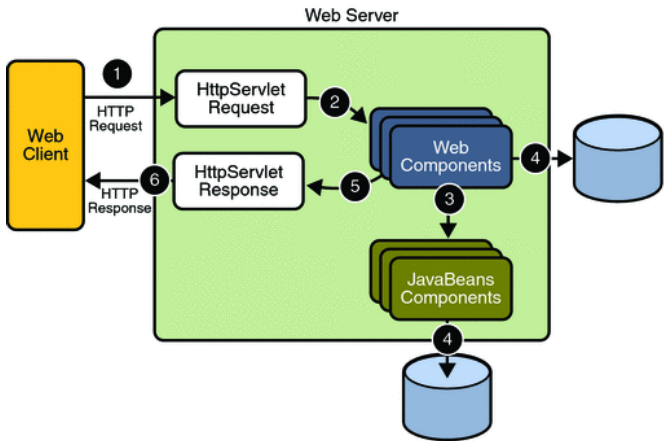
- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.

- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



A servlet is a Java class that can be loaded dynamically into and run by a special web server. This servlet-aware web server is called servlet container, which is also called a servlet engine in the early days of the servlet technology.

Servlets interact with clients via request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses. However, a servlet container also supports similar protocols such as HTTPS for secure transaction.



STEPS TO CREATE SERVLET APPLICATION USING TOMCAT SERVER

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine.

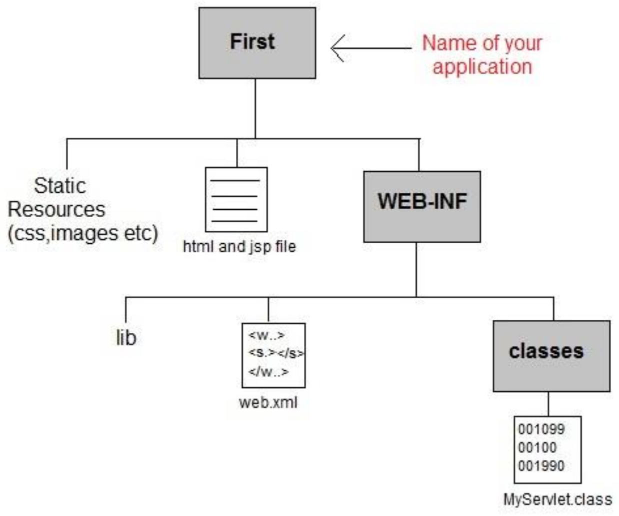
After installing Tomcat Server on your machine follow the below mentioned steps :

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployement Descriptor for your application
5. Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

1. Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, cssetc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside classes folder.

The web.xml (deployement descriptor) file is kept under WEB-INF folder.

Creating a Servlet

There are three different ways to create a servlet.

- By implementing **Servlet** interface
- By extending **GenericServlet** class
- By extending **HttpServlet** class

But mostly a servlet is created by extending **HttpServlet** abstract class. As discussed earlier **HttpServlet** gives the definition of `service()` method of the **Servlet** interface. The servlet class that we will create should not override `service()` method. Our servlet class will override only `doGet()` or `doPost()` method.

When a request comes in for the servlet, the Web Container calls the servlet's `service()` method and depending on the type of request the `service()` method calls either the `doGet()` or `doPost()` method.

NOTE: By default a request is **Get** request.

```
importjavax.servlet.*;

importjavax.servlet.http.*;

import java.io.*;


publicMyServletextendsHttpServlet

{

public void doGet(HttpServletRequestrequest,HttpServletResposne response)

throwsServletException

 {

response.setContentType("text/html");

PrintWriterout = response.getWriter();

out.println("<html><body>");

out.println("<h1>Hello Readers</h1>");

out.println("</body></html>"); }}
```

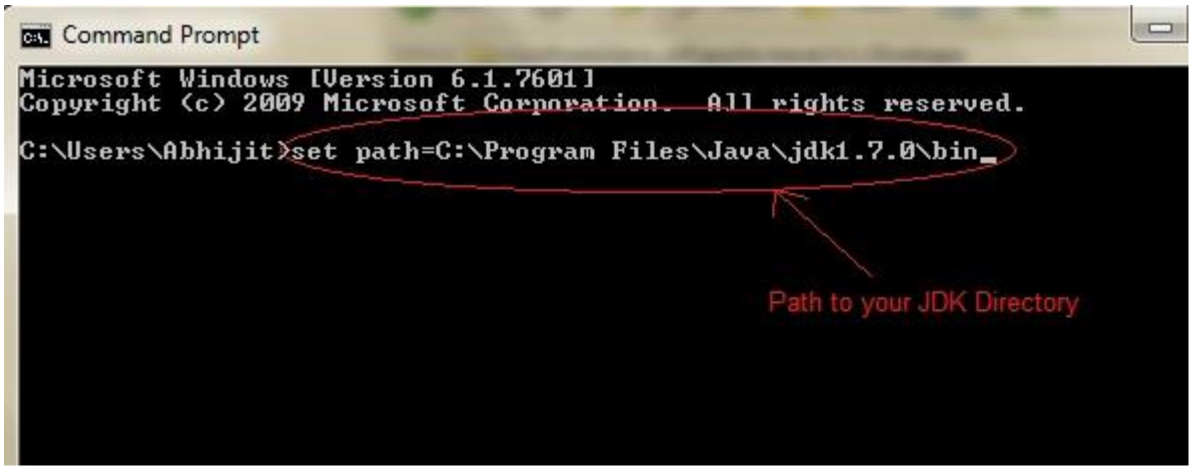
Write above code in a notepad file and save it as **MyServlet.java** anywhere on your PC. Compile it(explained in next step) from there and paste the class file into `WEB-INF/classes/` directory that you have to create inside **Tomcat/webapps** directory.

Compiling a Servlet

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server `servlet-api.jar` file is required to compile a servlet class.

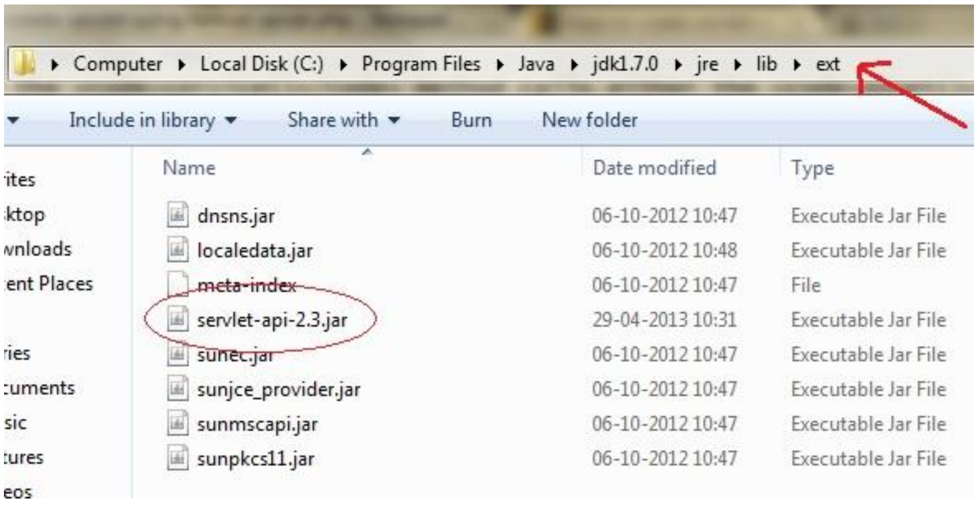
Steps to compile a Servlet

- Set the Class Path.



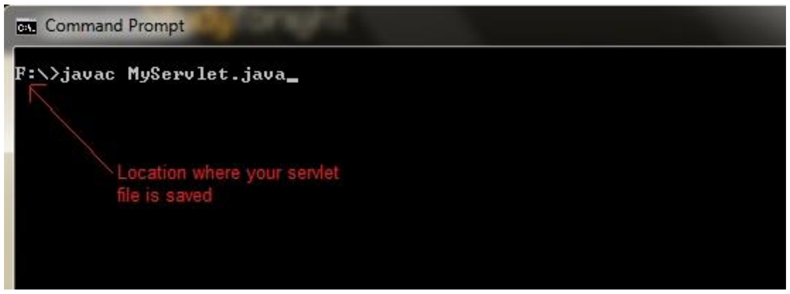
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Abhijit>set path=C:\Program Files\Java\jdk1.7.0\bin_
```

- Download **servlet-api.jar** file.
- Paste the servlet-api.jar file inside `Java\jdk\jre\lib\ext` directory.



Name	Date modified	Type
dnsns.jar	06-10-2012 10:47	Executable Jar File
localedata.jar	06-10-2012 10:48	Executable Jar File
meta-index	06-10-2012 10:47	File
servlet-api-2.3.jar	29-04-2013 10:31	Executable Jar File
sunec.jar	06-10-2012 10:47	Executable Jar File
sunjce_provider.jar	06-10-2012 10:47	Executable Jar File
sunmscapi.jar	06-10-2012 10:47	Executable Jar File
sunpkcs11.jar	06-10-2012 10:47	Executable Jar File

- Compile the Servlet class.



```
F:\>javac MyServlet.java_
```

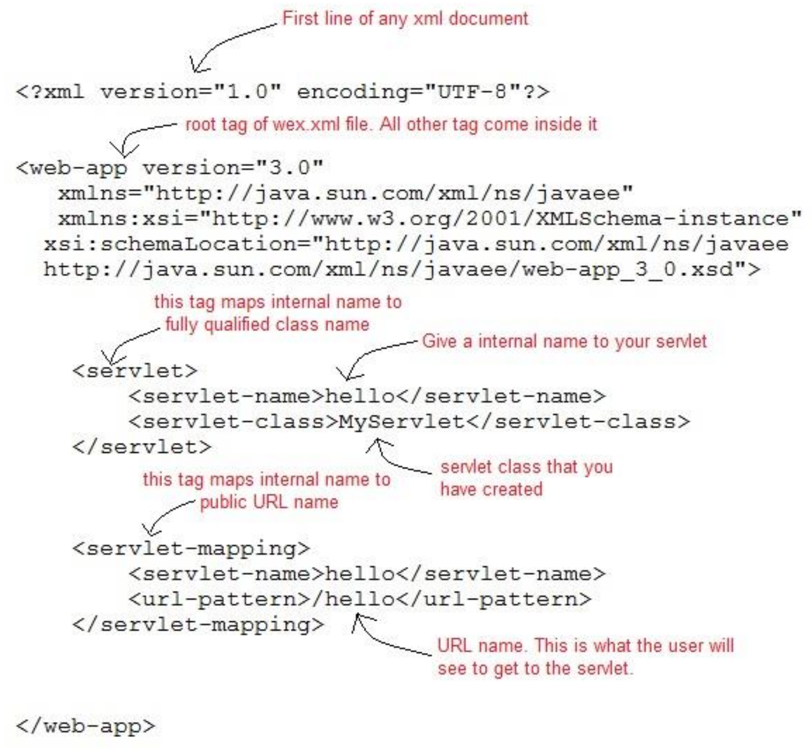
NOTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

- Mapping URL to Servlet class.
- Initializing parameters.
- Defining Error page.
- Security roles.
- Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.



```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

First line of any xml document
root tag of wex.xml file. All other tag come inside it
this tag maps internal name to fully qualified class name
Give a internal name to your servlet
servlet class that you have created
this tag maps internal name to public URL name
URL name. This is what the user will see to get to the servlet.

Start the Server

Double click on the **startup.bat** file to start your Apache Tomcat Server.

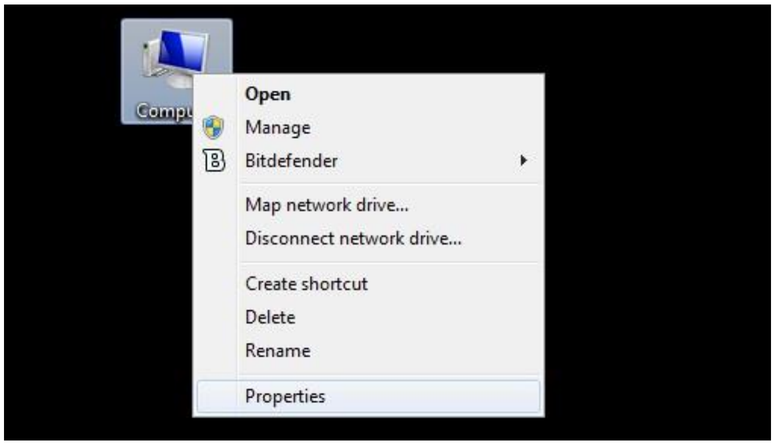
Or, execute the following command on your windows machine using RUN prompt.

```
C:\apache-tomcat-7.0.14\bin\startup.bat
```

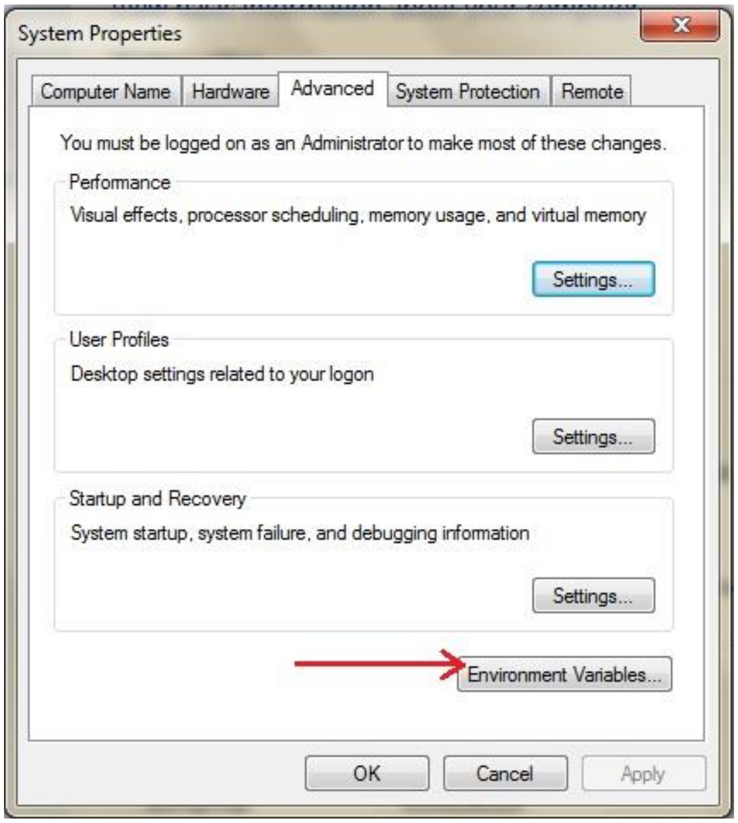
Starting Tomcat Server for the first time

If you are starting Tomcat Server for the first time you need to set JAVA_HOME in the Enviroment variable. The following steps will show you how to set it.

- Right Click on **My Computer**, go to **Properites**.



- Go to **Advanced** Tab and Click on **Environment Variables...** button.



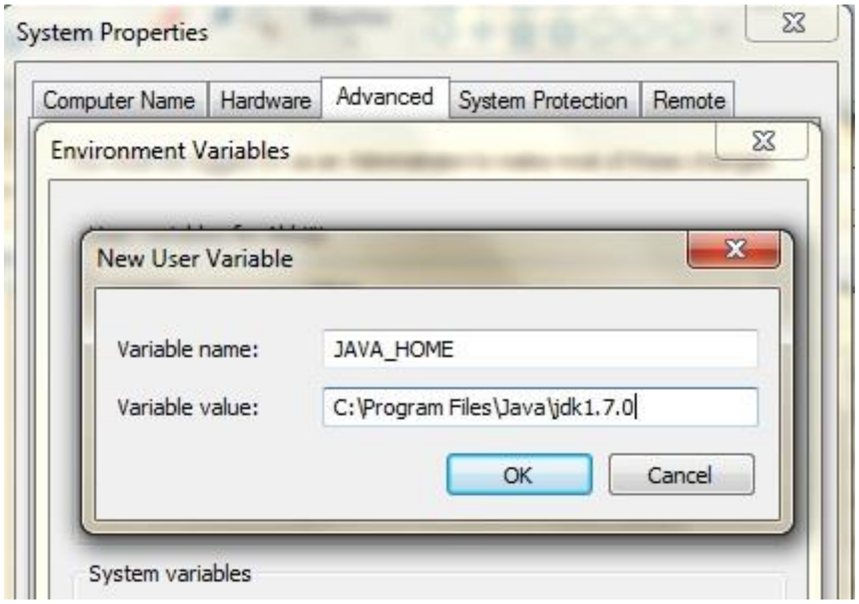
You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your logon

System startup, system failure, and debugging information

- Click on **New** button, and enter **JAVA_HOME** inside Variable name text field and path of JDK inside Variable value text field. Click OK to save.

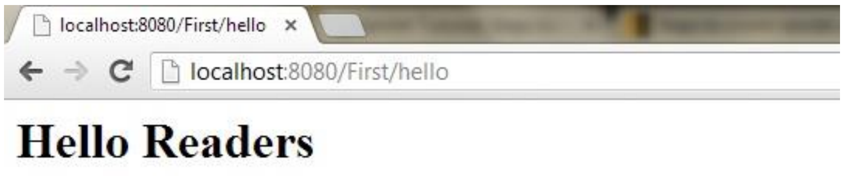


Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.7.0

Run Servlet Application

Open Browser and type **http:localhost:8080/First/hello**



localhost:8080/First/hello

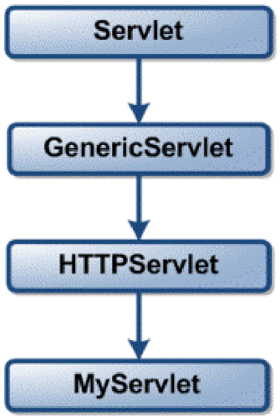
Hello Readers

Hurray! Our first Servlet Application ran successfully.

SERVLET API

There are two packages used to implement the Servlet. These packages are:

- javax.servlet
- javax.servlet.http



The javax.servlet package

- To implement the servlet
- Out of many other interfaces and classes defined in this package the most important interface is Servlet.
- Contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- All the servlet must implement this interface.

Interfaces in javax.servlet package

Interface	**Description**
Servlet	This interface defines all the lifecycle methods
ServletConfig	This interface obtains the initialization parameters
ServletContext	Using this interface the events can be logged
ServletRequest	This interface is useful in reading the data from the client request.
ServletResponse	This interface is useful in writing the data to the client request.

Servlets Interface

Servlet interface provides common behavior to all the servlets.Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfigconfig)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only
public void service(ServletRequestrequest,ServletResponse	provides response for the incoming request. It is invoked at each
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfiggetServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version

The ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet using its initialization phase. This object can be used to get configuration information from web.xml file.

Method	Description
Enumeration getInitParameterNames()	Returns an enumeration of all the initialization parameter names.
String getServletName()	Returns the name of the Servlet
String getInitParameter(String name)	Returns the parameter value for the specified parameter name.

ServletContextgetServletContext()	Returns an object of ServletContext.

The ServletContext Interface

ServletContext is one of pre-defined interface available in javax.servlet.*; Object of ServletContext interface is available one per web application. An object of ServletContext is automatically created by the container when the web application is deployed.

Method	**Description**
Object getAttribute(String attribute_name)	The value of the attribute attribute_name in the current session is returned.
void setAttribute(**String attribute_name, object value)**	The attribute_name is passed to the object value.
String getServerInfo( )	This method returns the information about the server
String log(String str)	Writes the str to the servlet log.
String getMimeType(String file)	It returns the MIME type of the file.

The ServletRequest Interface

Method	**Description**
Object getAttribute(String attribute_name)	The value of the attribute attribute_name in the current session is returned.
initgetContentlength( )	Returns the content size of the request
String getContentType( )	Returns the type of the request
getInputStream( )	Read the binary data from the request
getProtocol( )	Read the name of the protocol used
getReader( )	Reading the text from the request
getServerName( )	Returns the name of the server on which the servlet is running
initgetServerPort( )	Returns the port number of the servlet

The ServletResponse Interface

Method	Description
String getCharacterEncoding( )	Returns the character encoding
ServletOutputStreamgetOutputStream( )	Returns the output stream which is used to write the data for responding
PrintWritergetWriter( )	Write the character data to the response
void setContentLength(int size)	Sets the length of the content equal to the size.
void setContentLength(String Type)	Sets the type of the content

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

Class	Description
GenericServlet	Implements the Servlet and ServletConfig interfaces
ServletInputStream	Provides the input stream for reading the client's request
ServletOutputStream	Provides the output stream for reading the client's request
ServletException	Raise the exception when an error occurs

The javax.servlet.http

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interface	Description
HttpSession	Session data can be read or written using this interface
HttpServletRequest	Servlet can read the information from the HTTP request using this information

HttpServletResponse	Servlet can write the information to HTTP response using this interface
HttpSessionBindingListener	Tells the object about its binding with the particular session

HttpSession

Method	Description
String getId()	Returns the session ID
ObjectgetAttribute(String attribute_name)	The value of the attribute attribute_name in the current session is returned
Enumeration getAttributeNames()	Returns the attribute names

Http Servlet Request

Method	Description
String getMethod( )	Returns the HTTP method for the client request
String getPathInfo( )	Returns the path information about the servlet path
HttpSessiongetSession()	Returns the current session
String getHeader(string fields)	Returns the value of header field
Cookie[ ] getCookies()	Returns the information in the cookies in the request made
String getAuth Type()	Returns the type of authentication

Http Servlet Response

Method	Description
Void addCookie(Cookie cookie)	This method is used to add cookie in the response
String encodeURL(String url)	This method is used to encode the specified URL
Boolean containsHeader(String f)	This method returns **TRUE** if the response header contains the field f.
Void sendError(int code)	This method sends error code to the client.

HttpSessionbindinglistener

Method	Description
Object getValue()	This function returns the value of bounded or unbounded attribute**.**
String getName()	This function returns the name being bound or unbound.
HttpSessiongetSession()	This function returns the session to which the listener can be bound or unbound.

Classes

Class	Description
Cookie	This class is used to write the cookies.
Http Servlet	It is used when developing servlet that receive and process the HTTP request.
HttpSessionEvent	This class is used to handle the session event.
HttpSessionBindingEvent	When a listener is bound to a value.

1.The Cookie class

Class	Description
String getValue()	This function returns a value of the cookie.
Void setValue(String s)	This function sets the value to the cookie.
String getName()	This function returns the name.

2.TheHttpServlet class

Class	Description
Void doGet(HttpServletRequestreq, HttpServletResponse res)	This method performs HTTP get request.
Void doPost(HttpServletRequestreq, HttpServletResponse res)	This method performs HTTP post request.
Void doPut(HttpServletRequestreq, HttpServletResponse res)	This method performs HTTP put request.
Void service(HttpServletRequestreq, HttpServletResponse res)	This method is invoked for processing HTTP request and response.

3.TheHttpSessionEvent:

This method encapsulates the session event.

4.HttpSessionBindingEvent:

The **HttpSessionBindingEvent** class extends **HttpSessionEvent.**It is generated when a listener is bound to a value. The **getSession**() method obtains the session to which the listener is being bound or unbound**.**

Servlet Example by implementing Servlet interface

Let's see the simple example of servlet by implementing the servlet interface.

File: First.java

```
import java.io.*;
import javax.servlet.*;

public class First implements Servlet{
ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("servlet is initialized");
}

public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");

}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}

}
```

FORM GET AND POSTACTIONS

The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ?character as follows:

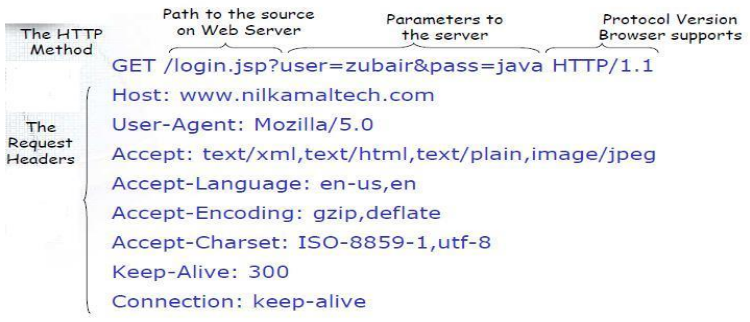
```
http://www.test.com/hello?key1=value1&key2=value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

Anatomy of Get Request

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what informations are sent to the server.


The HTTP Method
Path to the source on Web Server
Parameters to the server
Protocol Version Browser supports
The Request Headers

```
GET /login.jsp?user=zubair&pass=java HTTP/1.1
Host: www.nilkamaltech.com
User-Agent: Mozilla/5.0
Accept: text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive: 300
Connection: keep-alive
```

POST method:

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ?in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()**method.

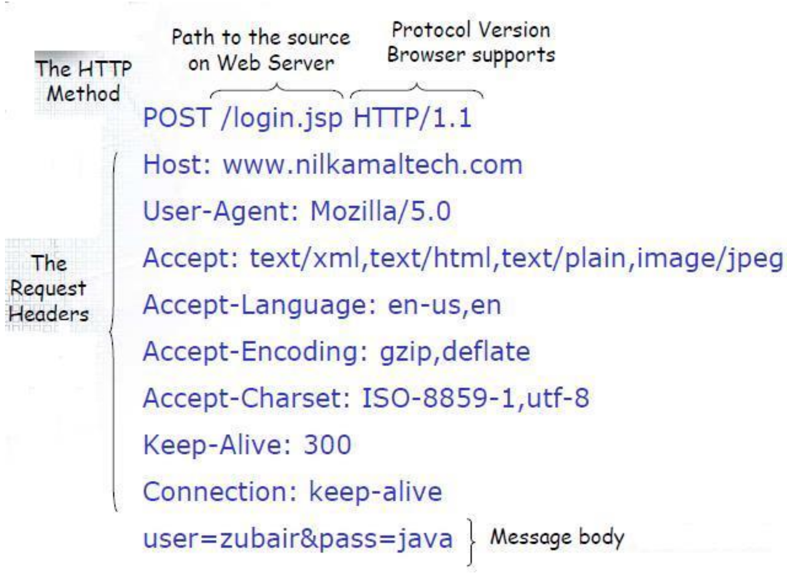
Reading Form Data using Servlet:

Servlets handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

Anatomy of Post Request

As we know, in case of post request original data is sent in message body. Let's see how informations are passed to the server in case of post request.



```
POST /login.jsp HTTP/1.1
Host: www.nilkamaltech.com
User-Agent: Mozilla/5.0
Accept: text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive: 300
Connection: keep-alive
user=zubair&pass=java
```

The HTTP Method
Path to the source on Web Server
Protocol Version Browser supports
The Request Headers
Message body

GET Method Example Using URL:

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

Below is **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information:

```
importjava.io.IOException;

importjava.io.PrintWriter;

importjavax.servlet.ServletException;

importjavax.servlet.http.HttpServlet;

importjavax.servlet.http.HttpServletRequest;

importjavax.servlet.http.HttpServletResponse;

public class HelloFormData extends HttpServlet {

      private static final long serialVersionUID = 1L;

      public    void    doGet(HttpServletRequest    request,    HttpServletResponse
response) throws ServletException, IOException {

            response.setContentType("text/html");

            PrintWriter out = response.getWriter();

            String title = "Servlet: Read Form Data";

            out.println("<html>" + "<head><title>" + title

                        + "</title></head><body>"

                        + "<h1>" + title + "</h1>"

                        + "<p>Hi  "

                        + request.getParameter("name")

                        + "</p>"

      }

}
```

Web.xml

This web.xml defines the Servlet mapping for the form data servlet.

```
<web-app xmlns=http://java.sun.com/xml/ns/j2ee

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"

version="2.4">

<display-name>Servlet Form Data Handling</display-name>

<servlet>

<servlet-name>HelloFormData</servlet-name>

<servlet-class>HelloFormData</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloFormData</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

</web-app>
```

index.html

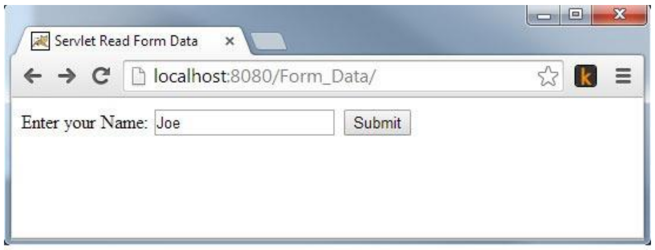
```
<!DOCTYPE html>

<html>

<head>

<title>Servlet Read Form Data</title>

</head>

<body>
```

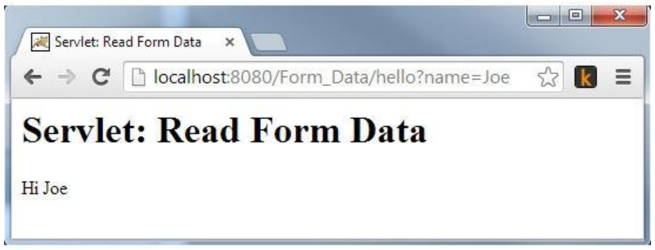
```
<form action="./hello" method="GET">

Enter your Name: <input type="text" name="name">

<input type="submit" value="Submit" />

</form>

</body>

</html>
```

Read using POST method

In the above HTML page in form tag instead of GET use as POST. Then in the servlet to read the form data add a doPost method as below,

```
public void doPost(HttpServletRequest request,

HttpServletResponse response)

throwsServletException, IOException {

doGet(request, response);  }
```





Servlet: Read Form Data

Hi Joe

What is the difference between Get and Post?

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked**	Post request **cannot be** bookmarked
4) Get request is **idempotent**. It means second request will be ignored until response of first request is delivered.	Post request is **non-idempotent**
5) Get request is **more efficient** and used more than Post	Post request is **less efficient** and used less than get.

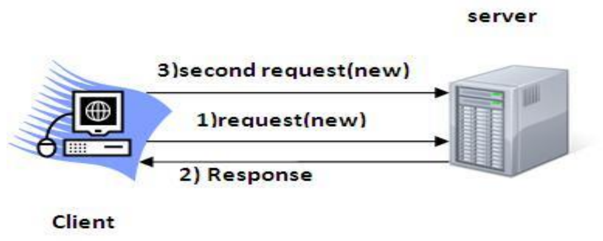
SESSION HANDLING

Session simply means a particular interval of time.

Session Tracking is a mechanism used by Webcontainer to maintain state (data) of a user. It is also known as **session management** in servlet.

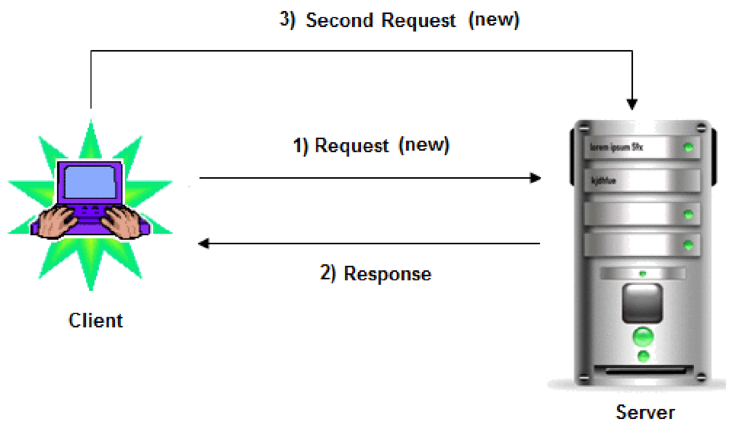
HTTP protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:

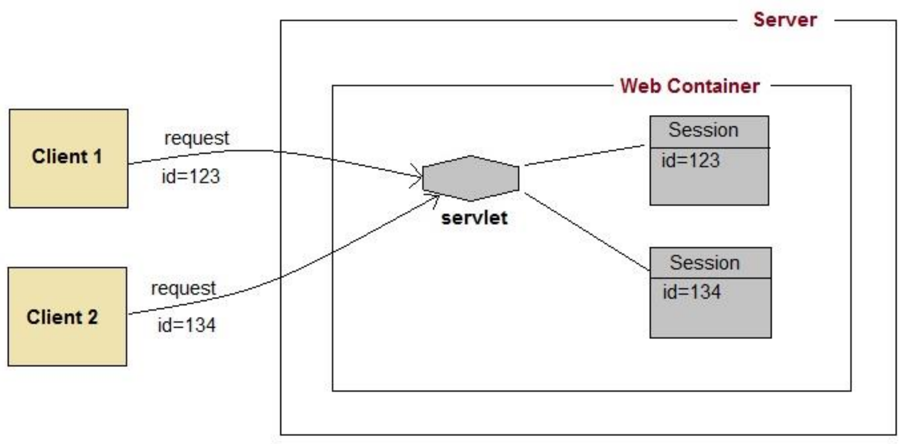


Why use Session Tracking?

- ✓ Http protocol is stateless, to make stateful between client and server we need Session Tracking.
- ✓ Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.
- ✓ **To recognize the user** it is used to recognize the particular user.



How Session Works

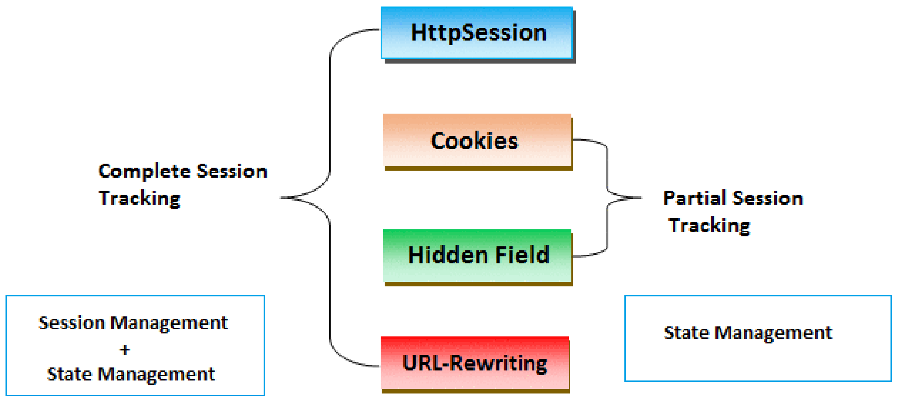


The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**



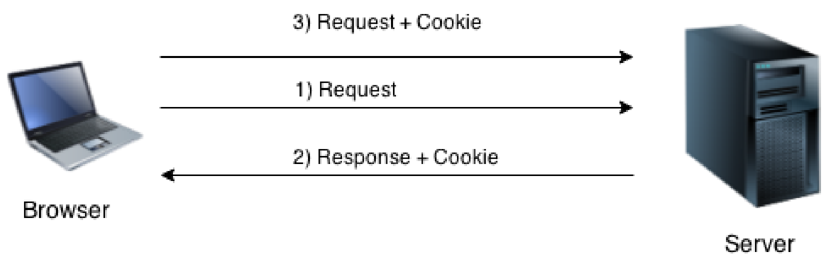
1. COOKIES

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

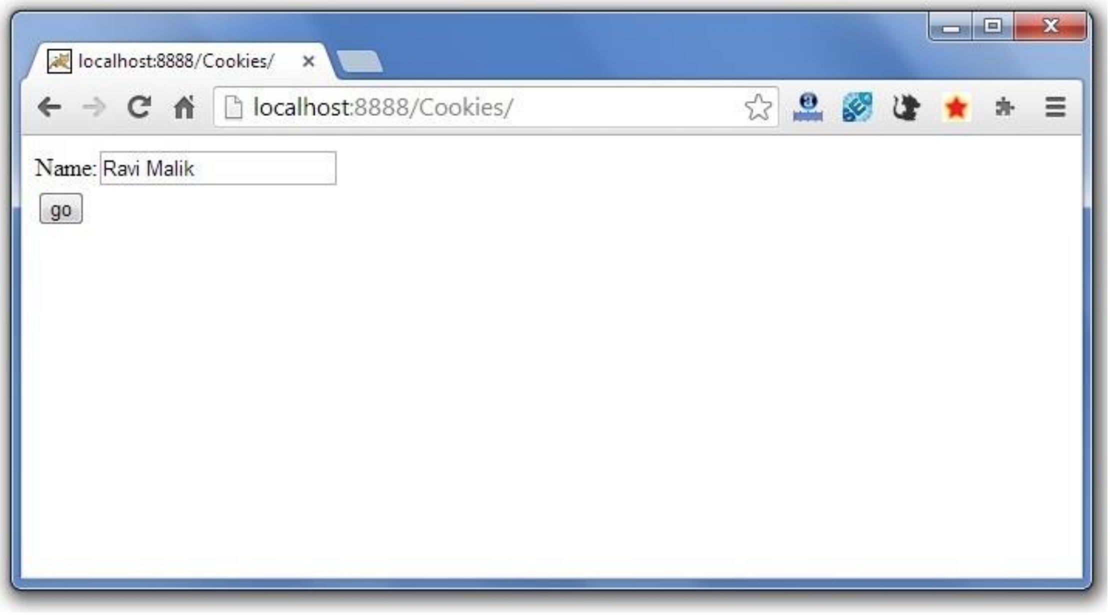
Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

Let's see the simple code to get all the cookies.

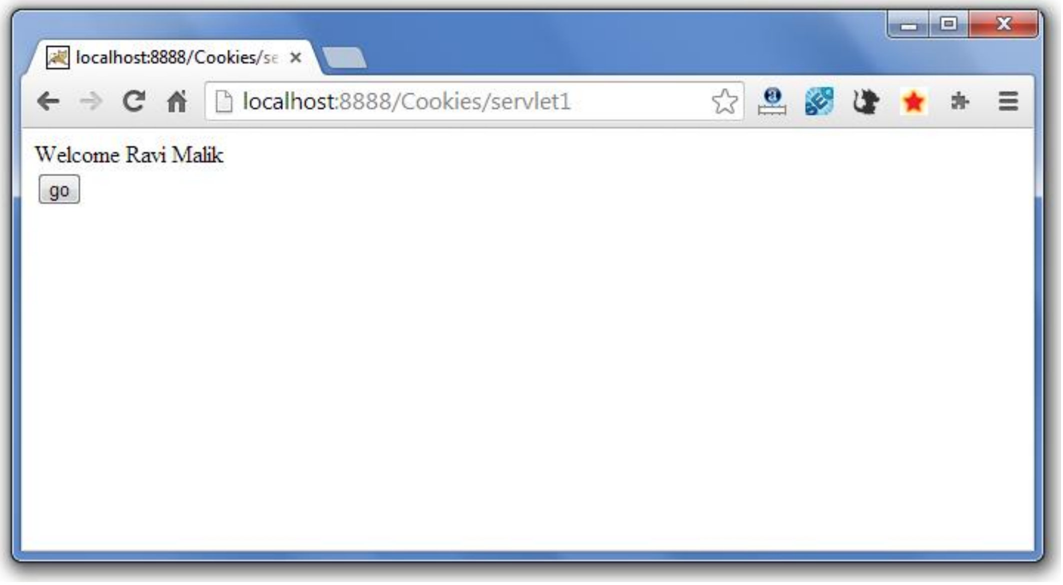
```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name an
d value of cookie
}
```



Name: Ravi Malik

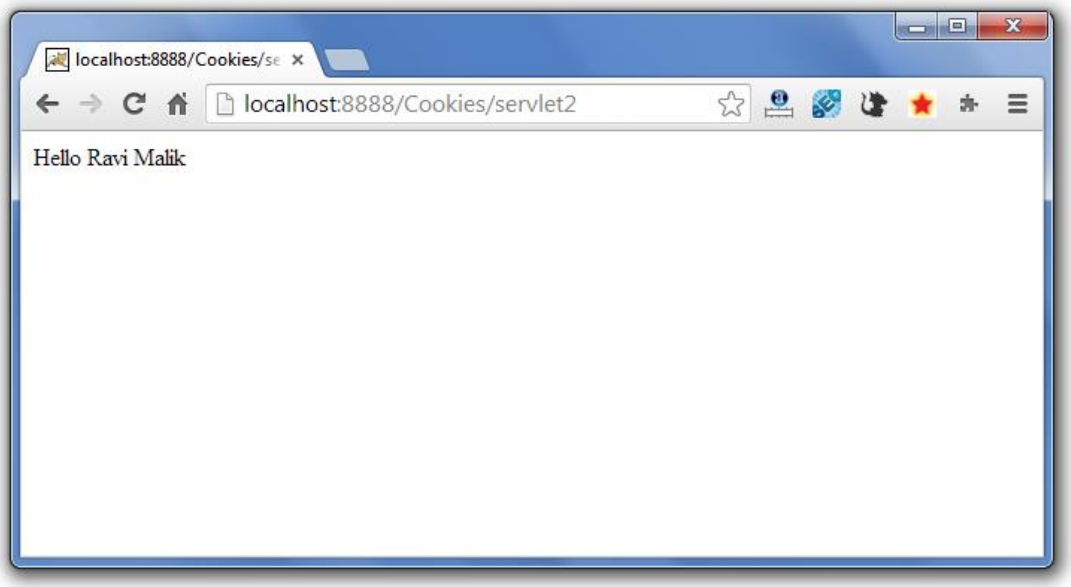
go

Output



Welcome Ravi Malik

go



Hello Ravi Malik

2. Hidden Form Field

Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

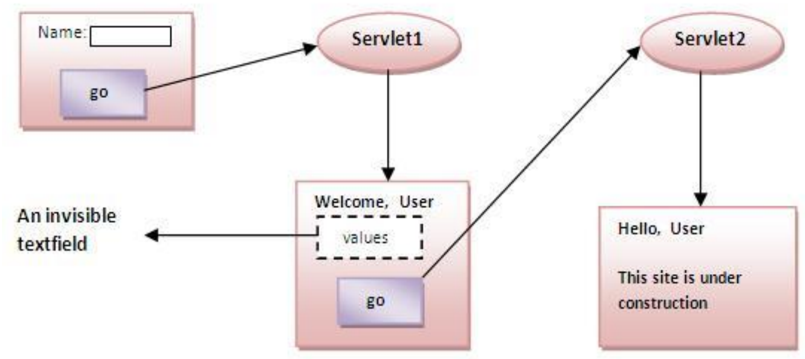
Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.

3. Only textual information can be used.

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
       try{

       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

       String n=request.getParameter("userName");
       out.print("Welcome "+n);

       //creating form that have invisible textfield
       out.print("<form action='servlet2'>");
       out.print("<input type='hidden' name='uname' value='"+n+"'>");
       out.print("<input type='submit' value='go'>");
       out.print("</form>");
```

```
        out.close();

                }catch(Exception e){System.out.println(e);}
    }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //Getting the value from the hidden field
        String n=request.getParameter("uname");
        out.print("Hello "+n);

        out.close();
                }catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
```

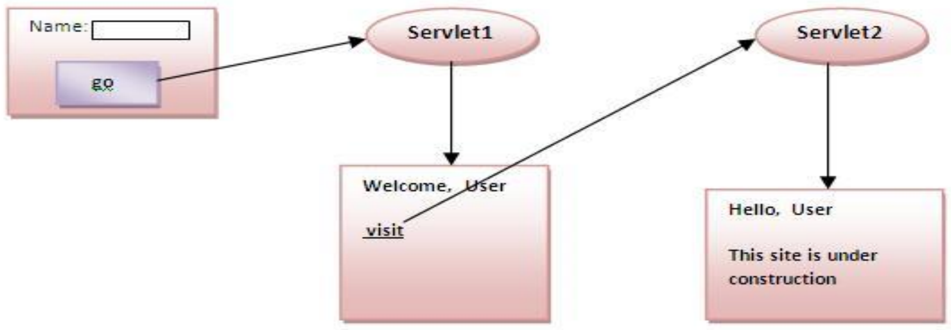
```
</web-app>
```

3. URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send only textual information.

Example of using URL Rewriting

In this example, we are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);

        //appending the username in the query string
        out.print("<a href='servlet2?uname="+n+"'>visit</a>");

        out.close();

                }catch(Exception e){System.out.println(e);}
   }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //getting value from the query string
        String n=request.getParameter("uname");
        out.print("Hello "+n);

        out.close();

                }catch(Exception e){System.out.println(e);}
    }


}
```

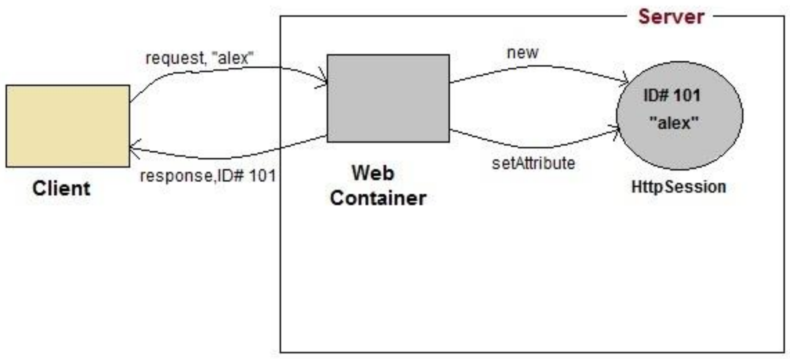
web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

4.HttpSession

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the `getSession()` method of the **HttpServletRequest** object.

How HttpSession works



1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

How to get the HttpSessionobject ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **publicHttpSessiongetSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **publicHttpSessiongetSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);

        HttpSession session=request.getSession();
        session.setAttribute("uname",n);

        out.print("<a href='servlet2'>visit</a>");

        out.close();

                }catch(Exception e){System.out.println(e);}
    }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("uname");
        out.print("Hello "+n);

        out.close();

                }catch(Exception e){System.out.println(e);}
    }


}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
```

```
</servlet-mapping>
</web-app>
```

INSTALLING AND CONFIGURING APACHE TOMCAT WEB SERVER

This section will show the process to install and configure Tomcat 6 on your computer system.

Step 1:

Installation of JDK:

Before beginning the process of installing Tomcat on your system, ensure first the availability of JDK on your system program directory. Install it on your system if not already installed (because any version of tomcat requires the Java 1.6 or higher versions) and then set the class path (environment variable) of JDK. To set the **JAVA_HOME Variable:** you need to specify the location of the java run time environment to support the Tomcat else Tomcat server cannot run.

This variable contains the path of JDK installation directory.

set JAVA_HOME=C:\Program Files\Java\jdk1.6

Note: it should not contain the path up to bin folder. Here, we have taken the URL path according to our installation convention.
For Windows OS, go through the following steps:

Start menu->Control Panel->System->Advanced tab->Environment Variables->New->set the Variable name = JAVA_HOME and variable value = C:\Program Files\Java\jdk1.6

Now click on all the subsequent ok buttons one by one. It will set the JDK path.

Step 2:

For setting the class path variable for JDK, do like this:

Start menu->Control Panel->System->Advanced tab->Environment Variables->New-> Set PATH="C:\Program Files\Java\jdk1.6\bin"; %PATH%
OR

First, right click on the

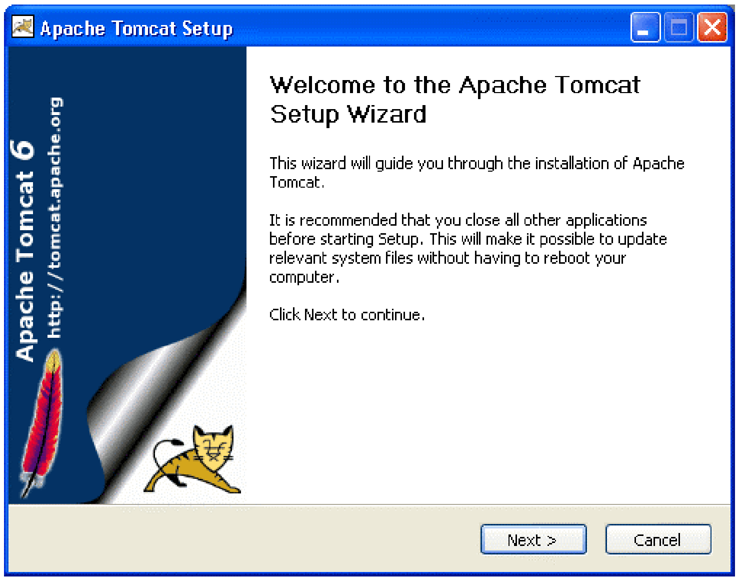
My Computer->properties->advance->Environment Variables->path.

Now, set bin directory path of JDK in the path variable

Step 3:

The process of installing Tomcat 6.0 begins here from now. It takes various steps for installing and configuring the Tomcat 6.0.

For Windows OS, Tomcat comes in two forms: .zip file and .exe file (the Windows installer file). Here we are exploring the installation process by using the .exe file. First unpack the zipped file and simply execute the '.exe' file.



Welcome to the Apache Tomcat Setup Wizard

This wizard will guide you through the installation of Apache Tomcat.

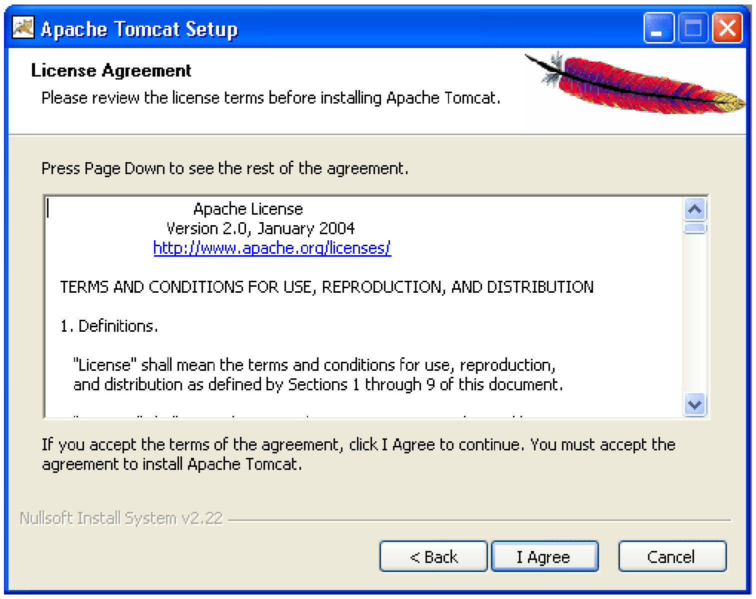
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

A Welcome screen shot appears that shows the beginning of installation process. Just click on the 'Next' button to precede the installation process.

Steps 4:

A screen of 'License Agreement' displays.


Apache Tomcat Setup

License Agreement

Please review the license terms before installing Apache Tomcat.

Press Page Down to see the rest of the agreement.

```
                    Apache License
              Version 2.0, January 2004
           http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.
```

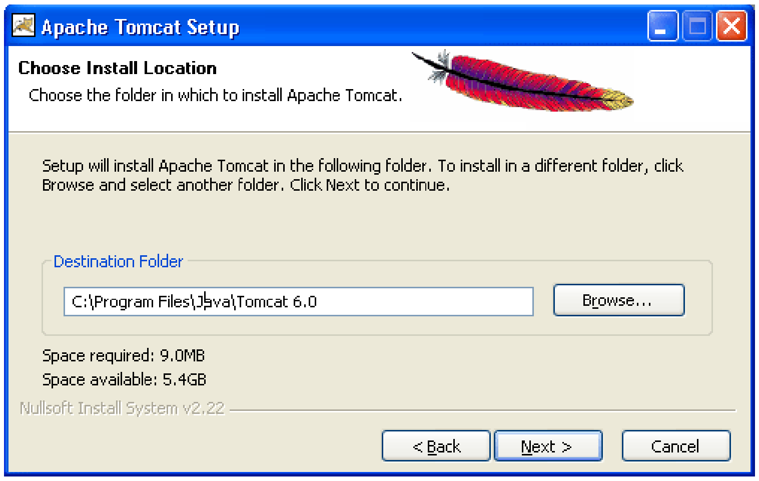
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Apache Tomcat.

Nullsoft Install System v2.22

Click on the 'I Agree' button.

Step 5:

A screen shot appears asking for the 'installing location'


Apache Tomcat Setup

Choose Install Location

Choose the folder in which to install Apache Tomcat.

Setup will install Apache Tomcat in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\Java\Tomcat 6.0

Space required: 9.0MB

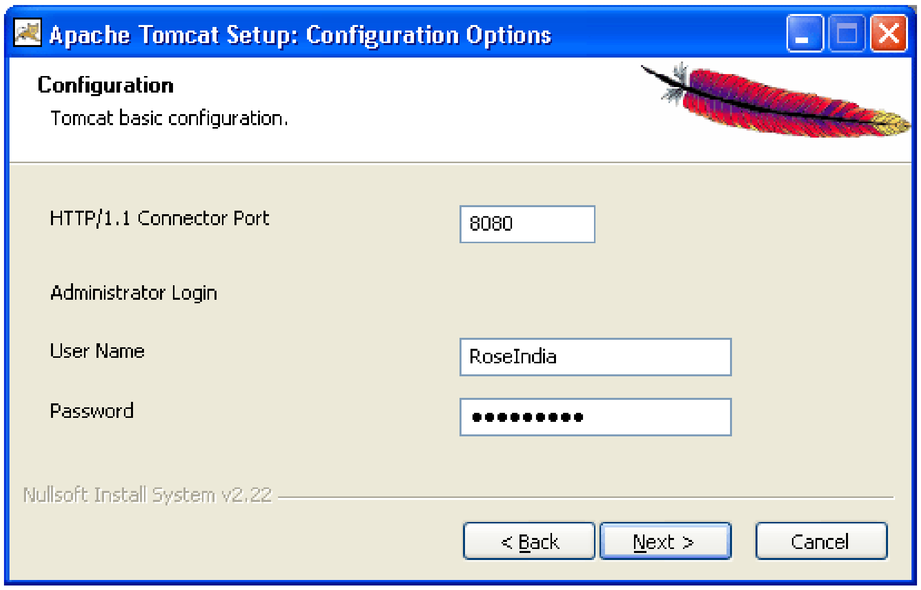
Space available: 5.4GB

Nullsoft Install System v2.22

Choose the default components and click on the 'Next' button.

Step 6:

A screen shot of 'Configuration Options' displays on the screen. Choose the location for the Tomcat files as per your convenience. You can also select the default Location


Apache Tomcat Setup: Configuration Options

Configuration

Tomcat basic configuration.

HTTP/1.1 Connector Port: 8080

Administrator Login

User Name: RoseIndia

Password: ●●●●●●●●●●

Nullsoft Install System v2.22

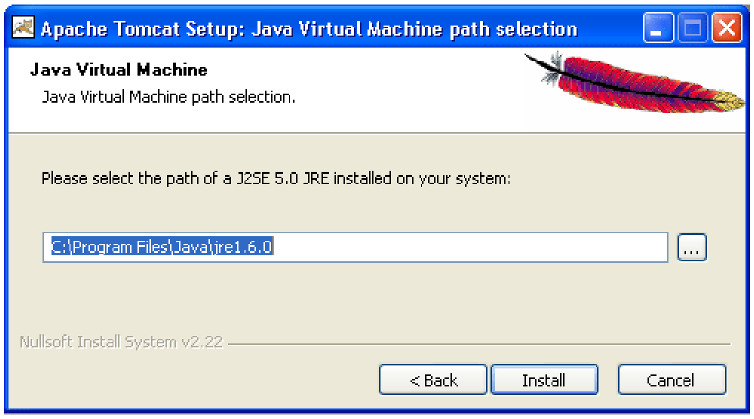
The port number will be your choice on which you want to run the tomcat server. The port number 8080 is the default port value for tomcat server to proceed the HTTP requests. The user can also change the 'port number' after completing the process of installation; for this, users have to follow the following tips.

Go to the specified location as " **Tomcat 6.0 \conf \server.xm**l ". Within the server.xml file choose "Connector" tag and change the port number.

Now, click on the 'Next' button to further proceed the installation process.

Step 7:

A Window of Java Virtual Machine displays on the screen


Apache Tomcat Setup: Java Virtual Machine path selection

Java Virtual Machine

Java Virtual Machine path selection.

Please select the path of a J2SE 5.0 JRE installed on your system:

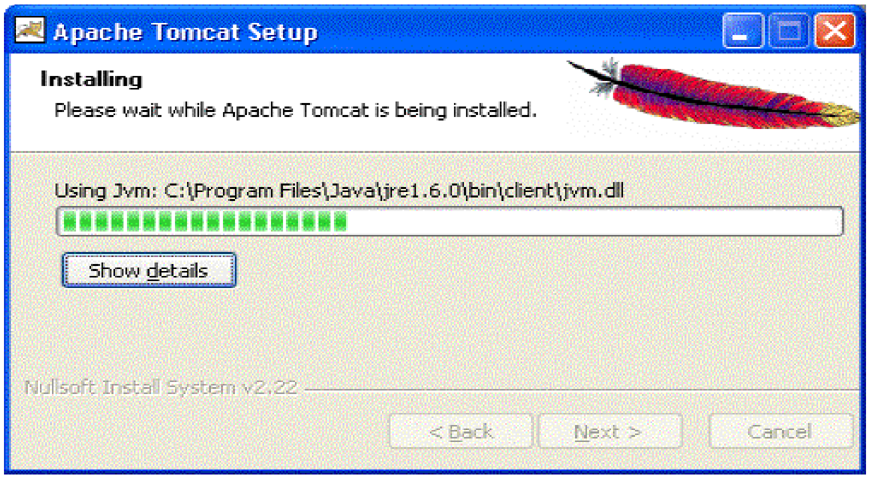
C:\Program Files\Java\jre1.6.0

Nullsoft Install System v2.22

This window asks for the location of the installed Java Virtual Machine. Browse the location of the JRE folder and click on the Install button. This will install the Apache tomcat at the specified location.

Step 8:

A processing window of installing displays on the screen.



Installing

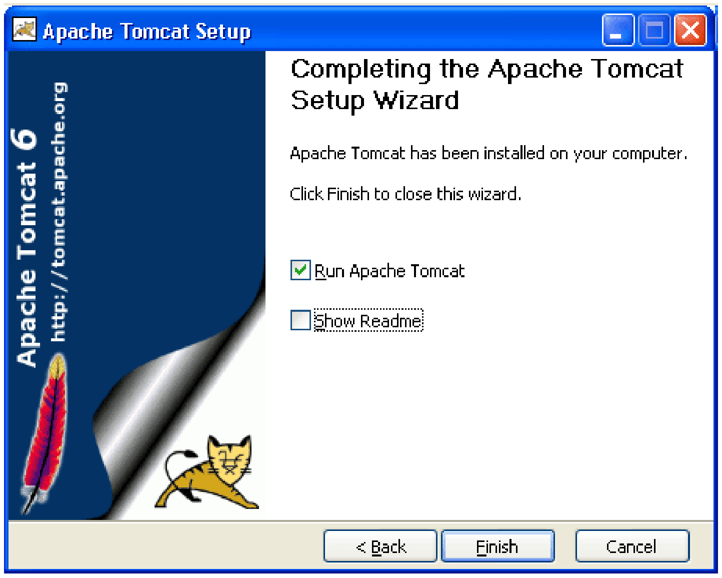
Please wait while Apache Tomcat is being installed.

Using Jvm: C:\Program Files\Java\jre1.6.0\bin\client\jvm.dll

To get the information about installer click on the "Show details" button

Step 9:

A screen shot of 'Tomcat Completion' displays on the screen.



Completing the Apache Tomcat Setup Wizard

Apache Tomcat has been installed on your computer.

Click Finish to close this wizard.

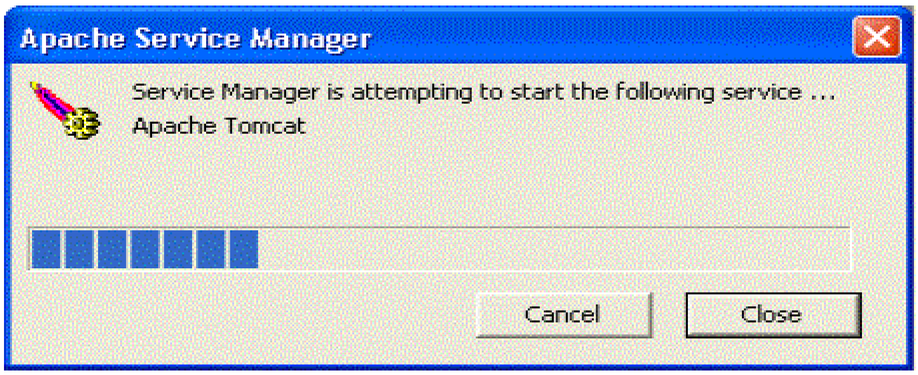
[✓] Run Apache Tomcat

[ ] Show Readme

Click on the 'Finish' button.

Step 10:

A window of Apache Service Manager appears with displaying the running process.

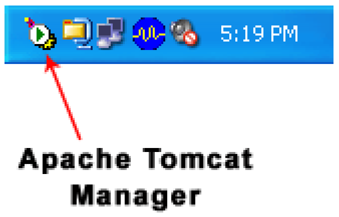


Service Manager is attempting to start the following service ...
Apache Tomcat

Let the running process goes on.

Step 11:

After completing the installation process, the Apache Tomcat Manager appears on the toolbar panel like shown in the below picture.

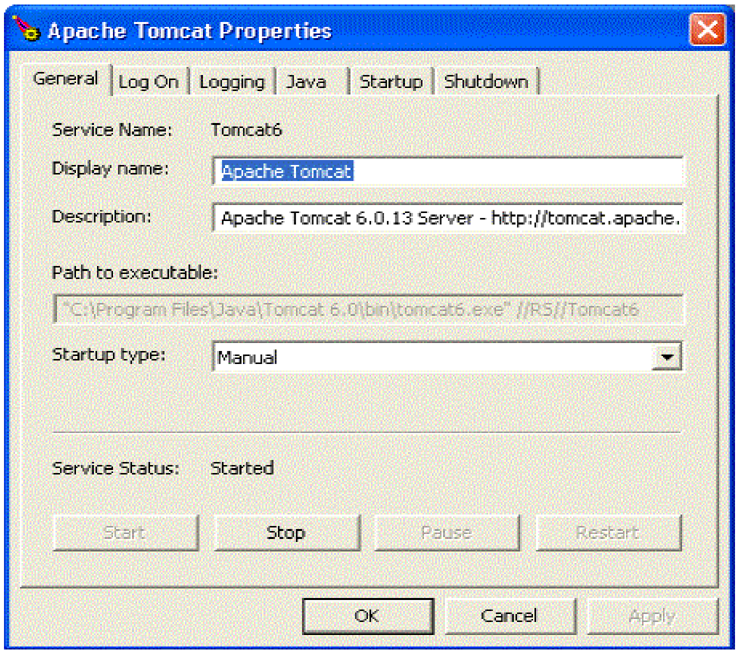


Configuring Tomcat Manager

To Configure the Tomcat Manager, there are two ways; either user can configure Tomcat directly from the toolbar panel or can configure it from Control Panel Section.

i) Configuring from toolbar Panel

To Configure Apache Tomcat web server from the toolbar panel, you have to press 'double click' on the appeared icon.


Apache Tomcat Properties

General | Log On | Logging | Java | Startup | Shutdown

Service Name: Tomcat6

Display name: Apache Tomcat

Description: Apache Tomcat 6.0.13 Server - http://tomcat.apache.

Path to executable:

"C:\Program Files\Java\Tomcat 6.0\bin\tomcat6.exe" //RS//Tomcat6

Startup type: Manual

Service Status: Started

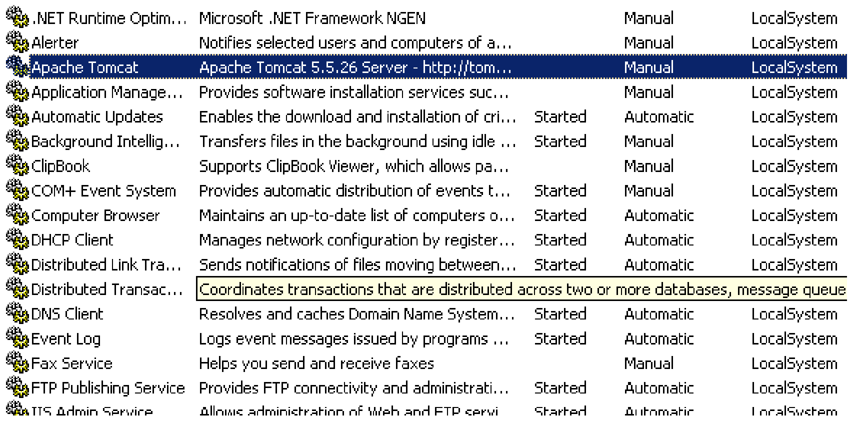
A configured window appears on the desktop. Now, just click on the Startup button. The installation process will be completed.

ii) Configuration from the Control Panel

To configure the Apache Tomcat Manager, Users will have to follow the follwing steps:

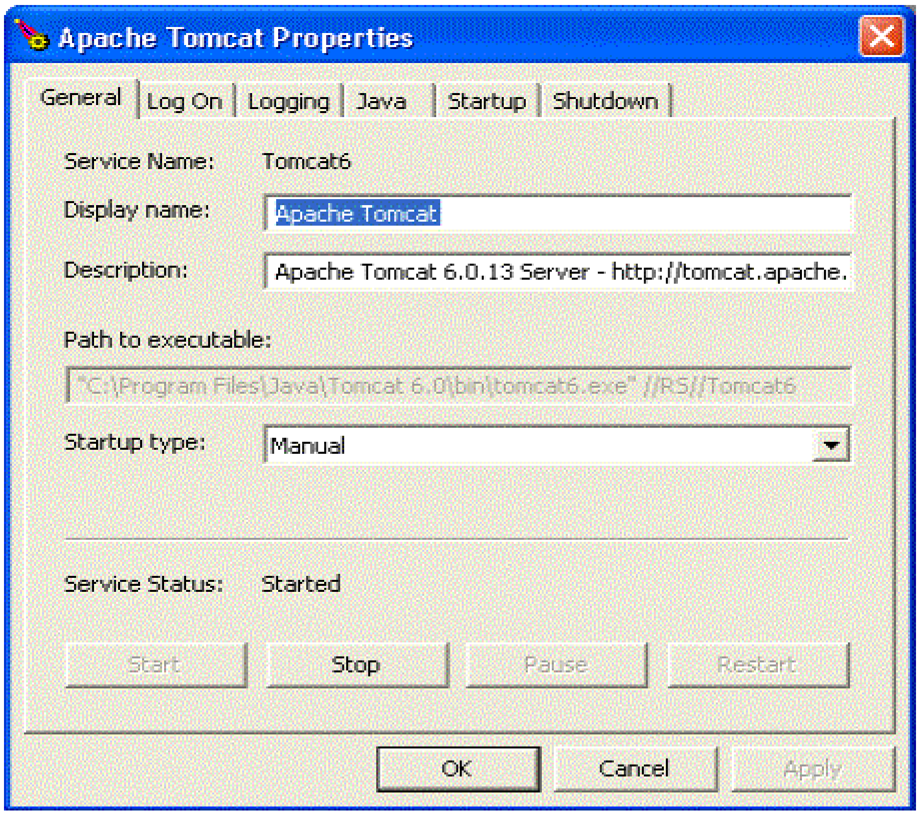
Click on the Startup button -- select Control Panel -- Administrator Tool -- Services -- select Apache Tomcat.

The following screen displays on the monitor.



.NET Runtime Optim...	Microsoft .NET Framework NGEN		Manual	LocalSystem
Alerter	Notifies selected users and computers of a...		Manual	LocalSystem
Apache Tomcat	Apache Tomcat 5.5.26 Server - http://tom...		Manual	LocalSystem
Application Manage...	Provides software installation services suc...		Manual	LocalSystem
Automatic Updates	Enables the download and installation of cri...	Started	Automatic	LocalSystem
Background Intellig...	Transfers files in the background using idle ...	Started	Manual	LocalSystem
ClipBook	Supports ClipBook Viewer, which allows pa...		Manual	LocalSystem
COM+ Event System	Provides automatic distribution of events t...	Started	Manual	LocalSystem
Computer Browser	Maintains an up-to-date list of computers o...	Started	Automatic	LocalSystem
DHCP Client	Manages network configuration by register...	Started	Automatic	LocalSystem
Distributed Link Tra...	Sends notifications of files moving between...	Started	Automatic	LocalSystem
Distributed Transac...	Coordinates transactions that are distributed across two or more databases, message queue			
DNS Client	Resolves and caches Domain Name System...	Started	Automatic	LocalSystem
Event Log	Logs event messages issued by programs ...	Started	Automatic	LocalSystem
Fax Service	Helps you send and receive faxes		Manual	LocalSystem
FTP Publishing Service	Provides FTP connectivity and administrati...	Started	Automatic	LocalSystem
IIS Admin Service	Allows administration of Web and FTP servi	Started	Automatic	LocalSystem

Double click on the Apache Tomcat. The window of Apache Tomcat Properties appears on the screen.


Apache Tomcat Properties

Service Name: Tomcat6

Display name: Apache Tomcat

Description: Apache Tomcat 6.0.13 Server - http://tomcat.apache.

Path to executable:

"C:\Program Files\Java\Tomcat 6.0\bin\tomcat6.exe" //RS//Tomcat6

Startup type: Manual

Service Status: Started

Now, Click on the start up button. The Apache Tomcat is now ready to function.

To operate it, follow the below steps of processing.

Start the Tomcat Server:

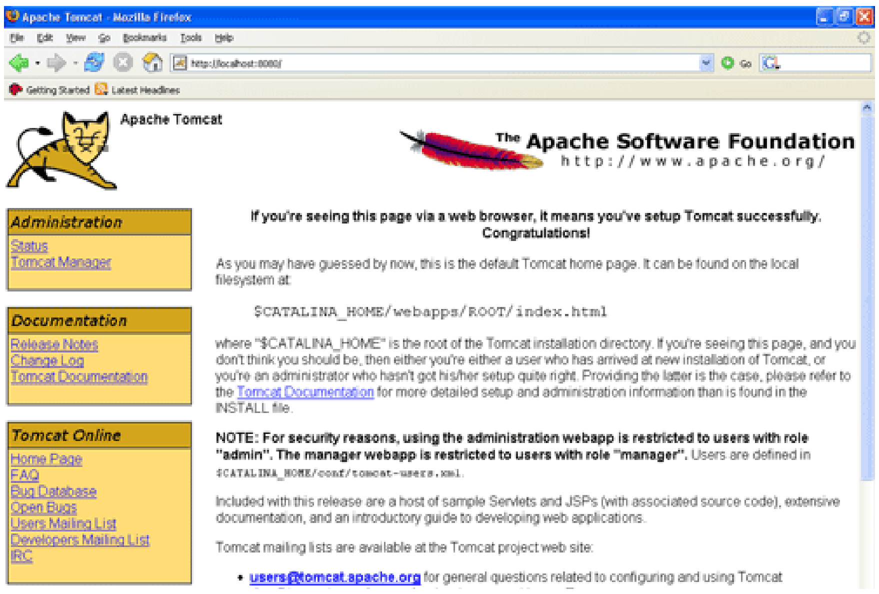
1.Start the tomcat server from the bin folder of Tomcat 6.0 directory by double clicking the "tomcat6.exe" file.

OR create a shortcut of this .exe file at your desktop.

2. Now Open web browser and type URL http://localhost:8080 in the address bar to test the server

3. To Stop the Tomcat Server: Stop the server by pressing the "Ctrl + c" keys.

The screen of Apache Tomcat software looks like this:



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

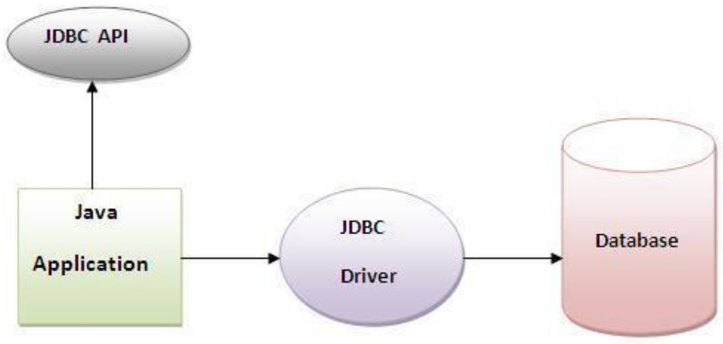
Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- **users@tomcat.apache.org** for general questions related to configuring and using Tomcat

DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example

JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

What is API?

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

JDBC Architecture

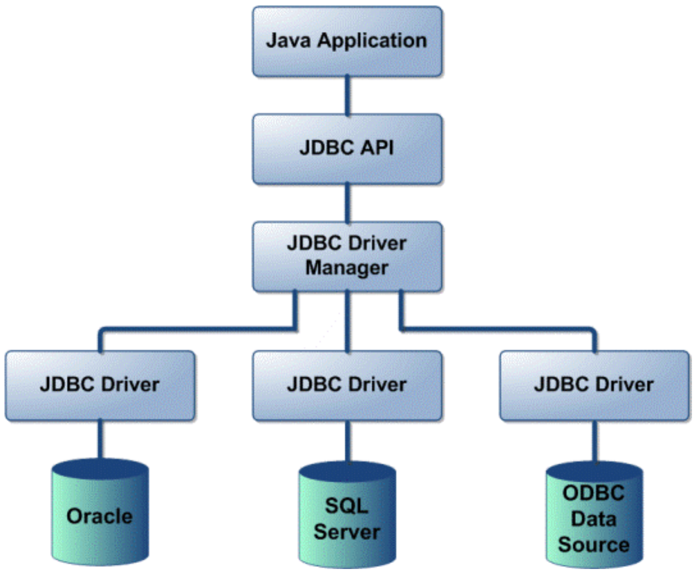
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException:** This class handles any errors that occur in a database application.

JDBC Driver is a software component that enables java application to interact with the database.There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

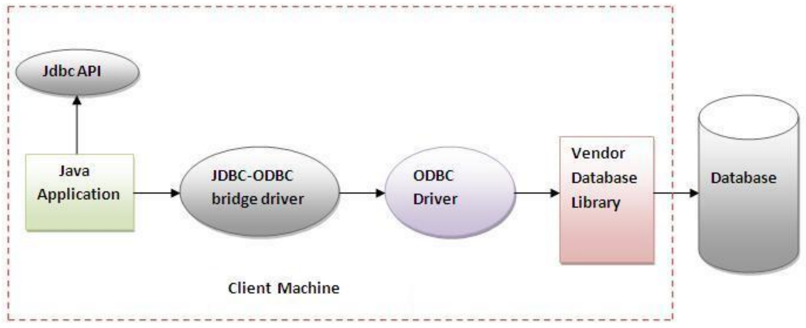


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

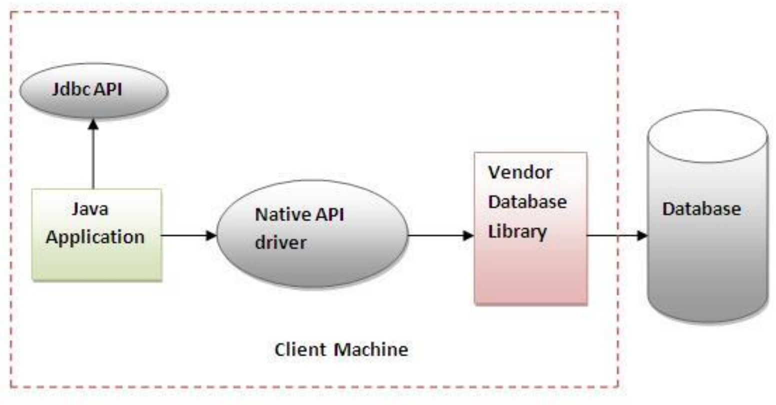


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

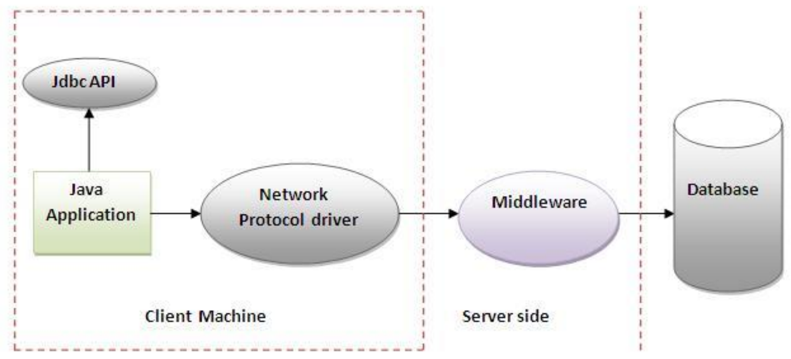


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

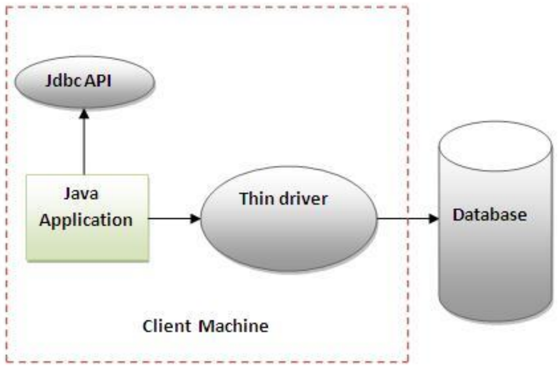


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

EXAMPLE PROGRAM ON SERVLET WITH DATABASE CONNECTIVITY

- Download Jar and save to lib folder of Apache (ojdbc14-1.0.jar Required):
- Update Classpath to following (Path in Bold are new): C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar;C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\jsp-api.jar;C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\ojdbc14-1.0.jar;**C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\ com.mysql.jdbc_5.1.5.jar;**.
- Download and Install MySQL.

First Create Your Database and Tables

Table Name: student

FieldName	**Type**	
Sid	number(10)	Primary Key

Sname	varchar2(20)	

Hello JDBC MySQL using Servlet Example

FilePath **: C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\example03\WEB-INF\classes**

ex03.java

```
importjava.io.IOException;
importjava.io.PrintWriter;
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.ResultSet;
importjava.sql.Statement;
importjavax.servlet.ServletException;
importjavax.servlet.http.HttpServlet;
importjavax.servlet.http.HttpServletRequest;
importjavax.servlet.http.HttpServletResponse;

public class ex03 extends HttpServlet
{
   protected void doGet(HttpServletRequestreq,HttpServletResponse
res)throws ServletException,IOException
   {
       PrintWriter pw=res.getWriter();
       res.setContentType("text/html");
       String tb="student";
       pw.println("Initializing Connection....");

       try
       {

             Class.forName("com.mysql.jdbc.Driver");
                Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","adars
h","patel");
            Statement st=con.createStatement();
             pw.println("connection established successfully...!!");
             ResultSetrs=st.executeQuery("Select * from "+tb);
             pw.println("<table border=1>");

                while(rs.next())
                {

pw.println("<tr><td>"+rs.getInt(1)+"</td><td>"+rs.getString(2)+"</td>"+"<
/tr>");
                }
             pw.println("</table>");
             pw.close();
```

```
        }
        catch (Exception e)
        {
           pw.println("The error is:" + e.getMessage());
        }    }}
```

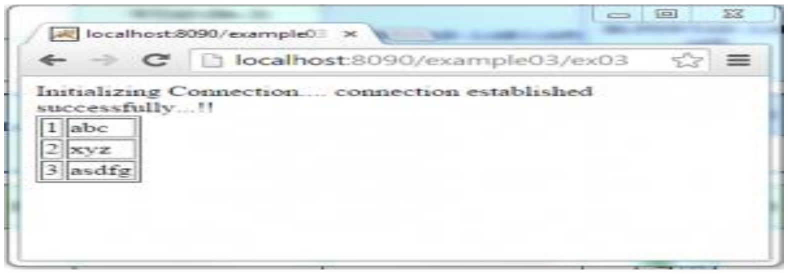
Hello JDBC using Servlet Example Explanation:

Line :Class.forName(“com.mysql.jdbc.Driver “);
Explanation : This Line if to load Driver for MySQL Database Connectivity with Java Servlet.

Line: Connection con=DriverManager.getConnection(“jdbc:mysql://localhost:3306/test”,”root”,””);
Explanation : This Line is most important as it contains userid and password of MySQL. Here I have used userid :adarsh and password : patel. Only 2 Fields you need to change is userid and password. (Default userid for MySQL is root and password is blank);
FilePath **: C:\Program Files\ApacheSoftwareFoundation\Tomcat 7.0\webapps\example03\WEB-INF\**

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
<display-name>Example 02 </display-name>
<description> JDBC Using Servlet </description>
  <servlet>
  <servlet-name>ex03</servlet-name>
  <servlet-class>ex03</servlet-class>
</servlet>
 <servlet-mapping>
  <servlet-name>ex03</servlet-name>
    <url-pattern>/ex03</url-pattern>
</servlet-mapping></web-app>
```



```
Initializing Connection.... connection established
successfully...!!
1 abc
2 xyz
3 asdfg
```

UNIT III
SERVER SIDE PROGRAMMING

JSP: Understanding Java Server Pages-JSP Standard Tag Library(JSTL)-Creating HTML forms by embedding JSP code.

JSP DEFINITION:

- Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.
- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
- A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

1) Extension to Servlet
JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP which makes JSP development easy.

2) **Easy to maintain**
JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy
If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Why Use JSP?

- JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.
- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

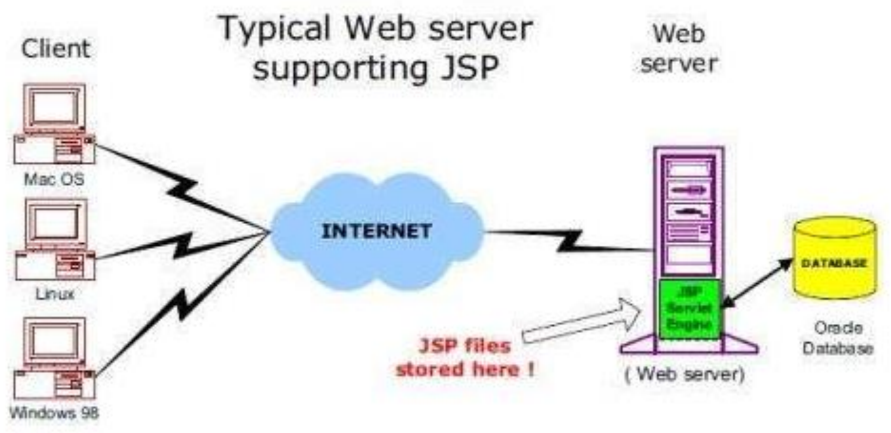
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

JSP ARCHITECTURE

The web server needs a JSP engine ie. container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web Application.



PROCESSING OF JSP

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.

- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jspinstead of .html.

- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

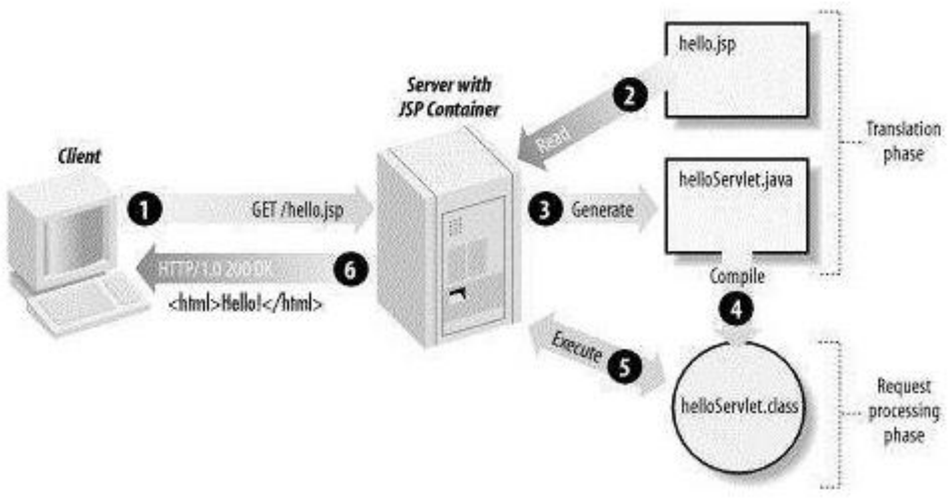
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.

- The web server forwards the HTTP response to your browser in terms of static HTML content.

- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be shown below in the following diagram:



- Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet.
- If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents.
- This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.
- So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz.
- Except for the translation phase, a JSP page is handled exactly like a regular servlet.

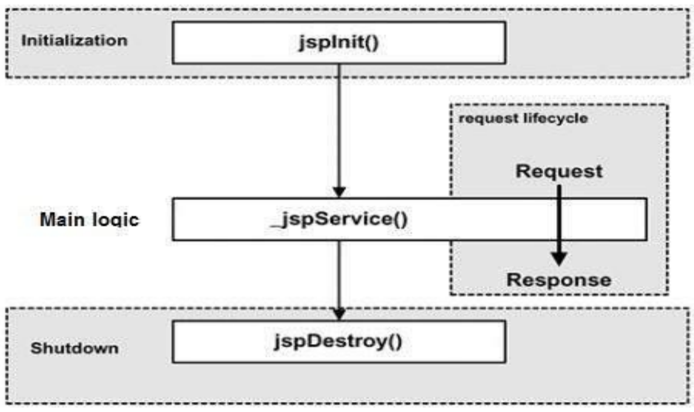
LIFECYCLE OF JSP

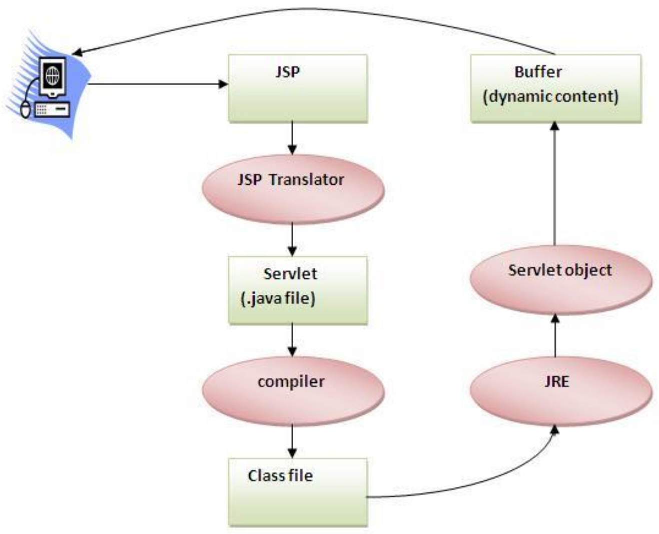
A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

The following are the paths followed by a JSP

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of JSP life cycle are very similar to Servlet Life Cycle and they are as follows:





JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
publicvoid jspInit(){

// Initialization code...

}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

```
void _jspService(HttpServletRequest request,

HttpServletResponse response)

{

// Service handling code...

}
```

The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

```
publicvoid jspDestroy()

{

// Your cleanup code goes here.

}
```

CONTROL-FLOW STATEMENTS:

- JSP provides full power of Java to be embedded in your web application.
- You can use all the APIs and building blocks of Java in your JSP programming including decision making statements, loops etc.

Decision-Making Statements:

- The if...else block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

```
<%!int day =3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<%if(day ==1| day ==7){ %>
<p> Today is weekend</p>
<%}else{ %>
<p> Today is not weekend</p>
<%} %>
</body>
</html>
```

This would produce following result:

```
Today is not weekend
```

Switch Case:

Now look at the following **switch...case** block which has been written a bit differentlty using**out.println()** and inside Scriptletas:

```
<%!int day =3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day){
case0:
out.println("It\'s Sunday.");
break;
case1:
out.println("It\'s Monday.");
break;
case2:
out.println("It\'s Tuesday.");
break;
case3:
out.println("It\'s Wednesday.");
break;
case4:
out.println("It\'s Thursday.");
break;
case5:
out.println("It\'s Friday.");
break;
default:
out.println("It's Saturday.");
}
%>
</body>
</html>
```

This would produce following result:

```
It's Wednesday.
```

Loop Statements:

- You can also use three basic types of looping blocks in Java: **for, while,and do…while**blocks in your JSP programming.

Let us look at the following **for** loop example:

```
<%!int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for( fontSize =1; fontSize <=3; fontSize++){ %>
<font color="green" size="<%= fontSize %>">
   JSP Tutorial
</font><br/>
<%}%>
</body>
</html>
```

This would produce following result:

```
   JSP Tutorial


   JSP Tutorial


   JSP Tutorial
```

Above example can be written using **while** loop as follows:

```
<%!int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while( fontSize <=3){ %>
<font color="green" size="<%= fontSize %>">
   JSP Tutorial
</font><br/>
<%fontSize++;%>
<%}%>
</body>
</html>
```

This would also produce following result:

```
JSP Tutorial


JSP Tutorial


JSP Tutorial
```

JSP Operators:

- JSP supports all the logical and arithmetic operators supported by Java.
- Following table give a list of all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.
- Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>>>>>><<	Left to right
Relational	>>= <<=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left

Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

JSP Literals:

The JSP expression language defines the following literals:

- **Boolean:** true and false
- **Integer:** as in Java
- **Floating point:** as in Java
- **String:** with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\.
- **Null:** null
- **Null:** null

SCRIPTLET:

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet:

<% code fragment %>

You can write XML equivalent of the above syntax as follows:

<jsp:scriptlet>

code fragment

</jsp:scriptlet>

JSP Scriptlet tag (Scripting elements)

1. Scripting elements
2. JSP scriptlet tag
3. Simple Example of JSP scriptlet tag
4. Example of JSP scriptlet tag that prints the user name

In JSP, java code can be written inside the jsp page using the scriptlet tag.

JSP Scripting elements

The scripting elements provide the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag

- declaration tag

<u>JSP scriptlet tag</u>

- A scriptlet tag is used to execute java source code in JSP.
- Syntax is as follows:

```
<%  java source code %>
```

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

- In this example, we have created two files index.html and welcome.jsp.
- The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

<u>JSP expression tag</u>

- The code placed within JSP expression tag is *written to the output stream of the response*.

- So you need not write out.print() to write data.
- It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
```

```
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP Declaration Tag

- The JSP declaration tag is used *to declare fields and methods*.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
- So it doesn't get memory at each request.

Syntax of JSP declaration tag

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

- In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag.
- But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
```

```
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Comments:

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

Following is the **syntax** of JSP comments:

<%-- This is JSP comment --%>

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary:

Syntax	Purpose
<%-- comment --%>	A JSP comment. Ignored by the JSP engine.
<!-- comment -->	An HTML comment. Ignored by the browser.
<\%	Represents static <% literal.
%\>	Represents static %> literal.
\'	A single quote in an attribute that uses single quotes.
\"	A double quote in an attribute that uses double quotes.

JSP DIRECTIVES:

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language,
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the

1. page directive

The page directive defines attributes that apply to an entire JSP page.

<%@ page attribute="value" %>

2. include Directive:

- The **include** directive is used to includes a file during the translation phase.
- This directive tells the container to merge the content of other external files with the current JSP during the translation phase.
- You may code *include* directives anywhere in your JSP page.

The general usage form of this directive is as follows:

```
<%@ include file="relative url">
```

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

3. taglib Directive:

- The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.

The taglib directive follows the following syntax:

```
<%@ taglib uri="uri" prefix="prefixOfTag">
```

Where the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

<u>JSP ACTIONS:</u>

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard:

<jsp:action_name attribute="value" />

Action elements are basically predefined functions and there are following JSP actions available:

Syntax	**Purpose**
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

JSP IMPLICIT OBJECTS:

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are:

Objects	**Description**
request	This is the **HttpServletRequest** object associated with the request.
response	This is the **HttpServletResponse** object associated with the response to the client.
out	This is the **PrintWriter** object used to send output to the client.
session	This is the **HttpSession** object associated with the request.
application	This is the **ServletContext** object associated with application context.
config	This is the **ServletConfig** object associated with the page.
pageContext	This encapsulates use of server-specific features like higher
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
Exception	The **Exception** object allows the exception data to be accessed by designated JSP.

JSTL (JSP STANDARD TAG LIBRARY)

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions

Core Tags:

The core group of tags are the most frequently used JSTL tags. Following is the syntax to include JSTL Core library in your JSP:

```
<%@ taglib prefix="c"
        uri="http://java.sun.com/jsp/jstl/core" %>
```

There are following Core JSTL Tags:

Tag	Description
<c:out >	Like <%= ... >, but for expressions.
<c:set >	Sets the result of an expression evaluation in a 'scope'
<c:remove >	Removes a scoped variable (from a particular scope, if specified).
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evalutes its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<c:when>	Subtag of <choose> that includes its body if its condition evalutes to 'true'.
<c:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.
<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
<c:forEach >	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .

<c:forTokens>	Iterates over tokens, separated by the supplied delimeters.
<c:param>	Adds a parameter to a containing 'import' tag's URL.
<c:redirect >	Redirects to a new URL.
<c:url>	Creates a URL with optional query parameters

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title><c:out> Tag Example</title>
</head>
<body>
<c:out value="${'<tag> , &'}"/>
</body>
</html>
```

This would produce following result:

```
<tag> , &
```

```
<body>
<c:setvar="salary"scope="session"value="${2000*2}"/>
<c:outvalue="${salary}"/>
</body>
```

This would produce following result:

```
4000
```

```
<body>
<c:setvar="salary"scope="session"value="${2000*2}"/>
<p>Before Remove Value: <c:outvalue="${salary}"/></p>
<c:removevar="salary"/>
<p>After Remove Value: <c:outvalue="${salary}"/></p>
</body>
```

This would produce following result:

```
Before Remove Value: 4000
After Remove Value:
```

```
<body>
<c:setvar="salary"scope="session"value="${2000*2}"/>
<c:iftest="${salary > 2000}">
<p>My salary is: <c:outvalue="${salary}"/><p>
</c:if>
</body>
```

This would produce following result:

```
My salary is: 4000
```

Formatting tags:

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites. Following is the syntax to include Formatting library in your JSP:

```
<%@ taglib prefix="fmt"
        uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Following is the list of Formatting JSTL Tags:

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern
<fmt:parseDate>	Parses the string representation of a date and/or time
<fmt:bundle>	Loads a resource bundle to be used by its tag body.

<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>


<html>

<head>

<title>JSTL fmt:formatNumber Tag</title>

</head>

<body>

<h3>Number Format:</h3>

<c:setvar="balance"value="120000.2309"/>

<p>Formatted Number (1): <fmt:formatNumbervalue="${balance}"

type="currency"/></p>

<p>Formatted Number (2): <fmt:formatNumbertype="number"

maxIntegerDigits="3"value="${balance}"/></p>

</body>

</html>
```

This would produce following result:

```
Number Format:
```

```
Formatted Number (1): £120,000.23
Formatted Number (2): 000.231
```

SQL tags:

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

Following is the syntax to include JSTL SQL library in your JSP:

```
<%@ taglib prefix="sql"
        uri="http://java.sun.com/jsp/jstl/sql" %>
```

Following is the list of SQL JSTL Tags:

Tag	Description
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
<sql:param>	Sets a parameter in an SQL statement to the specified value.
<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<sql:transaction >	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>

<head>

<title>JSTL sql:setDataSource Tag</title>

</head>

<body>
```

```
<sql:setDataSourcevar="snapshot"driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/TEST"
user="user_id"password="mypassword"/>

<sql:querydataSource="${snapshot}"sql="..."var="result"/>

</body>
</html>
```

XML tags:

The JSTL XML tags provide a JSP-centric way of creating and manipulating XML documents. Following is the syntax to include JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with XML data. This includes parsing XML, transforming XML data, and flow control based on XPath expressions.

```
<%@ taglib prefix="x"
        uri="http://java.sun.com/jsp/jstl/xml" %>
```

Before you proceed with the examples, you would need to copy following two XML and XPath related libraries into your <Tomcat Installation Directory>\lib:

- XercesImpl.jar: Download it fromhttp://www.apache.org/dist/xerces/j/
- xalan.jar: Download it from http://xml.apache.org/xalan-j/index.html

Following is the list of XML JSTL Tags:

Tag	Description
<x:out>	Like <%= ... >, but for XPath expressions.
<x:parse>	Use to parse XML data specified either via an attribute or in the tag body.
<x:set >	Sets a variable to the value of an XPath expression.
<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.

<x:forEach>	To loop over nodes in an XML document.
<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<x:when >	Subtag of <choose> that includes its body if its expression evalutes to 'true'
<x:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<x:transform >	Applies an XSL transformation on a XML document
<x:param >	Use along with the transform tag to set a parameter in the XSLT stylesheet

JSTL Functions:

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP:

```
<%@ taglib prefix="fn"
        uri="http://java.sun.com/jsp/jstl/functions" %>
```

Following is the list of JSTL Functions:

Function	Description
fn:contains()	Tests if an input string contains the specified substring.
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
fn:endsWith()	Tests if an input string ends with the specified suffix.
fn:escapeXml()	Escapes characters that could be interpreted as XML markup.

fn:indexOf()	Returns the index withing a string of the first occurrence of a specified substring.
fn:join()	Joins all elements of an array into a string.
fn:length()	Returns the number of items in a collection, or the number of characters in a string.
fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
fn:split()	Splits a string into an array of substrings.
fn:startsWith()	Tests if an input string starts with the specified prefix.
fn:substring()	Returns a subset of a string.
fn:substringAfter()	Returns a subset of a string following a specific substring.
fn:substringBefore()	Returns a subset of a string before a specific substring.
fn:toLowerCase()	Converts all of the characters of a string to lower case.
fn:toUpperCase()	Converts all of the characters of a string to upper case.
fn:trim()	Removes white spaces from both ends of a string.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<html>

<head>

<title>Using JSTL Functions</title>
```

```
</head>
<body>


<c:setvar="string1"value="This is first String."/>
<c:setvar="string2"value="This is second String."/>


<p>Length of String (1) : ${fn:length(string1)}</p>
<p>Length of String (2) : ${fn:length(string2)}</p>


</body>
</html>
```

This would produce following result:

```
Length of String (1) : 21
Length of String (2) : 22
```

```
<body>


<c:setvar="string1"value="This is first String."/>
<c:setvar="string2"value="${fn:substring(string1, 5, 15)}"/>


<p>Final sub string : ${string2}</p>


</body>
```

This would produce following result:

```
Final sub string : is first S
```

```
<body>


<c:setvar="string1"value="This is first String."/>
<c:setvar="string2"value="${fn:toLowerCase(string1)}"/>


<p>Final string : ${string2}</p>
```

```
</body>
```

This would produce following result:

```
Final string : this is first string.
```

```
<body>
<c:setvar="string1"value="This is first String."/>
<c:setvar="string2"value="${fn:toUpperCase(string1)}"/>
<p>Final string : ${string2}</p>
</body>
```

This would produce following result:

```
Final string : THIS IS FIRST STRING.
```

CREATING HTML FORM BY EMBEDDING JSP CODE:

JSP handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.
- **getInputStream():** Call this method to read binary data stream coming from the client.

GET Method Example Using Form:

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same JSP main.jsp to handle this input.

```
<html>
<body>
<formaction="main.jsp"method="GET">
First Name: <inputtype="text"name="first_name">
<br/>
Last Name: <inputtype="text"name="last_name"/>
```

```
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Keep this HTML in a file Hello.html and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access*http://localhost:8080/Hello.html*, here is the actual output of the above form.

First Name:
Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

GET Method Example Using URL:

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI

Below is **main.jsp** JSP program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information:

```
<html>
<head>
<title>Using GET Method to Read Form Data</title>
</head>
<body>
<center>
<h1>Using GET Method to Read Form Data</h1>
<ul>
<li><p><b>First Name:</b>
<%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
<%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

Now type *http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI* in your browser's Location:box. This would generate following result:

Using GET Method to Read Form Data

- **First Name**: ZARA
- **Last Name**: ALI

POST Method Example Using Form:

- There is no change in above JSP because only way of passing parameters is changed and no binary data is being passed to the JSP program.

```
<html>
<head>
<title>Using GET and POST Method to Read Form Data</title>
</head>
<body>
<center>
<h1>Using GET Method to Read Form Data</h1>
<ul>
<li><p><b>First Name:</b>
<%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last  Name:</b>
<%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

Following is the content of **Hello.html** file:

```
<html>
<body>
```

```
<formaction="main.jsp"method="POST">
First Name: <inputtype="text"name="first_name">
<br/>
Last Name: <inputtype="text"name="last_name"/>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

Now let us keep main.jsp and hello.htm in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access*http://localhost:8080/Hello.html*, below is the actual output of the above form.

First Name: ☐

Last Name: ☐

Passing Checkbox Data to JSP Program

- Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
<body>
<formaction="main.jsp"method="POST"target="_blank">
<inputtype="checkbox"name="maths"checked="checked"/> Maths
<inputtype="checkbox"name="physics"/> Physics
<inputtype="checkbox"name="chemistry"checked="checked"/>  Chemistry
<inputtype="submit"value="Select Subject"/>
</form>
</body>
</html>
```

The result of this code is the following form

☑ Maths ☐ Physics ☑ Chemistry

Below is main.jsp JSP program to handle input given by web browser for checkbox button.

```
<html>
<head>
<title>Reading Checkbox Data</title>
</head>
<body>
<center>
<h1>Reading Checkbox Data</h1>
<ul>
<li><p><b>Maths Flag:</b>
<%= request.getParameter("maths")%>
</p></li>
<li><p><b>Physics Flag:</b>
<%= request.getParameter("physics")%>
</p></li>
<li><p><b>Chemistry Flag:</b>
<%= request.getParameter("chemistry")%>
</p></li>
</ul>
</body>
</html>
```

For the above example, it would display following result:

```
                    Reading Checkbox Data

  • Maths Flag : : on

  • Physics Flag: : null

  • Chemistry Flag: : on
```

Reading All Form Parameters:

- Following is the generic example which uses getParameterNames() method of HttpServletRequest to read all the available form parameters.
- This method returns an Enumeration that contains the parameter names in an unspecified order.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>HTTP Header Request Example</title>
</head>
<body>
<center>
<h2>HTTP Header Request Example</h2>
<tablewidth="100%"border="1"align="center">
<trbgcolor="#949494">
<th>Param Name</th><th>Param Value(s)</th>
</tr>
<%
Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements()){
String paramName =(String)paramNames.nextElement();
out.print("<tr><td>"+ paramName +"</td>\n");
String paramValue = request.getHeader(paramName);
out.println("<td> "+ paramValue +"</td></tr>\n");
}
%>
</table>
</center>
</body>
</html>
```

Following is the content of Hello.html:

```
<html>
```

```
<body>
<formaction="main.jsp"method="POST"target="_blank">
<inputtype="checkbox"name="maths"checked="checked"/> Maths
<inputtype="checkbox"name="physics"/> Physics
<inputtype="checkbox"name="chemistry"checked="checked"/> Chem
<inputtype="submit"value="Select Subject"/>
</form>
</body>
</html>
```

Now try calling JSP using above Hello.htm, this would generate a result something like as below based on the provided input:

Reading All Form Parameters

Param Name	Param Value(s)
maths	on
chemistry	on

4
PHP and XML 8

4.1 INTRODUCTION TO PHP

The PHP Hypertext Pre-processor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases

PHP is basically used for developing web based software applications.PHP is probably the most popular scripting language on the web. It is used to enhance web pages.PHP is known as a **server-sided language.** That is because the PHP doesn't get executed on the client’s computer, but on the computer the user had requested the page from. The results are then handed over to client, and then displayed in the browser.

Features of PHP:

- PHP is a server side scripting language that is embedded in HTML
- PHP was originally developed by the **Danish Greenlander Rasmus Lerdorf**, and was subsequently developed as open source.
- It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP supports a large number of protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP language tries to be as forgiving as possible.
- PHP syntax is C-Like.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.

- ❖ PHP can handle forms, i.e. gather data from files, save data to a file, through email the user can send data, return data to the user.
- ❖ The user can add, delete, and modify elements within the database through PHP.
- ❖ They can access cookies variables and set cookies.
- ❖ Using PHP, the user can restrict users to access some pages of the website.
- ❖ It can encrypt data.

Working of PHP

When the client requests a PHP page residing on the server, the server first performs the operations mentioned by the PHP code of the page. Then is sends the output of the PHP page in HTML format. So when the user views the source code of the page, it will be full of HTML tags. All the work is done at the server side.

4.2 PROGRAMMING WITH PHP

➢ **Comments**

A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

- **Single-line comments:** They are generally used for short explanations or notes relevant to the local code.
- **Multi-line comments:** They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C.

Rules of PHP

- ❖ PHP is white space insensitive
- ❖ PHP is case sensitive
- ❖ Statements are expressions terminated by semicolons
- ❖ Expressions are combinations of tokens
- ❖ Braces make blocks

➢ **PHP Variable Types**

All variables in PHP are denoted with a leading dollar sign ($). The value of a variable is the value of its most recent assignment. Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.

Variables in PHP do not have **intrinsic types (data types) -** a variable does not know in advance whether it will be used to store a number or a string of characters.

Variables used before they are assigned have default values. PHP automatically converts types from one to another when necessary. PHP has a total of eight data types:

- Integers are whole numbers, without a decimal point. They can be in decimal, octal or hexadecimal. Eg: 87.
- Doubles are floating-point numbers. Eg: 3.87
- Booleans have only two possible values either true or false.
- NULL is a special type that only has one value: NULL
- Strings are sequences of characters
- Arrays are named and indexed collections of other values.
- Objects are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- Resources are special variables that hold references to resources external to PHP (such as database connections).
- The first five are simple types, and the arrays and objects are compound types.
- The compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

Variable Scope

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types: Local variables, Function parameters, Global variables and Static variables.

Variable Naming

Rules for naming a variable are:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but the user cannot use characters like + , - , % , (,) . & , etc

PHP Constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. To define a constant , use define() function and retrieve the value of a constant. The function constant() is used to read a constant's value .

define(name, value, case-insensitive)

The name specifies the name of the constant, value: Specifies the value of the constant and case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

Differences between constants and variables in PHP

Constants in PHP	Variables in PHP
No $ sign before constants.	$ sign is present before variables.
Constants are defined using define().	Variables are defined using assignment statement.
Constants may be defined and accessed anywhere without any regard.	Variables follow certain scope.
Constants cannot be redefined or undefined.	They can be redefined.

Pre-defined constants

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances
__FUNCTION__	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

➢ Echo and Print statements

Differences between echo and print statement

echo	print
This does not have return value.	This has a return value of 1.
This cannot be used in expressions.	This can be used in expressions.
This can take multiple parameters.	This can take only one parameter.
This is slightly faster than print.	This is comparatively slower.

➢ Operators

Operators are used to perform operations on variables and values.PHP divides the operators in the following groups: Arithmetic operators, Assignment operators, Comparison operators, Increment/Decrement operators, Logical operators, String operators and Array operators.

PHP CONTROL STATMENTS

Decision Making Statements

The if, else if ...else and switch statements are used to take decision based on the different condition.

- if statement - executes some code only if a specified condition is true.
- if...else statement - executes some code if a condition is true and another code if the condition is false.
- if...else if....else statement - specifies a new condition to test, if the first condition is false
- switch statement - selects one of many blocks of code to be executed

➢ if statement

if statement

```
<?php
$t = date("H");
if ($t < "20")  //This program outputs the echo statement if the hour is <20
{
   echo "Have a good day!";
}
?>
```

- **if-else statement**

  Use the if....else statement to execute some code if a condition is true and another code if the condition is false.

if else statement

```
<?php
$t = date("H");
if ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

- **if-else ladder**

  Use the if....else if...else statement to specify a new condition to test, if the first condition is false.

if-else ladder

```
<?php
$t = date("H");
if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

- **switch statement**

  To select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code. The switch statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the labels match then statement will execute any specified default code.

Switch statement

```
<html>
<body>
<?php $d=date("D");
switch ($d)
 {
 case "Mon":
   echo "Today is Monday";
   break;
case "Tue":
   echo "Today is Tuesday";
    break;
case "Wed":
   echo "Today is Wednesday";
    break;
case "Thu":
    echo "Today is Thursday";
     break;
case "Fri":
      echo "Today is Friday";
      break;
case "Sat":
     echo "Today is Saturday";
     break;
case "Sun":
     echo "Today is Sunday";
    break;
default:
     echo "Wonder which day is this ?";
}
?>
</body></html>
```

Looping Statements

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types

- for : loops through a block of code a specified number of times.
- while: loops through a block of code if and as long as a specified condition is true.
- do...while: loops through a block of code once, and then repeats the loop as long as a special condition is true.
- foreach: loops through a block of code for each element in an array.

➢ **For loop**

for loop

```
<html>
<body>
<?php
$a = 0;
$b = 0;
for( $i=0; $i<5; $i++ )
{
 $a += 10;
$b += 5;
}
echo ("At the end of the loop a=$a and b=$b" );
?>
</body></html>
```

```
At the end of the loop a=50 and b=25
```

➢ **While loop**

The while statement will execute a block of code as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

While loop

```
<html> <body>
<?php
$i = 0;
$num = 50;
while( $i < 10)
{
$num--;
$i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?> </body></html>
```

```
Loop stopped at i = 1 and num = 40
```

- **Do…while loop**

```
<html> <body>
<?php
$i = 0;
$num = 0;
do {
    $i++;
}while( $i < 10 );
echo ("Loop stopped at i = $i" );
?></body></html>
```

```
Loop stopped at i = 10
```

- **For Each loop**

The for each statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed**.**

```
<html> <body>
<?
php $array = array( 11, 12, 13,14, 15);
```

```
foreach( $array as $value )
{
  echo "Value is $value <br />";
}  ?> </body></html>
```

```
Value is 11
Value is 12
Value is 13
Value is 14
Value is 15
```

- **Break statement**

  The PHP break keyword is used to terminate the execution of a loop prematurely. The break statement is situated inside the statement blockAfter coming out of a loop immediate statement to the loop will be executed**.**

- **Continue statement**

  The PHP continue keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering continue statement, rest of the loop code is skipped and next pass starts.

FUNCTIONS

A function is a block of statements that can be used repeatedly in a program. A function will not execute immediately when a page loads. A function will be executed by a call to the function. There are two types of functions: Built-in functions and User defined functions

User defined Functions

A user defined function declaration starts with the word function. Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses.

```
<?php
function sum($x, $y) {
   $z = $x + $y;
   return $z; }
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";

echo "2 + 4 = " . sum(2, 4); ?>
```

```
5 + 10 = 15

7 + 13 = 20

2 + 4 = 6
```

Built-in functions

Array functions

Function	Description
array()	Creates an array
array_chunk()	Splits an array into chunks of arrays
array_column()	Returns the values from a single column in the input array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays, and returns the differences (compare values only)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the values of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays, and returns the matches (compare values only)
array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_multisort()	Sorts multiple or multi-dimensional arrays

array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user-defined function
array_replace()	Replaces the values of the first array with the values from following arrays
array_replace_recursive()	Replaces the values of the first array with the values from following arrays recursively
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array
array_sum()	Returns the sum of the values in an array
array_udiff()	Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function)
array_uintersect()	Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function)
array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
array_values()	Returns all the values of an array
array_walk()	Applies a user function to every member of an array
arsort()	Sorts an associative array in descending order, according to the value
asort()	Sorts an associative array in ascending order, according to the value
compact()	Create array containing variables and their values

count()	Returns the number of elements in an array
current()	Returns the current element in an array
each()	Returns the current key and value pair from an array
end()	Sets the internal pointer of an array to its last element
extract()	Imports variables into the current symbol table from an array
in_array()	Checks if a specified value exists in an array
key()	Fetches a key from an array
list()	Assigns variables as if they were an array
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm
natsort()	Sorts an array using a "natural order" algorithm
next()	Advance the internal array pointer of an array
prev()	Rewinds the internal array pointer
range()	Creates an array containing a range of elements
reset()	Sets the internal pointer of an array to its first element
rsort()	Sorts an indexed array in descending order
shuffle()	Shuffles an array
sort()	Sorts an indexed array in ascending order
uasort()	Sorts an array by values using a user-defined comparison function
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array using a user-defined comparison function

- **Calendar Functions**

The calendar extension contains functions that simplify converting between different calendar formats. It is based on the Julian Day Count, which is a count of days starting from January 1st, 4713 B.C. To convert between calendar formats, first convert to Julian Day Count, then to the calendar of the user's choice.

Function	Description
cal_days_in_month()	Returns the number of days in a month for a specified year and calendar
cal_from_jd()	Converts a Julian Day Count into a date of a specified calendar

cal_info()	Returns information about a specified calendar
cal_to_jd()	Converts a date in a specified calendar to Julian Day Count
easter_date()	Returns the Unix timestamp for midnight on Easter of a specified year
easter_days()	Returns the number of days after March 21, that the Easter Day is in a specified year
gregoriantojd()	Converts a Gregorian date to a Julian Day Count
jddayofweek()	Returns the day of the week
jdmonthname()	Returns a month name
jdtogregorian()	Converts a Julian Day Count to a Gregorian date
jdtounix()	Converts Julian Day Count to Unix timestamp
jewishtojd()	Converts a Jewish date to a Julian Day Count
juliantojd()	Converts a Julian date to a Julian Day Count
unixtojd()	Converts Unix timestamp to Julian Day Count

- **Date Functions**

  The date/time functions allow to get the date and time from the server on which PHP script runs. These functions depend on the locale settings of the server. Remember to take daylight saving time and leap years into consideration when working with these functions.

Function	Description
checkdate()	Validates a Gregorian date
date_add()	Adds days, months, years, hours, minutes, and seconds to a date
date_create_from_format()	Returns a new DateTime object formatted according to a specified format
date_create()	Returns a new DateTime object
date_date_set()	Sets a new date
date_default_timezone_get()	Returns the default timezone used by all date/time functions
date_default_timezone_set()	Sets the default timezone used by all date/time functions

date_diff()	Returns the difference between two dates
date_format()	Returns a date formatted according to a specified format
date_interval_format()	Formats the interval
date_isodate_set()	Sets the ISO date
date_modify()	Modifies the timestamp
date_parse()	Returns an associative array with detailed info about a specified date
date_sub()	Subtracts days, months, years, hours, minutes, and seconds from a date
date_sun_info()	Returns an array containing info about sunset/sunrise and twilight begin/end, for a specified day and location
date_sunrise()	Returns the sunrise time for a specified day and location
date_sunset()	Returns the sunset time for a specified day and location
date_time_set()	Sets the time
date()	Formats a local date and time
getdate()	Returns date/time information of a timestamp or the current local date/time
gettimeofday()	Returns the current time
gmdate()	Formats a GMT/UTC date and time
gmmktime()	Returns the Unix timestamp for a GMT date
gmstrftime()	Formats a GMT/UTC date and time according to locale settings
idate()	Formats a local time/date as integer
localtime()	Returns the local time
microtime()	Returns the current Unix timestamp with microseconds
mktime()	Returns the Unix timestamp for a date

strftime()	Formats a local time and/or date according to locale settings
time()	Returns the current time as a Unix timestamp
timezone_name_get()	Returns the name of the timezone
timezone_offset_get()	Returns the timezone offset from GMT
timezone_open()	Creates new DateTimeZone object
timezone_version_get()	Returns the version of the timezone db

➢ **Directory functions**

The directory function allows retrieving information about directories and their contents.

Function	Description
chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle
readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

➢ **Error handling functions**

The error functions are used to deal with error handling and logging. The error functions allow us to define own error handling rules, and modify the way the errors can be logged. The logging functions allow us to send messages directly to other machines, emails, or system logs. The error reporting functions allow us to customize what level and kind of error feedback is given.

Function	Description
debug_backtrace()	Generates a backtrace
debug_print_backtrace()	Prints a backtrace
error_get_last()	Returns the last error that occurred
error_log()	Sends an error message to a log, to a file, or to a mail account
error_reporting()	Specifies which errors are reported
restore_error_handler()	Restores the previous error handler
restore_exception_handler()	Restores the previous exception handler
set_error_handler()	Sets a user-defined error handler function
set_exception_handler()	Sets a user-defined exception handler function
trigger_error()	Creates a user-level error message
user_error()	Alias of trigger_error()
debug_backtrace()	Generates a backtrace

- **File system Functions**

The file system functions allow the user to access and manipulate the file system.

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a directory
disk_total_space()	Returns the total size of a directory

fclose()	Closes an open file
feof()	Tests for end-of-file on an open file
fflush()	Flushes buffered output to an open file
fgetc()	Returns a character from an open file
fgets()	Returns a line from an open file
fgetss()	Returns a line, with HTML and PHP tags removed, from an open file
file()	Reads a file into an array
file_exists()	Checks whether or not a file or directory exists
file_get_contents()	Reads a file into a string
file_put_contents()	Writes a string to a file
fileatime()	Returns the last access time of a file
filectime()	Returns the last change time of a file
filegroup()	Returns the group ID of a file
fileinode()	Returns the inode number of a file
filemtime()	Returns the last modification time of a file
fileowner()	Returns the user ID (owner) of a file
fileperms()	Returns the permissions of a file
filesize()	Returns the file size
filetype()	Returns the file type
flock()	Locks or releases a file
fnmatch()	Matches a filename or string against a specified pattern
fopen()	Opens a file or URL
fpassthru()	Reads from an open file, until EOF, and writes the result to the output buffer
fputcsv()	Formats a line as CSV and writes it to an open file
fread()	Reads from an open file
fscanf()	Parses input from an open file according to a specified format

fseek()	Seeks in an open file
fstat()	Returns information about an open file
ftell()	Returns the current position in an open file
ftruncate()	Truncates an open file to a specified length
fwrite()	Writes to an open file
glob()	Returns an array of filenames / directories matching a specified pattern
is_dir()	Checks whether a file is a directory
is_executable()	Checks whether a file is executable
is_file()	Checks whether a file is a regular file
is_link()	Checks whether a file is a link
is_readable()	Checks whether a file is readable
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST
is_writable()	Checks whether a file is writeable
lchgrp()	Changes group ownership of symlink
lchown()	Changes user ownership of symlink
link()	Creates a hard link
linkinfo()	Returns information about a hard link
lstat()	Returns information about a file or symbolic link
mkdir()	Creates a directory
move_uploaded_file()	Moves an uploaded file to a new location
pathinfo()	Returns information about a file path
pclose()	Closes a pipe opened by popen()
popen()	Opens a pipe
readfile()	Reads a file and writes it to the output buffer
readlink()	Returns the target of a symbolic link
realpath()	Returns the absolute pathname
realpath_cache_get()	Returns realpath cache entries

realpath_cache_size()	Returns realpath cache size
rename()	Renames a file or directory
rewind()	Rewinds a file pointer
rmdir()	Removes an empty directory
set_file_buffer()	Sets the buffer size of an open file
stat()	Returns information about a file
symlink()	Creates a symbolic link
touch()	Sets access and modification time of a file
umask()	Changes file permissions for files
unlink()	Deletes a file

- **Math functions**

Function	Description
abs()	Returns the absolute (positive) value of a number
acos()	Returns the arc cosine of a number
acosh()	Returns the inverse hyperbolic cosine of a number
asin()	Returns the arc sine of a number
asinh()	Returns the inverse hyperbolic sine of a number
atan()	Returns the arc tangent of a number in radians
atan2()	Returns the arc tangent of two variables x and y
atanh()	Returns the inverse hyperbolic tangent of a number
bindec()	Converts a binary number to a decimal number
ceil()	Rounds a number up to the nearest integer
cos()	Returns the cosine of a number
cosh()	Returns the hyperbolic cosine of a number
decbin()	Converts a decimal number to a binary number
dechex()	Converts a decimal number to a hexadecimal number
decoct()	Converts a decimal number to an octal number
deg2rad()	Converts a degree value to a radian value

exp()	Calculates the exponent of e
expm1()	Returns exp(x) – 1
floor()	Rounds a number down to the nearest integer
fmod()	Returns the remainder of x/y
getrandmax()	Returns the largest possible value returned by rand()
hexdec()	Converts a hexadecimal number to a decimal number
hypot()	Calculates the hypotenuse of a right-angle triangle
max()	Returns the highest value in an array, or the highest value of several specified values
min()	Returns the lowest value in an array, or the lowest value of several specified values
octdec()	Converts an octal number to a decimal number
pi()	Returns the value of PI
pow()	Returns x raised to the power of y
rad2deg()	Converts a radian value to a degree value
rand()	Generates a random integer
round()	Rounds a floating-point number
sin()	Returns the sine of a number
sinh()	Returns the hyperbolic sine of a number
sqrt()	Returns the square root of a number
srand()	Seeds the random number generator
tan()	Returns the tangent of a number
tanh()	Returns the hyperbolic tangent of a number

➢ **String functions**

Function	Description
bin2hex()	Converts a string of ASCII characters to hexadecimal values
chop()	Removes whitespace or other characters from the right end of a string
chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts

convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string encryption (hashing)
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
hex2bin()	Converts a string of hexadecimal values to ASCII characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the left side of a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string

sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings
sprintf()	Writes a formatted string to a variable
sscanf()	Parses input from a string according to a format
str_ireplace()	Replaces some characters in a string (case-insensitive)
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times
str_replace()	Replaces some characters in a string (case-sensitive)
str_shuffle()	Randomly shuffles all characters in a string
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
strcoll()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)
strlen()	Returns the length of a string
strncmp()	String comparison of the first n characters (case-sensitive)
strpbrk()	Searches a string for any of a set of characters
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string

strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-sensitive)
substr_count()	Counts the number of times a substring occurs in a string
substr_replace()	Replaces a part of a string with another string
trim()	Removes whitespace or other characters from both sides of a string
vfprintf()	Writes a formatted string to a specified output stream
vprintf()	Outputs a formatted string
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

➢ **Miscellaneous Functions**

Function	Description
connection_aborted()	Checks whether the client has disconnected
connection_status()	Returns the current connection status
connection_timeout()	Deprecated in PHP 4.0.5. Checks whether the script has timed out
constant()	Returns the value of a constant

define()	Defines a constant
defined()	Checks whether a constant exists
die()	Prints a message and exits the current script
eval()	Evaluates a string as PHP code
exit()	Prints a message and exits the current script
get_browser()	Returns the capabilities of the user's browser
halt_compiler()	Halts the compiler execution
pack()	Packs data into a binary string
php_check_syntax()	Deprecated in PHP 5.0.5
php_strip_whitespace()	Returns the source code of a file with PHP comments and whitespace removed
sleep()	Delays code execution for a number of seconds
sys_getloadavg()	Gets system load average
time_nanosleep()	Delays code execution for a number of seconds and nanoseconds
time_sleep_until()	Delays code execution until a specified time
uniqid()	Generates a unique ID
unpack()	Unpacks data from a binary string
usleep()	Delays code execution for a number of microseconds

PHP COOKIES

***Cookies are small data stored on the client computer and they are kept of use tracking purpose*.**

PHP transparently supports HTTP cookies. There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

- **Setting Cookies with PHP:**

  PHP provided **setcookie()** function to set a cookie. This function requires six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

  setcookie(name, value, expire, path, domain, security);

- **Name:** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value:** This sets the value of the named variable and is the content that the user wants to store.
- **Expiry:** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the browser is closed.
- **Path**: This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain**: This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security:** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Setting a cookie

```
<?php
  setcookie("name", "John Watkin", time()+3600, "/","", 0);
  setcookie("age", "36", time()+3600, "/", "",  0); ?>
<html><head> <title>Setting Cookies with PHP</title> </head>
<body>
<?php echo "Set Cookies"?> </body></html>
```

- **Accessing Cookies with PHP**

  PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables.isset() function is used to check if a cookie is set or not.

Accessing cookies

```
<html><head> <title>Accessing Cookies with PHP</title> </head>
<body> <?php
echo $_COOKIE["name"]. "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"]. "<br />";
echo $_COOKIE["age"] . "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"] . "<br />";
?> </body></html>
```

➢ **Deleting Cookie**

To delete a cookie users should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on. It is safest to set the cookie with a date that has already expired

```
<?php
 setcookie( "name", "", time()- 60, "/","", 0);
 setcookie( "age", "", time()- 60, "/","", 0);
?>
<html><head> <title>Deleting Cookies with PHP</title> </head>
<body>
<?php echo "Deleted Cookies" ?> </body></html>
```

REGULAR EXPRESSIONS IN PHP

Regular expressions are nothing more than a sequence or pattern of characters. They provide the foundation for pattern-matching functionality.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression: POSIX Regular Expressions and PERL Style Regular Expressions.

POSIX Regular Expressions

- The structure of a POSIX regular expression is similar to a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

- The simplest regular expression is one that matches a single character.
    - **Brackets:** Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.
    - **Quantifiers:** The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.
    - **Predefined Character Ranges**: For programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.
[[:alpha:]]	It matches any string containing alphabetic characters aA through zZ.
[[:digit:]]	It matches any string containing numerical digits 0 through 9.
[[:alnum:]]	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
[[:space:]]	It matches any string containing a space.

PHP currently offers seven functions for searching strings using POSIX-style regular expressions:

Function	**Description**
ereg()	The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
sql_regcase()	The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PERL Style Regular Expressions:

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions

- **Metacharacters:** A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

Metacharacters	**Description**
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit

\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified

Modifiers: Several modifiers are available that can make the user work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifiers	Description
I	Makes the match case insensitive
M	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of string boundary
O	Evaluates the expression only once
S	Allows use of . to match a newline character
X	Allows the user to use white space in the expression for clarity
G	Globally finds all matches
Cg	Allows a search to continue even after a global match fails

PHP's Regexp PERL Compatible Functions

Functions	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_ quote()	Quote regular expression characters

DATABASE CONNECTIVITY

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

Opening Database Connection:

PHP provides mysql_connect function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

```
connection mysql_connect(server,user,password,new_link,client_flag);
```

- **Server:** The host name running database server. If not specified then default value is localhost:3306. This is optional.
- **User:** The username accessing the database. If not specified then default is the name of the user that owns the server process. This is optional.
- **Password:** The password of the user accessing the database. If not specified then default is an empty password.
- **new_link:** If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned. This is optional.
- **client_flags:** This is optional. A combination of the following constants:
  1. MYSQL_CLIENT_SSL - Use SSL encryption
  2. MYSQL_CLIENT_COMPRESS - Use compression protocol
  3. MYSQL_CLIENT_IGNORE_SPACE - Allow space after function names
  4. MYSQL_CLIENT_INTERACTIVE - Allow interactive timeout seconds of inactivity before closing the connection

➢ **Closing Database Connection:**

PHP uses mysql_close to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure. If a resource is not specified then last opened database is closed.

```
bool mysql_close ( resource $link_identifier );
```

➢ **Creating a Database:**

To create and delete a database the users should have admin privilege. PHP uses mysql_query function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

```
bool mysql_query( sql, connection );
```

- **Sql:** SQL query to create a database
- **Connection:** if not specified then last opened connection by mysql_connect will be used.

➢ **Selecting a Database:**

Once the user establishes a connection with a database server then it is required to select a particular database where all the tables are associated. This is required because there may be multiple databases residing on a single server. PHP provides function mysql_select_db to select a database. It returns TRUE on success or FALSE on failure.

bool mysql_select_db(db_name, connection);

- **db_name:** Database name to be selected
- **connection**: if not specified then last opened connection by mysql_connect will be used.

➢ **Creating Database Tables:**

To create tables in the new database the user need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using mysql_query() function.

Creating and selecting a database table

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);  //Creating a connection
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
echo 'Connected successfully';
$sql = 'CREATE TABLE employee( '. 'emp_id INT NOT NULL AUTO_INCREMENT, '.
       'emp_name VARCHAR(20) NOT NULL, '. 'emp_address  VARCHAR(20) NOT NULL, 
'.'emp_salary   INT NOT NULL, '. 'join_date    timestamp(14) NOT NULL, '.
       'primary key ( emp_id ))'; //Creating a table
mysql_select_db('test_db'); //Seeting a table
$retval = mysql_query( $sql, $conn );
if(! $retval )
```

```
{   die('Could not create table: ' . mysql_error()); }
echo "Table employee created successfully\n";
mysql_close($conn); //Closing a connection ?>
```

In case the user need to create many tables then it is better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and execute those commands.

- **Deleting a Database:**

  If a database is no longer required then it can be deleted forever. The users can use pass an SQL command to mysql_query to delete a database.

- **Deleting a Table:**

  It is again a matter of issuing one SQL command through mysql_query function to delete any database table. But be very careful while using this command because by doing so the users can delete some important information the user has in the table.

Deleting a table

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
$sql = 'DROP TABLE employee'; //Deleting a table
$retval = mysql_query( $sql, $conn );
if(! $retval )
{   die('Could not delete table employee: ' . mysql_error()); }
echo "Table deleted successfully\n";
mysql_close($conn); ?>
```

- **Insert Data into MySQL Database**

  Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function mysql_query. In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables.

Inserting values

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
$sql = 'INSERT INTO employee '.  '(emp_name,emp_address, emp_salary, join_date) '.
     'VALUES ( "guest", "XYZ", 2000, NOW() )';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{   die('Could not enter data: ' . mysql_error()); }
echo "Entered data successfully\n";
mysql_close($conn); ?>
```

- **Getting Data From MySQL Database**

  Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. The user have several options to fetch data from MySQL. The most frequently used option is to use function mysql_fetch_array(). This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Fetching data from tables

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

```
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{   die('Could not get data: ' . mysql_error()); }
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{     echo "EMP ID :{$row['emp_id']}  <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> "; }
echo "Fetched data successfully\n";
mysql_close($conn); ?>
```

- **Deleting Data from MySQL Database**

  Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function mysql_query. To delete a record in any table it is required to locate that record by using a conditional clause.

Deleting data from tables

```
<html> <head> <title>Delete a Record from MySQL Database</title> </head>
<body> <?php
if(isset($_POST['delete']))
{ $dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
$emp_id = $_POST['emp_id'];
$sql = "DELETE employee WHERE emp_id = $emp_id" ; //Query to delete a record
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{   die('Could not delete data: ' . mysql_error()); }
echo "Deleted data successfully\n";
mysql_close($conn); }
else
{ ?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="400" border="0" cellspacing="1" cellpadding="2"> <tr>
<td width="100">Employee ID</td>
<td><input name="emp_id" type="text" id="emp_id"></td> </tr>
<tr> <td width="100"></td>
<td></td> </tr>
<tr> <td width="100"></td>
<td> <input name="delete" type="submit" id="delete" value="Delete"> </td> </tr>
</table> </form>
<?php } ?> </body></html>
```

- **Updating Data into MySQL Database**

  Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function mysql_query. To update a record in any table it is required to locate that record by using a conditional clause.

Updating data

```
<html><head> <title>Update a Record in MySQL Database</title> </head><body>
<?php
if(isset($_POST['update']))
{ $dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{   die('Could not connect: ' . mysql_error()); }
$emp_id = $_POST['emp_id'];
$emp_salary = $_POST['emp_salary'];
$sql = "UPDATE employee SET emp_salary = $emp_salary WHERE emp_id = $emp_id"
;
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{   die('Could not update data: ' . mysql_error()); }
echo "Updated data successfully\n";
mysql_close($conn); }
else {?>
<form method="post" action="<?php $_PHP_SELF ?>">
<table width="400" border="0" cellspacing="1" cellpadding="2">
<tr> <td width="100">Employee ID</td>
<td><input name="emp_id" type="text" id="emp_id"></td> </tr>
<tr> <td width="100">Employee Salary</td>
<td><input name="emp_salary" type="text" id="emp_salary"></td> </tr>
<tr> <td width="100"></td> <td></td> </tr>
<tr> <td width="100"></td> <td>
<input name="update" type="submit" id="update" value="Update"> </td> </tr>
</table> </form> <?php }
?> </body></html>
```

4.2 XML (Extensible Markup Language)

XML is a text-based markup language derived from Standard Generalized Markup Language (SGML).It is a new markup language, developed by the W3C (World Wide Web Consortium), mainly to overcome the limitations in HTML. **Markup** is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML is not a programming language. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML. XML don't have pre - defined tags and the tags are stricter than HTML.

XML is extensible: XML allows the user to create user defined self-descriptive tags, or language that suits the application.

XML carries the data, does not present it: XML allows storing the data irrespective of how it will be presented.

XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

Features of XML

- ❖ XML simplifies the creation of HTML documents for large web sites.
- ❖ XML can be used to exchange the information between organizations and systems.
- ❖ XML can be used for offloading and reloading of databases.
- ❖ XML can be used to store and arrange the data, which can customize the user data handling needs.
- ❖ XML can easily be merged with style sheets to create almost any desired output.
- ❖ Any type of data can be expressed as an XML document
- ❖ It has syndicated content, where content is being made available to different web sites.
- ❖ It suits well for electronic commerce applications where different organizations collaborate to serve a customer.
- ❖ It supports scientific applications with new markup languages for mathematical and chemical formulas.
- ❖ It also supports electronic books with new markup languages to express rights and ownership.
- ❖ XML could also be used in handheld devices and smart phones with new markup languages optimized for these devices.

Syntax Rules

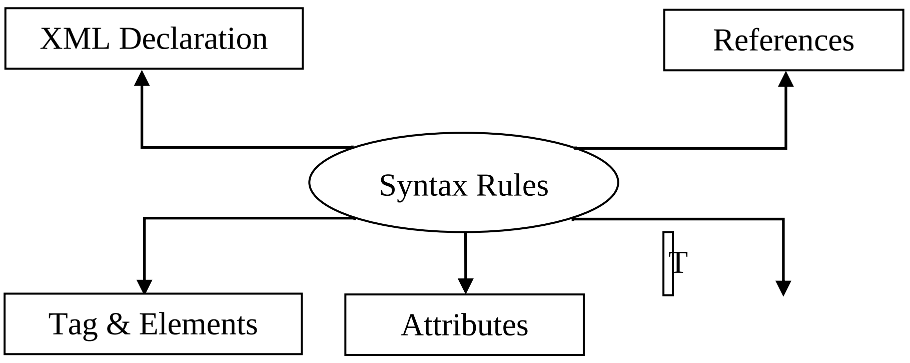


Fig 4.1 Syntax Rules of XML

➢ **XML Declaration**

The XML document can optionally have an XML declaration.

<?xml version="1.0" encoding="UTF-8"?>.

Version is the XML version and encoding specifies the character encoding used in the document.

Syntax Rules for XML declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of encoding that the user put in the XML declaration.

➢ **Tags and Elements**

An XML file is structured by several XML-elements also called **XML-nodes** or XML-tags. XML-elements' names are enclosed by angular brackets <>.

<element>....</element> or

<element/>

Nesting of elements

An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

```
<?xml version="1.0"?>
<contact-info>
<company>abc</company>
<contact-info>
```

- **Root element:** An XML document can have only one root element.
- **Case sensitivity:** The names of XML-elements are case-sensitive.

➢ **Attributes**

An attribute specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. It is possible to attach additional information to elements in the form of attributes. The names follow the same rules as element names.

<tel preferred=”true”>513-555-8889</tel>

- **XML References**

  References allow the user to add or include additional text or markup in an XML document. References always begin with the symbol "&", which is a reserved character and end with the symbol ";". XML has two types of references:

  - **Entity References:** An entity reference contains a name between the start and the end delimiters. The entity reference is replaced by the content of the entity.
  - **Character References:** A letter is replaced by its Unicode character code. Character references that start with &#x provides a hexadecimal representation of the character code.

XML Text

The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files. Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored. Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly.

XML Documents

An XML document is a basic unit of XML information composed of elements and other markup in an orderly package. An XML document can contains wide variety of data.

```
<?xml version="1.0"?> //document prolog
<contact-info> //all the following lines are document element
<name>Sharanya</name>
<company>abc</company>
<phone>(044)257887296</phone>
</contact-info>
```

The XML documents are divided into: Document prolog section and document element sections.

Document prolog: The document prolog comes at the top of the document, before the root element. This section contains: XML declaration and Document type declaration. The first line in the above example belongs to prolog section.

Document element: Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. The other lines except the first line come under document element section.

XML Declaration

The XML declaration is the first line of the document. The declaration identifies the document as an XML document. The declaration also lists the version of XML used in the document. The declaration can contain other attributes to support other features such as character set encoding.

Syntax:

```
<?xml
  version="version_number"
  encoding="encoding_declaration"
  standalone="standalone_status"
?>
```

- ❖ **Version-** Specifies the version of the XML standard used.
- ❖ **Encoding declaration-** It defines the character encoding used in the document. UTF-8 is the default encoding used.
- ❖ **Standalone-** It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to no. Setting it to yes tells the processor there are no external declarations required for parsing the document.

Rules for XML declaration

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: version, encoding and standalone.
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

Example	**Description**
<?xml >	XML declaration with no parameters.
<?xml version="1.0">	XML declaration with version definition.

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>	XML declaration with all parameters defined.
<?xml version='1.0' encoding='iso-8859-1' standalone='no' ?>	XML declaration with all parameters defined in single quotes.

XML tags

XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

- **Start Tag:** The beginning of every non-empty XML element is marked by a start-tag.

  Example: <address>.

- **End Tag:**Every element that has a start tag should end with an end-tag.

  Example: </address>

- **Empty Tag:** The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways:

  (1) A start-tag immediately followed by an end-tag :<hr></hr>

  (2) A complete empty-element tag : <hr />

XML Tags Rules

- XML tags are case-sensitive.
- XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed.

XML Elements

XML elements can be defined as building blocks of an XML. Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

```
<element-name attribute1 attribute2>

....content

</element-name>
```

- **element-name** is the name of the element.

- attribute1, attribute2 are **attributes** of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as: name = "value".

Empty Element: An empty element is an element with no content.

Example: <name attribute1 attribute2.../>

XML Elements Rules

- An element name can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements.

XML Attributes

Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

```
<element-name attribute1 attribute2 >

        ....content….

< /element-name>
```

where attribute1 and attribute2 has the following form: name = "value"

The following are the types of attributes:

➢ **String:** It takes any literal string as a value. CDATA is a String type. CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute.

➢ **Tokenized:** This is more constrained type. The validity constraints noted in the grammar are applied after the attribute value is normalized.

➢ **Enumerated**: This has a list of predefined values in its declaration, out of which, it must assign one value. There are two types of enumerated attribute:

  ❖ **NotationType:** It declares that an element will be referenced to a NOTATION declared somewhere else in the XML document.

  ❖ **Enumeration:** Enumeration allows the user to define a specific list of values that the attribute value must match.

Element Attribute Rules

An attribute name must not appear more than once in the same start-tag or empty-element tag. An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration. Attribute values must not contain direct or indirect entity references to external entities. The replacement text of any entity referred to directly or indirectly in an attribute value must not contain either less than sign <.

XML other features

- **Comments:**

  A comment starts with <!-- and ends with -->. Comments cannot appear before XML declaration. Comments can appear anywhere in a document. Comments must not appear within attribute values. Nested comments are not allowed.

- **Whitespaces:**

  Whitespace is a collection of spaces, tabs, and newlines. XML document contain two types of white spaces Significant Whitespace and Insignificant Whitespace. A **significant Whitespace** occurs within the element which contain text and markup present together.

```
<name>Adhithya Ramanan</name>
```

Insignificant whitespace means the space where only element content is allowed.

```
<address.category="residence">
```

Differences between XML and HTML

XML	HTML
XML was designed to be a software and hardware independent tool used to transport and store data, with focus on what data is.	HTML was designed to display data with focus on how data looks.
XML provides a framework for defining markup languages.	HTML is a markup language itself.
XML is neither a programming language nor a presentation language.	HTML is a presentation language.
XML is case sensitive.	HTML is case insensitive.
XML is used basically to transport data between the application and the database.	HTML is used for designing a web-page to be rendered on the client side.
In XML custom tags can be defined and the tags are invented by the author of the XML document.	HTML has its own predefined tags

XML makes it mandatory for the user the close each tag that has been used.	HTML is not strict if the user does not use the closing tags.
XML preserve white space.	HTML does not preserve white space.
XML is about carrying information, hence it is dynamic.	HTML is about displaying data, hence it is static.

DOCUMENT TYPE DECLARATION (DTD)

The document type declaration attaches a DTD to a document. It is a way to describe XML language precisely.

```
<!DOCTYPE element DTD identifier
[   declaration1
  declaration2
  ........          ]>
```

The DTD starts with <!DOCTYPE delimiter. An element tells the parser to parse the document from the specified root element. DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.** The square brackets [ ] enclose an optional list of entity declarations called **Internal Subset.**

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes.

```
<!DOCTYPE root-element [element-declarations]>
```

root-element is the name of root element and element-declarations is where the user declare the elements.

Internal DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name, company, phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)> ]>
<address> <name>Sharanya</name>
<company>abc</company>
<phone>12347890</phone> </address>
```

Start Declaration- Begin the XML declaration with following statement

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where elements, attributes, entities, and notations are declared.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>).

Rules

- The document type declaration must appear at the start of the document.
- The element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

```
<!DOCTYPE root-element SYSTEM "file-name">
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE address SYSTEM "address.dtd">

<address> <name>Sharanya</name>

<company>abc</company>

<phone>1234689</phone> </address>
```

address.dtd

```
<!ELEMENT address (name,company,phone)> <!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)> <!ELEMENT phone (#PCDATA)>
```

Types of external DTD

- **System Identifiers:** A system identifier enables the user to specify the location of an external file containing DTD declarations.

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

- **Public Identifiers:** Public identifiers provide a mechanism to locate DTD resources.

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers, or FPIs**.

Advantages of DTD

- The XML processor enforces the structure, as defined in the DTD.
- The application easily accesses the document structure.
- The DTD gives hints to the XML processor—that is, it helps separate indenting from content.
- The DTD can declare default or fixed values for attributes. This might result in a smaller document.

XML SCHEMAS

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
<xs:complexType><xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" /> </xs:sequence>
</xs:complexType></xs:element> </xs:schema>
```

- **Elements:** An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

- **Simple Type** - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date.

  Example: <xs:element name="phone_number" type="xs:int" />

- **Complex Type** - A complex type is a container for other element definitions. This allows the user to specify which child elements an element can contain and to provide some structure within the user's XML documents.

```
        <xs:element name="Address">
<xs:complexType><xs:sequence>
        <xs:element name="name" type="xs:string" />
                <xs:element name="company" type="xs:string" />
        <xs:element name="phone" type="xs:int" />
</xs:sequence><xs:complexType>
</xs:element>
```

- **Global Types** - With global type, the user can define a single type in the user's document, which can be used by all other references.

```
<xs:element name="AddressType">
<xs:complexType><xs:sequence>
        <xs:element name="name" type="xs:string" />
                <xs:element name="company" type="xs:string" />
</xs:sequence></xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
<xs:complexType><xs:sequence>
<xs:element name="address" type="AddressType" />
   <xs:element name="phone1" type="xs:int" />
</xs:sequence></xs:complexType>
```

```
</xs:element>
<xs:element name="Address2">
<xs:complexType><xs:sequence>
<xs:element name="address" type="AddressType" />
    <xs:element name="phone2" type="xs:int" />
</xs:sequence></xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition.

- **Attributes**

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below:

```
<xs:attribute name="x" type="y"/>
```

XML DOM

The Document Object Model (DOM) is the foundation of XML. XML documents have a hierarchy of informational units called nodes; DOM is a way of describing those nodes and the relationships between them.

DOM

A DOM Document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information.

```
<!DOCTYPE html> <html><body>
<h1> DOM example </h1>
<div> <b>Name:</b><span id="name"></span><br>
<b>Company:</b><span id="company"></span><br>
<b>Phone:</b><span id="phone"></span> </div>
<script>        if (window.XMLHttpRequest)
        {// code for IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp = new XMLHttpRequest();        }
        else        {// code for IE6, IE5
          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");        }
        xmlhttp.open("GET","/xml/address.xml",false);
        xmlhttp.send();
        xmlDoc=xmlhttp.responseXML;
        document.getElementById("name").innerHTML=
        xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
        document.getElementById("company").innerHTML=
        xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
        document.getElementById("phone").innerHTML=
        xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
</script></body></html>
```

address.xml

```
<?xml version="1.0"?>
<contact-info>
<name>Sharanya</name>
<company>abc</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

XML PARSERS

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents.

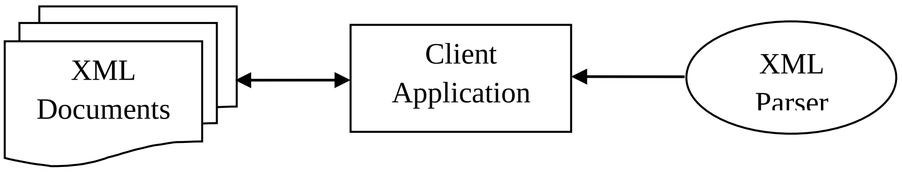


Fig 4.2 XML Parsers

The goal of a parser is to transform XML into a readable code. To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

VALIDATION

An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are: Well-formed XML document and Valid XML document

➢ Well-formed XML document

- Non DTD XML files must use the predefined character entities for amp(&), apos (single quote), gt >), lt (<), quot (double quote).
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self- ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- Amp (&), apos (single quote), gt (>), lt (<), quot (double quote) entities other than these must be declared.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name, company, phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)> ]>
<address> <name>Sharanya</name>
<company>abc</company>
<phone>(011) 123-4567</phone> </address>
```

➢ Valid XML document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

XSL (XML Style Sheet)

XML concentrates on the structure of the information and not its appearance. The W3C has published two recommendations for style sheets: CSS (Cascading Style Sheet) and XSL(XML Style sheet Language).XSL supports transforming the document before display. XSL would typically be used for advanced styling. XSL originally consisted of three parts:

- XSLT (XSL Transformation) - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO (XSL Formatting Objects) - a language for formatting XML documents

XSL

```
<P><B>Table of Contents</B></P> <UL>
<xsl:for-each select=”article/section/title”>
<LI><A><xsl:value-of select=”.”/></A></LI>
</xsl:for-each> </UL>
```

XSLT (XSL Transformation)

XSLT is a language to specify transformation of XML documents. It takes an XML document and transforms it into another XML document. XSLT is an XML-related technology that is used to manipulate and transform XML documents.

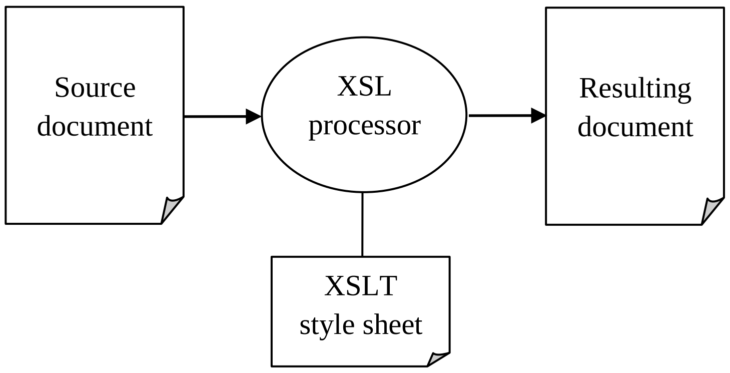


Fig 4.2 XSLT Transformation

With XSLT, the user can take an XML document and choose the elements and values, then generate a new file with new choices. Because of XSLT's ability to change the content of an XML document, XSLT is referred to as the **stylesheet for XML**. XSLT is not limited to styling activities. Many applications require transforming documents. XSLT can be used to:

- Add elements specifically for viewing, such as add the logo or the address of the sender to an XML invoice.

- Create new content from an existing one, such as create the table of contents
- Present information with the right level of details for the reader, such as using a style sheet to present high-level information to a managerial person while using another style sheet to present more detailed technical information to the rest of the staff.
- Convert between different DTDs or different versions of a DTD, such as convert a company specific DTD to an industry standard
- Transform XML documents into HTML for backward compatibility with existing browsers.

XSLT

XML code	XSLT code
<?xml version="1.0" encoding="UTF-8"?> <?xml-stylesheet type="text/xsl" href="class.xsl"?> <class> <student>Arthi</student> <student>Ambarish</student> <student>Anitha</student> <teacher>Sharanya</teacher> </class>	<?xml version="1.0" ?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" > <xsl:template match="teacher"> <p><u><xsl:value-of select="."/></u></p> </xsl:template> <xsl:template match="student"> <p><b><xsl:value-of select="."/></b></p> </xsl:template> <xsl:template match="/"> <html><body> <xsl:apply-templates/> </body></html> </xsl:template></xsl:stylesheet>

The XML file class.xml is linked to the XSLT code by adding the xml-stylesheet reference. The XSLT code then applies its rules to transform the XML document.

- Before XSLT: classoriginal.xml
- After XSLT rules are applied: class.xml

XSLT Syntax

➢ **XSLT - XML Declaration**

The user includes an XML declaration at the top of the XSLT documents. The attribute version defines what version of XML is used.

Example: <?xml version="1.0" ?>

➢ **XSLT - Stylesheet Root Element**

Every XSLT file must have the root element xsl:stylesheet. This root element has two attributes that must be included:

- version - the version of XSLT
- xmlns:xsl - the XSLT namespace, which is a URI to w3.org

➢ **XSLT - XSL: Namespace Prefix**

The root element specifies the XSL namespace. The standard form of an XSL element is: xsl:element

XSLT - Stylesheet Reference

Linking XML document to XSLT stylesheet is stylesheet reference. This is the magic step that connects XML to a XSLT file

XSLT - xml-stylesheet

xml-stylesheet is a special declaration in XML for linking XML with stylesheets. Place this after XML declaration to link the XML file to the XSLT code. xml-stylesheet has two attributes:

- type: the type of file being linked to. We will be using the value text/xsl to specify XSLT.
- href: the location of the file. If the user saved the user XSLT and XML file in the same directory, the user can simply use the XSLT filename.

Make sure that both XSLT and XML file are in the same directory.

Reference

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="class.xsl"?>
<class>        <student>Arun</student>
        <student>Divya</student>
        <teacher>Sharanya</teacher> </class>
```

XSLT: XSL Template

The purpose of XSLT is to help transform an XML document into something new. To transform an XML document, XSLT must be able to do two things well:

- Find information in the XML document.
- Add additional text and/or data.

Both of these items are taken care of with the very important XSL element xsl:template.

XSLT - xsl:template Match Attribute

To find information in an XML document use xsl:template's match attribute. It is in this attribute the knowledge of XPath is used to find information in the XML document. In previous example, to find student elements, we would set the match attribute to a simple XPath expression: student.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="student">
                Found a learner!
        </xsl:template>
<stylesheet>
```

XSLT - xsl:apply-templates

The xsl:apply-templates element to be more selective of the XML data.

- **XSLT - Remove Unwanted Text**

The following attributes are used to remove unwanted text:

- select attribute: lets the user choose specific child elements
- xsl:apply-templates: to decide when and where the xsl:template elements are used

XSLT - Remove Unwanted Children

We could use the select attribute to select specific child elements. To do this, we need a new xsl:template that matches our XML document's root element, class. We can then pick the child student using the select attribute. Here's the XSLT code to get the job done.

apply templates

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="class">
<xsl:apply-templates select="student"/>
</xsl:template>
<xsl:template match="student">
```

```
   Found a learner!
</xsl:template></xsl:stylesheet>
```

```
Found a learner! Found a learner! Found a learner!
```

The XSLT processor begins at the root element when looking for template matches. Because we have a match for the root element, class, the code we just added is used first.

xsl:apply-templates

In our template that matched class, we use xsl:apply-templates which will check for template matches on all the children of class. The children of class in our XML document are student and teacher.

xsl:apply-templates select="student"

To have the teacher element, "Sharanya," ignored, we use the select attribute of xsl:apply-templates to specify only student children.

The XSLT processor then goes searching templates that only match student elements.

xsl:template match="student"

The processor finds the only other template in our XSLT, which prints out, "Found a learner!" for each student element in the XML document. XSLT finds three students, so "Found a learner!" is displayed three times.

XSLT - Well-Formed Output

To obtain well formed output remove the root element in an XSLT template and inserting a new root element for the output. To do this, we are going to need to add an <html> (root element) tag, a <body> tag, and maybe some <p> tags.

XSLT - Replacing the Old Root Element

In the template that matches the original root element, we will insert the <html> tag to be the output's root element. We can also put the <body> tag there. In the template that matches the student elements, we can insert a <p> tag to make a separate paragraph for each student.

Replacing root

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="class"> <html><body>
<xsl:apply-templates select="student"/> </body></html> </xsl:template>
```

```
<xsl:template match="student">
<p>    Found a learner!</p> </xsl:template></xsl:stylesheet>
```

```
<html><body>
<p>Found a learner!</p>
<p>Found a learner!</p>
<p>Found a learner!</p>
</body></html>
```

4.3 NEWSFEEDS

News feeds are an example of automated syndication. News feed technologies allow information to be automatically provided and updated on Web sites, emailed to users, etc. As the name implies news feeds are normally used to provide news; however the technology can be used to syndicate a wide range of information**.**

The **BBC ticker** is an example of a news feed application. A major limitation with this approach is that the ticker can only be used with information provided by the BBC. The RSS (Really Simple Syndication)standard was developed as an open standard for news syndication, allowing applications to display news supplied by any RSS provider.

RSS (Really Simple Syndication)

RSS is an XML dialect used to publish frequently updated content, such as blog posts or news headlines.

It is a way to easily distribute a list of headlines, update notices, and sometimes content to a wide number of people. It is used by computer programs that organize those headlines and notices for easy reading.

The content feed is identified by a unique URI, and this URI is used by an RSS Reader application to retrieve and display the content feed from a web site. An RSS feed can contain one or more images or items. An item can be a synopsis of an article with a link to the full article or the entire article itself. An RSS has a well-defined structure. Some Web APIs use RSS as the data format returned when a request is made.

Working of RSS

RSS works by having the website author maintain a list of notifications on their website in a standard way. This list of notifications is called an **RSS Feed**. People who are interested in finding out the latest headlines or changes can check this list. Special computer programs called **RSS aggregators** have been developed that automatically access the RSS feeds of websites of user’s interest and organize the results.

Producing an RSS feed is very simple and hundreds of thousands of websites now provide this feature, including major news organizations like the New York Times, the BBC, and Reuters, as well as many weblogs.

RSS provides very basic information to do its notification. It is made up of a list of items presented in order from newest to oldest. Each item usually consists of a simple title describing the item along with a more complete description and a link to a web page with the actual information being described. Sometimes this description is the full information the user want to read (such as the content of a weblog post) and sometimes it is just a summary.

Creating RSS feed

The special XML-format file that makes up an RSS feed is usually created in one of a variety of ways. Most large news websites and most weblogs are maintained using special **content management** programs. Authors add their stories and postings to the website by interacting with those programs and then use the program's publish facility to create the HTML files that make up the website. Those programs often also can update the RSS feed XML file at the same time, adding an item referring to the new story or post, and removing less recent items.

Blog creation tools like Blogger, LiveJournal, Movable Type, and Radio automatically create feeds. Websites that are produced in a more custom manner, such as with Macromedia Dreamweaver or a simple text editor, usually do not automatically create RSS feeds. Authors of such websites either maintain the XML files by hand, just as they do the website itself, or use a tool such as Software Garden, Inc.'s ListGarden program to maintain it.

There are also services that periodically read requested websites themselves and try to automatically determine changes (this is most reliable for websites with a somewhat regular news-like format), or that let the users create RSS feed XML files that are hosted by that service provider.

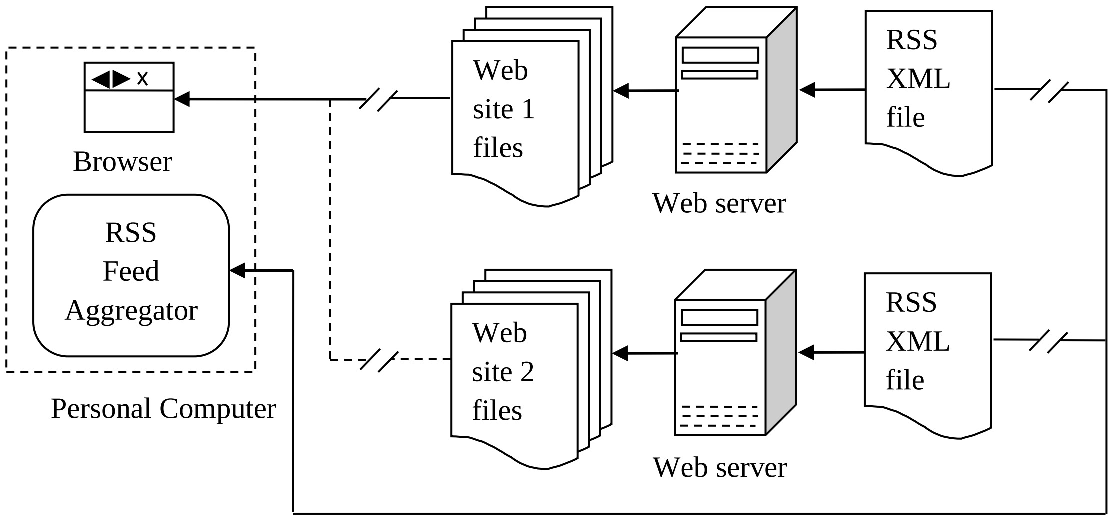


Fig 4.3: Communication between websites, RSS feed and PC

In the above diagram, a web browser being used to read first Web Site 1 over the Internet and then Web Site 2. It also shows the RSS feed XML files for both websites being monitored simultaneously by an RSS Feed Aggregator.

➢ **Sections of an RSS file**

Apart from the root element there are four main sections of the RSS file. These are the channel, image, item, and text input sections. In practical use, the channel and item elements are requirements for any useful RSS file, while the image and text input are optional.

The channel section

The channel element contains metadata that describe the channel itself, telling what the channel is and who created it. The channel is a required element that includes the name of the channel, its description, its language, and a URL. The URL is normally used to point to the channel's source of information.

Channel element

```
<channel><title>MozillaZine</title>

<link>http://www.mozillazine.org</link>

<description>The user source for Mozilla news, advocacy, interviews, builds, and more!
</description>

<language>en-us</language> </channel>
```

The title can be treated as a headline link with the description following. The **Channel Language definition** allow aggregators to filter news feeds and gives the rendering software the information necessary to display the language properly. The </channnel> tag is used after all the channel elements to close the channel. As RSS conforms to XML specs, the element must be well formed; it requires the closing tag.

The image section

The image element is an optional element that is usually used to include the logo of the channel provider. The default size for the image is 88 pixels wide by 31 pixels high, but it can be enlarged to 144 pixels wide by 400 pixels wide.

Image element

```
<image><title>MozillaZine</title>

<url>http://www.mozillazine.org/image/mynetscape88.gif</url>

<link>http://www.mozillazine.org</link>

<width>88</width>

<height>31</height> </image>
```

The image's title, URL, link, width, and height tags allow renderers to translate the file into HTML. The title tag is normally used for the image's ALT text.

The items

Items form the dynamic part of an RSS file. While channel, image, and text input elements create the channel's identity and typically stay the same over long periods of time, channel items are rendered as news headlines, and the channel's value depends on their changing fairly frequently.

Item element

```
<item><title>Java2 in Navigator 5?</title>
<link>http://www.mozillazine.org/talkback.html?article=607</link>
<description>Will Java2 be an integrated part of Navigator 5?
   Read more about it in this discussion...</description> </item>
```

Fifteen items are allowed in a channel. Titles should be less than 100 characters, while descriptions should be under 500 characters. The item title is normally rendered as a headline that links to the full article whose URL is provided by the item link. The item description is commonly used for either a summary of the article's content or for commentary on the article.

News feed channels use the description to highlight the content of news articles, usually on the channel owner's site, and Web log channels use the description to provide commentary on a variety of content, often on third-party sites.

The text input

The text input area is an optional element, with only one allowed per channel. This lets the user respond to the channel.

```
<textinput><title>Send</title>
<description>Comments about MozillaZine?</description>
<name>responseText</name>
<link>http://www.mozillazine.org/cgi-bin/sampleonly.cgi</link> </textinput>
```

The title is normally rendered as the label of the form's ssubmit button, and the description as the text displayed before or above the input field. The text input name is supplied along with the contents of the text field when the submit button is clicked.

4.4 ATOM

Atom is a syndication data format like RSS, as well as a publishing protocol. The Atom data format uses XML syntax with one or more entry elements containing the full and/or summary content.

The **Atom Publishing Protocol** (APP) defines a hierarchy for organizing published content (services, workspaces, collections, and resources) and uses the HTTP Get, Post, Put, and Delete methods for retrieving, creating, deleting, and editing published content. Atom’s use of HTTP’s built-in methods and XML as a data format is in the spirit of a RESTful web services implementation.

Atom was designed to be a universal publishing standard for blogs and other Web sites where content is updated frequently. Users visiting a Web site with an Atom feed can discover a file described as "atom.xml" in the URL that can be copied and pasted into an aggregator to subscribe to the feed.

Atom was originally developed as an alternative to RSS 2.0, the standard developed by Dave Winer and copyrighted by Harvard University, as a means of improving perceived shortcomings of the RSS format by the blogging community.

Features of ATOM

- Atom was developed to be vendor neutral and freely extensible by any user; it is an open standard.
- Atom lies within an XML-namespace.
- Atom includes auto discovery, allowing feed aggregators to automatically detect the presence of a feed.
- Atom differentiates between relative and non-relative URIs.
- Atom has separate summary and content elements.
- Atom explicitly labels a payload as plaintext or HTML.
- Each entry has a globally unique ID.

UNIT V

INTRODUCTION TO ANGULAR and WEB APPLICATIONS FRAMEWORKS

Introduction to AngularJS, MVC Architecture, Understanding ng attributes, Expressions and data binding, Conditional Directives, Style Directives, Controllers, Filters, Forms, Routers, Modules, Services; Web Applications Frameworks and Tools – Firebase- Docker- Node JS- React- Django- UI & UX.

1. OVERVIEW

AngularJS is an open source, JavaScript based web application development framework.

Definition of AngularJS as put by its official documentation is as follows:

AngularJS is a structural framework for dynamic web applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application components clearly and succinctly. Its data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google.

General Features

The most important general features of AngularJS are:

- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Overall, AngularJS is a framework to build large scale, high performance, and easy-to-maintain web applications.

Core Features

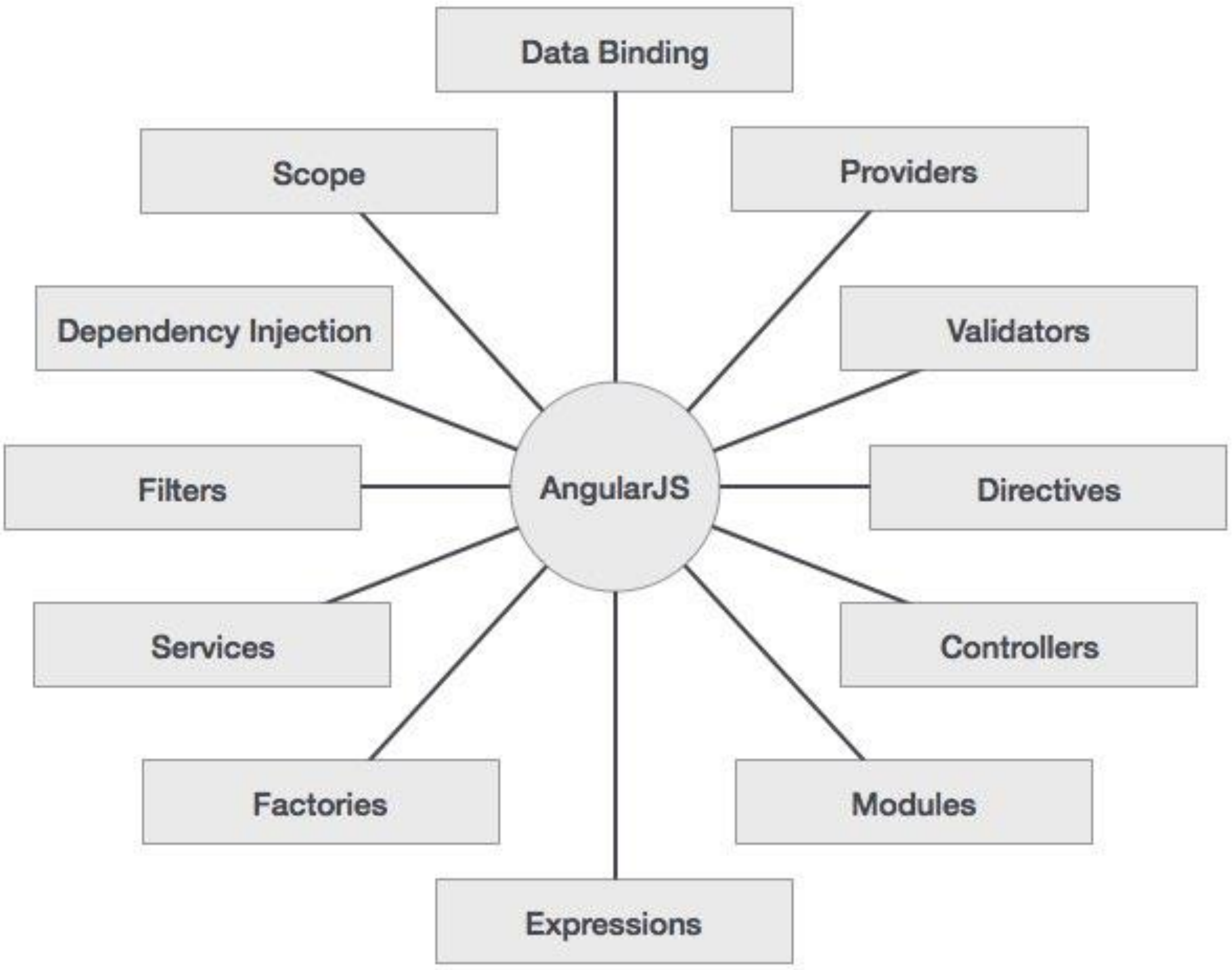
The most important core features of AngularJS are:

- **Data-binding:** It is the automatic synchronization of data between model and view components.
- **Scope:** These are objects that refer to the model. They act as a glue between controller and view.

- **Controller:** These are JavaScript functions bound to a particular scope.
- **Services:** AngularJS comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Filters:** These select a subset of items from an array and returns a new array.
- **Directives:** Directives are markers on DOM elements such as elements, attributes, css, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel etc.
- **Templates:**These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using *partials*.
- **Routing:** It is concept of switching views.
- **Model View Whatever:** MVW is a design pattern for dividing an application into different parts called Model, View, and Controller, each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.
- **Deep Linking:** Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
- **Dependency Injection:** AngularJS has a built-in dependency injection subsystem that helps the developer to create,understand, and test the applications easily.

Concepts

The following diagram depicts some important parts of AngularJS which we will discuss in detail in the subsequent chapters.



Advantages of AngularJS

The advantages of AngularJS are:

- AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
- AngularJS provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

On the top of everything, AngularJS applications can run on all major browsers and smart phones, including Android and iOS based phones/tablets.

Disadvantages of AngulaJS

Though AngularJS comes with a lot of merits, here are some points of concern:

- **Not Secure** : Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable**: If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

AngularJS Directives

The AngularJS framework can be divided into three major parts:

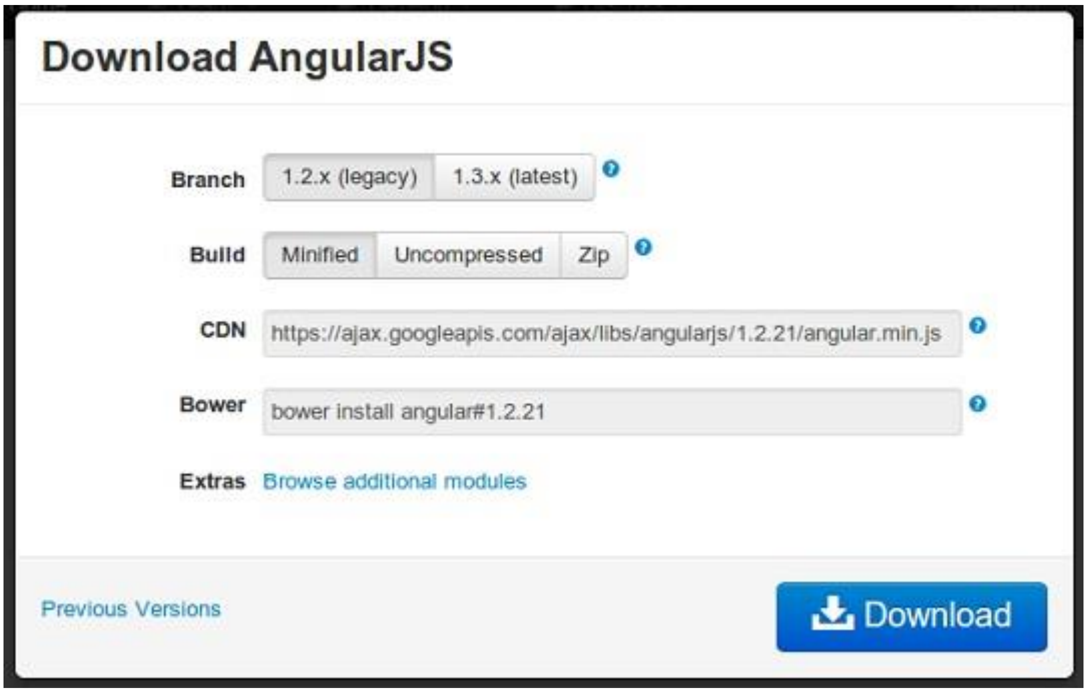
- **ng-app** : This directive defines and links an AngularJS application to HTML.
- **ng-model** : This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** : This directive binds the AngularJS application data to HTML tags.

2. ENVIRONMENT

This chapter describes how to set up AngularJS library to be used in web application development. It also briefly describes the directory structure and its contents.

When you open the link https://angularjs.org/, you will see there are two options to download AngularJS library:

- **View on GitHub**- By clicking on this button, you are diverted to GitHub and get all the latest scripts.
- **Download**- By clicking on this button, a screen you get to see a dialog box shown as:


Download AngularJS

Branch: 1.2.x (legacy) | 1.3.x (latest)

Build: Minified | Uncompressed | Zip

CDN: https://ajax.googleapis.com/ajax/libs/angularjs/1.2.21/angular.min.js

Bower: bower install angular#1.2.21

Extras: Browse additional modules

This screen offers various options for selecting Angular JS as follows:

- Downloading and hosting files locally
  - There are two different options : Legacy and Latest. The names themselves are self descriptive. The Legacy has version less than 1.2.x and the Latest come with version 1.3.x.
  - We can also go with the minimized, uncompressed, or zipped version.
- CDN access: You also have access to a CDN. The CDN gives you access around the world to regional data centres. In this case, the Google host. This means, using CDN transfers the responsibility of hosting files from your own servers to a series of external ones. This also offers an advantage that if the visitor of your web page has already downloaded a copy of AngularJS from the same CDN, there is no need to re-download it.

We are using the CDN versions of the library throughout this tutorial.

Example

Now let us write a simple example using AngularJS library. Let us create an HTML file*myfirstexample.html* shown as below:

```
<!doctype html>

<html>
```

```
   <head>

      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-
beta.17/angular.min.js"></script>

   </head>

   <body ng-app="myapp">

      <div ng-controller="HelloController" >

         <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>

      </div>

      <script>

         angular.module("myapp", [])

         .controller("HelloController", function($scope) {

            $scope.helloTo = {};

            $scope.helloTo.title = "AngularJS";

         });

      </script>

   </body>

</html>
```

Let us go through the above code in detail:

Include AngularJS

We include the AngularJS JavaScript file in the HTML page so that we can use it:

```
<head>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</head>
```

You can check the latest version of AngularJS on its official website.

Point to AngularJS app

Next, it is required to tell which part of HTML contains the AngularJS app. You can do this by adding the *ng-app* attribute to the root HTML element of the AngularJS app. You can either add it to *html* element or *body* element as shown below:

```
<body ng-app="myapp">

</body>
```

View

The view is this part:

```
<div ng-controller="HelloController" >

   <h2>Welcome {{helloTo.title}} to the world of Tutorialspoint!</h2>

</div>
```

ng-controller tells AngularJS which controller to use with this view. *helloTo.title* tells AngularJS to write the *model* value named helloTo.title in HTML at this location.

Controller

The controller part is:

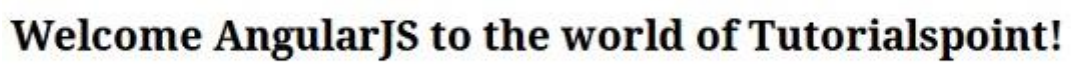
```
<script>

   angular.module("myapp", [])

   .controller("HelloController", function($scope) {

      $scope.helloTo = {};

      $scope.helloTo.title = "AngularJS";

    });

</script>
```

This code registers a controller function named *HelloController* in the angular module named *myapp*. We will study more about modules and controllers in their respective chapters. The controller function is registered in angular via the angular.module(...).controller(...) function call.

The $scope parameter *model* is passed to the controller function. The controller function adds a *helloTo* JavaScript object, and in that object it adds a *title* field.

Execution

Save the above code as *myfirstexample.html* and open it in any browser. You get to see the following output:



```
Welcome AngularJS to the world of Tutorialspoint!
```

What happens when the page is loaded in the browser ? Let us see:

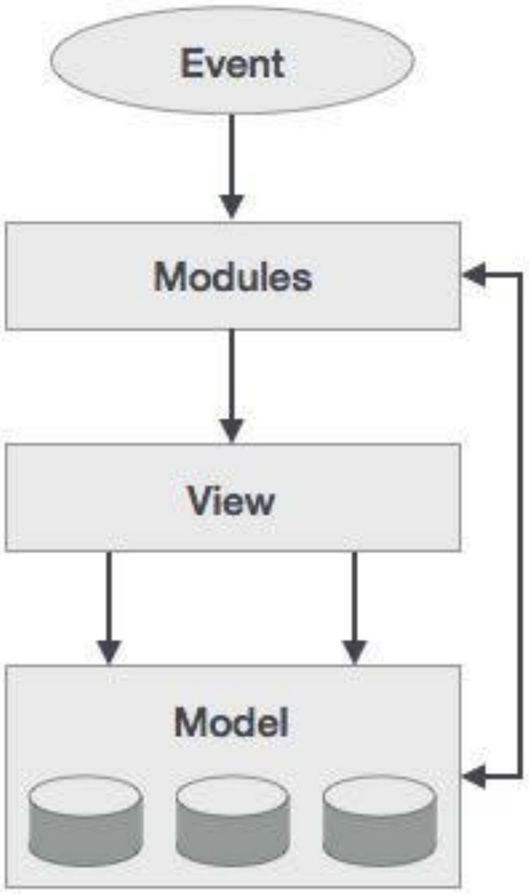
- HTML document is loaded into the browser, and evaluated by the browser.
- AngularJS JavaScript file is loaded, the angular *global* object is created.
- The JavaScript which registers controller functions is executed.
- Next, AngularJS scans through the HTML to search for AngularJS apps as well as views.
- Once the view is located, it connects that view to the corresponding controller function.
- Next, AngularJS executes the controller functions.
- It then renders the views with data from the model populated by the controller. The page is now ready.

3. MVC ARCHITECTURE

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- **Model** - It is the lowest level of the pattern responsible for maintaining data.
- **View** - It is responsible for displaying all or a portion of the data to the user.
- **Controller** - It is a software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. The controller receives all requests for the application and then works with the model to prepare any data needed by the view. The view then uses the data prepared by the controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



The model

The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself.

The view

A presentation of data in a particular format, triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

The controller

The controller responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.

AngularJS is a MVC based framework. In the coming chapters, let us see how AngularJS uses MVC methodology.

4. FIRST APPLICATION

Before creating actual *Hello World !* application using AngularJS, let us see the parts of a AngularJS application. An AngularJS application consists of following three important parts:

- **ng-app** : This directive defines and links an AngularJS application to HTML.
- **ng-model** : This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** : This directive binds the AngularJS Application data to HTML tags.

Creating AngularJS Application

Step 1: Load framework

Being a pure JavaScript framework, it can be added using <Script> tag.

```
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">

</script>
```

Step 2: Define AngularJS application using *ng-app* directive.

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

...

</div>
```

Step 3: Define a model name using *ng-model* directive.

```
<p>Enter your Name: <input type="text" ng-model="name"></p>
```

Step 4: Bind the value of above model defined using ng-bind directive.

```
<p>Hello <span ng-bind="name"></span>!</p>
```

Executing AngularJS Application

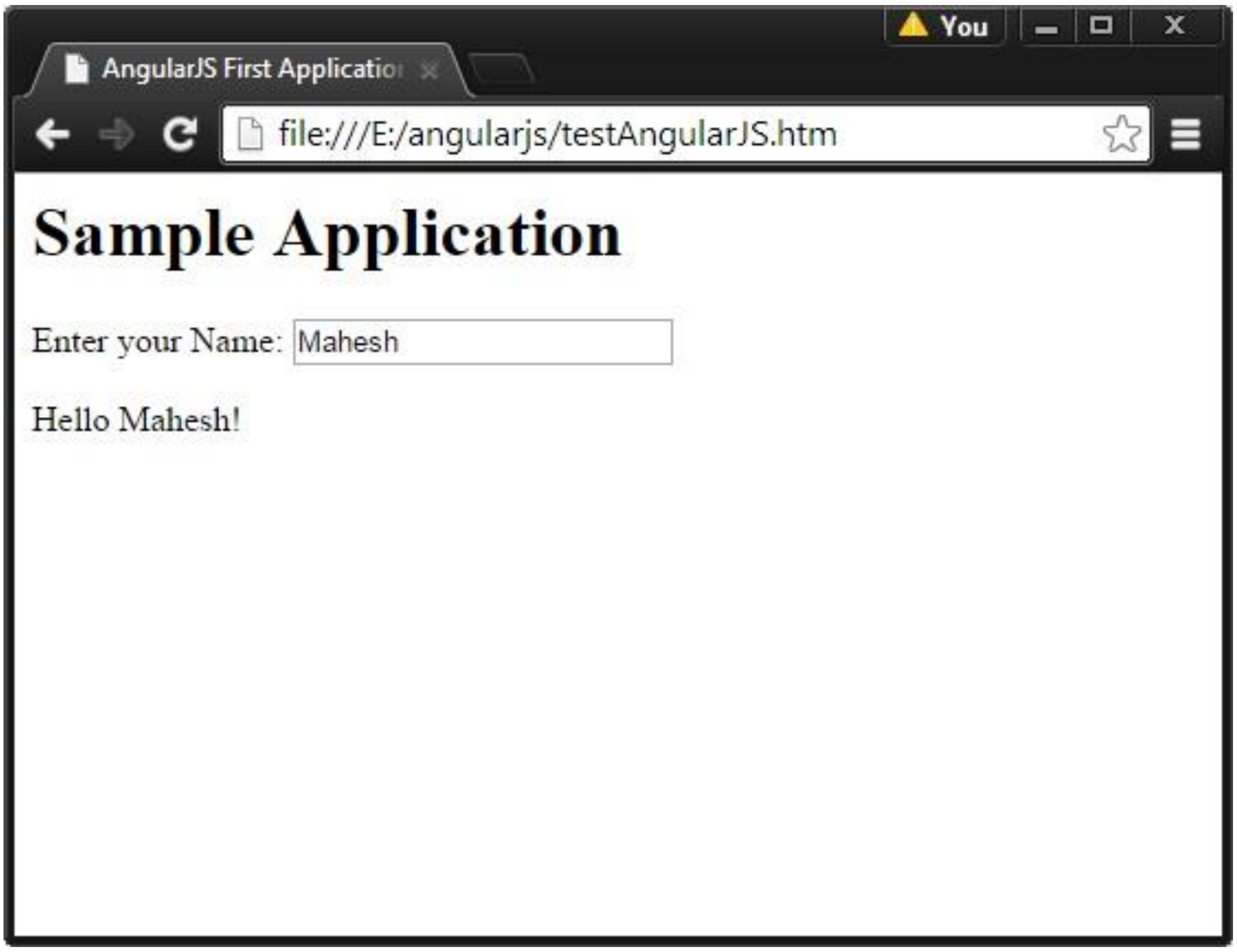
Use the above mentioned three steps in an HTML page.

testAngularJS.htm

```
<html>

<title>AngularJS First Application</title>

<body>

<h1>Sample Application</h1>

<div ng-app="">

   <p>Enter your Name: <input type="text" ng-model="name"></p>

   <p>Hello <span ng-bind="name"></span>!</p>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.


AngularJS First Application
You
file:///E:/angularjs/testAngularJS.htm

Sample Application

Enter your Name: Mahesh

Hello Mahesh!

How AngularJS integrates with HTML

- The ng-app directive indicates the start of AngularJS application.
- The ng-model directive creates a model variable named *name*, which can be used with the HTML page and within the div having ng-app directive.
- The ng-bind then uses the *name* model to be displayed in the HTML <span> tag whenever user enters input in the text box.
- Closing </div> tag indicates the end of AngularJS application.

5. DIRECTIVES

AngularJS directives are used to extend HTML. They are special attributes starting with **ng-**prefix. Let us discuss the following directives:

- **ng-app** - This directive starts an AngularJS Application.
- **ng-init** - This directive initializes application data.
- **ng-model** - This directive defines the model that is variable to be used in AngularJS.
- **ng-repeat** - This directive repeats HTML elements for each item in a collection.

ng-app directive

The ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when the web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application. In the following example, we define a default AngularJS application using ng-app attribute of a <div> element.

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

...

</div>
```

ng-init directive

The ng-init directive initializes an AngularJS Application data. It is used to assign values to the variables. In the following example, we initialize an array of countries. We use JSON syntax to define the array of countries.

```
<div ng-app="" ng-init="countries=[{locale:'en-US',name:'United States'},

                                    {locale:'en-GB',name:'United Kingdom'},

                                    {locale:'en-FR',name:'France'}]">



...
```

```
</div>
```

ng-model directive

The ng-model directive defines the model/variable to be used in AngularJS Application. In the following example, we define a model named *name*.

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

...

<p>Enter your Name: <input type="text" ng-model="name"></p>

</div>
```

ng-repeat directive

ng-repeat directive repeats HTML elements for each item in a collection. In the following example, we iterate over the array of countries.

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

...

   <p>List of Countries with locale:</p>

   <ol>

      <li ng-repeat="country in countries">

         {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

      </li>

   </ol>

</div>
```

Example

The following example shows the use of all the above mentioned directives.

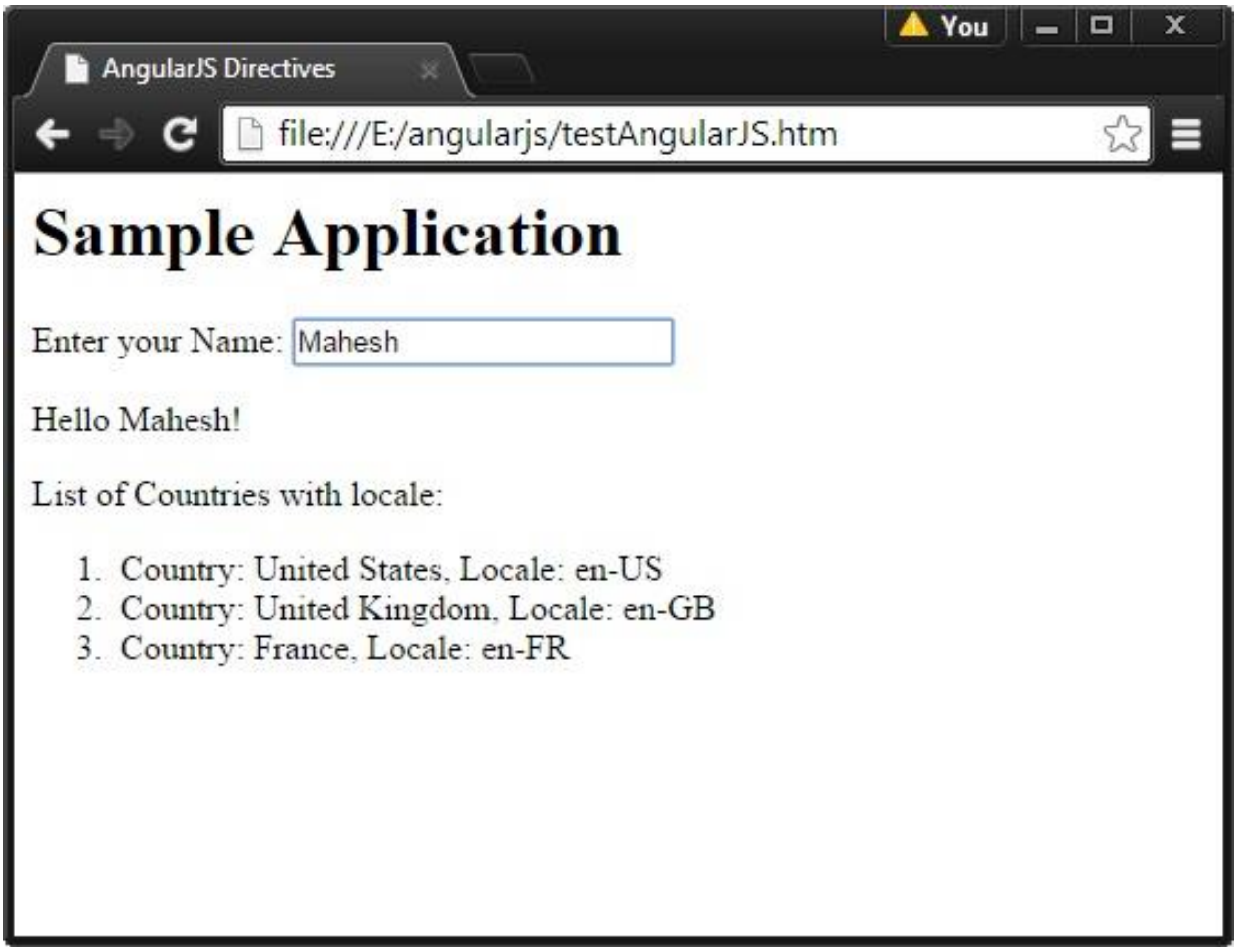
testAngularJS.htm

```
<html>

<title>AngularJS Directives</title>
```

```
<body>

<h1>Sample Application</h1>

<div ng-app="" ng-init="countries=[{locale:'en-US',name:'United States'},

                                    {locale:'en-GB',name:'United Kingdom'},

                                    {locale:'en-FR',name:'France'}]">

   <p>Enter your Name: <input type="text" ng-model="name"></p>

   <p>Hello <span ng-bind="name"></span>!</p>

   <p>List of Countries with locale:</p>

   <ol>

      <li ng-repeat="country in countries">

         {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}

      </li>

   </ol>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.



```
Sample Application

Enter your Name: Mahesh

Hello Mahesh!

List of Countries with locale:

  1. Country: United States, Locale: en-US
  2. Country: United Kingdom, Locale: en-GB
  3. Country: France, Locale: en-FR
```

6. EXPRESSIONS

Expressions are used to bind application data to HTML. Expressions are written inside double curly braces such as in {{ expression}}. Expressions behave similar to ng-bind directives. AngularJS expressions are pure JavaScript expressions and output the data where they are used.

Using numbers

```
<p>Expense on Books : {{cost * quantity}} Rs</p>
```

Using String

```
<p>Hello {{student.firstname + " " + student.lastname}}!</p>
```

Using Object

```
<p>Roll No: {{student.rollno}}</p>
```

Using Array

```
<p>Marks(Math): {{marks[3]}}</p>
```

Example

The following example shows use of all the above mentioned expressions:

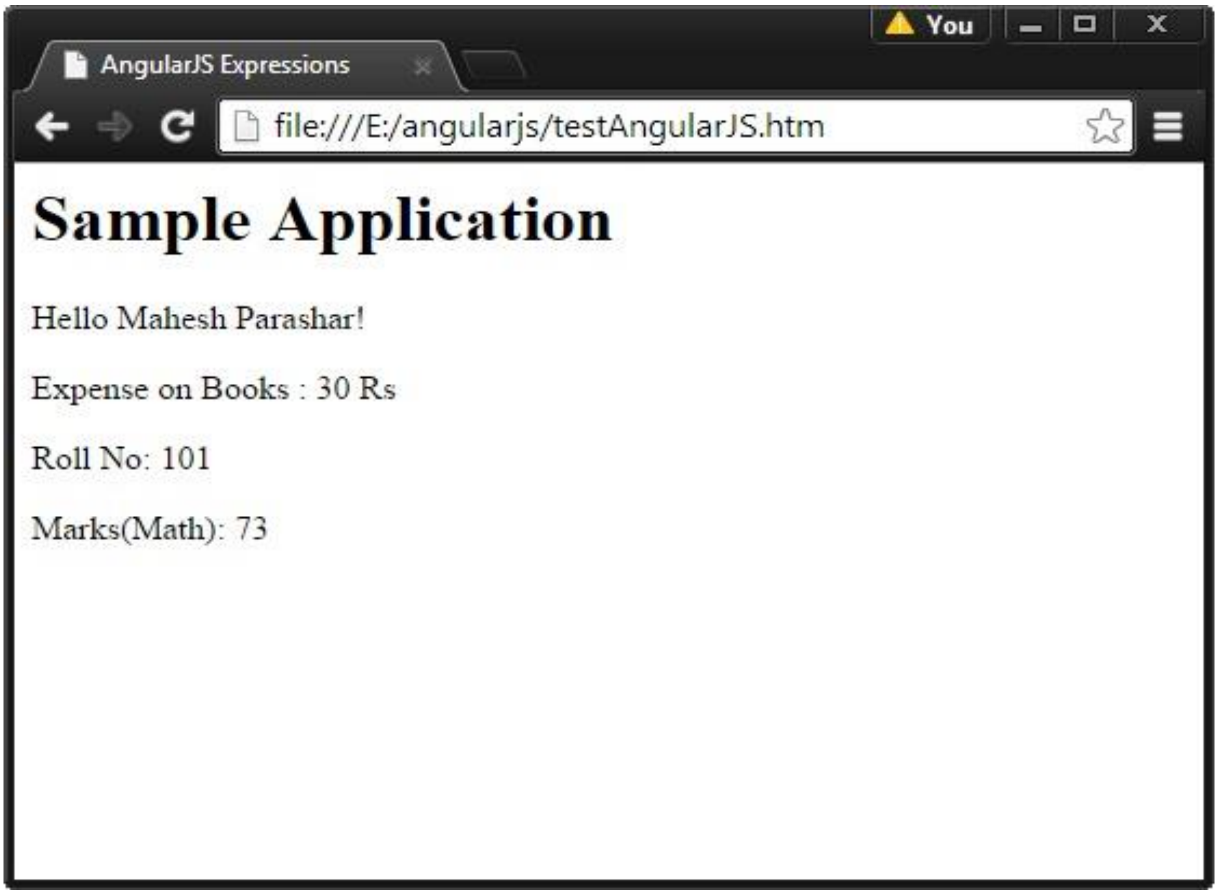
testAngularJS.htm

```
<html>

<title>AngularJS Expressions</title>

<body>

<h1>Sample Application</h1>
```

```
<div ng-app="" ng-init="quantity=1;cost=30;
student={firstname:'Mahesh',lastname:'Parashar',rollno:101};marks=[80,90,75
,73,60]">

   <p>Hello {{student.firstname + " " + student.lastname}}!</p>

   <p>Expense on Books : {{cost * quantity}} Rs</p>

   <p>Roll No: {{student.rollno}}</p>

   <p>Marks(Math): {{marks[3]}}</p>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



Sample Application

Hello Mahesh Parashar!

Expense on Books : 30 Rs

Roll No: 101

Marks(Math): 73

7. CONTROLLERS

AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using *ng-controller* directive. A controller is a JavaScript object that contains attributes/properties, and functions. Each controller accepts $scope as a parameter, which refers to the application/module that the controller needs to handle.

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

...

</div>
```

Here, we declare a controller named *studentController*, using ng-controller directive. As a next step, we define it as follows:

```
<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>
```

- The studentController is defined as a JavaScript object with $scope as an argument.
- The $scope refers to application which uses the studentController object.

- The $scope.student is a property of studentController object.
- The firstName and the lastName are two properties of $scope.student object. We pass the default values to them.
- The property fullName is the function of $scope.student object, which returns the combined name.
- In the fullName function, we get the student object and then return the combined name.
- As a note, we can also define the controller object in a separate JS file and refer that file in the HTML page.

Now we can use studentController's student property using ng-model or using expressions as follows:

```
Enter first name: <input type="text" ng-model="student.firstName"><br>

Enter last name: <input type="text" ng-model="student.lastName"><br>

<br>

You are entering: {{student.fullName()}}
```

- We bound student.firstName and student.lastname to two input boxes.
- We bound student.fullName() to HTML.
- Now whenever you type anything in first name and last name input boxes, you can see the full name getting updated automatically.

Example

The following example shows the use of controller:

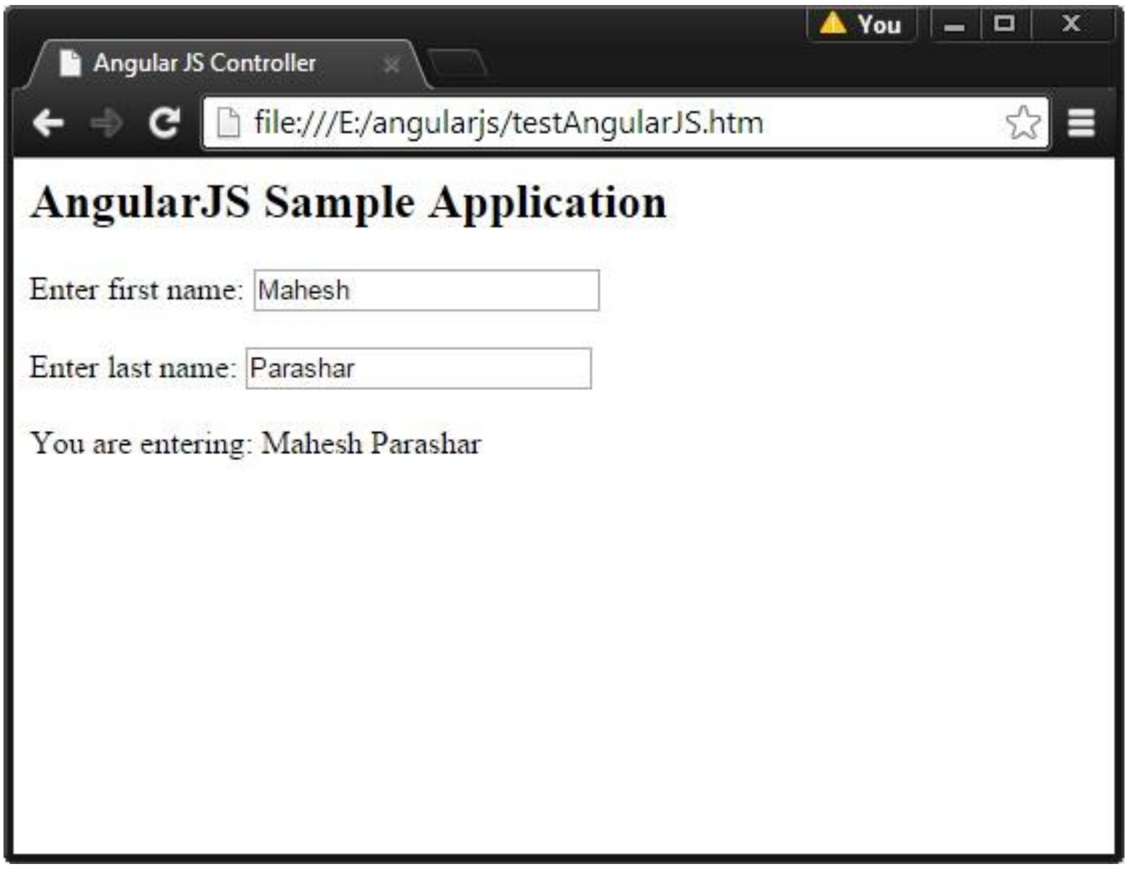
testAngularJS.htm

```
<html>

<head>

<title>Angular JS Controller</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>
```

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

Enter first name: <input type="text" ng-model="student.firstName"><br><br>

Enter last name: <input type="text" ng-model="student.lastName"><br>

<br>

You are entering: {{student.fullName()}}

</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



AngularJS Sample Application

Enter first name: Mahesh

Enter last name: Parashar

You are entering: Mahesh Parashar

8. FILTERS

Filters are used to modify the data. They can be clubbed in expression or directives using pipe (|) character. The following list shows commonly used filters.

S.No.	Name	Description
1	uppercase	converts a text to upper case text.
2	lowercase	converts a text to lower case text.
3	currency	formats text in a currency format.
4	filter	filter the array to a subset of it based on provided criteria.
5	orderby	orders the array based on provided criteria.

Uppercase Filter

This adds uppercase filter to an expression using pipe character. Here, we add uppercase filter to print student name in capital letters.

```
Enter first name:<input type="text" ng-model="student.firstName">

Enter last name: <input type="text" ng-model="student.lastName">

Name in Upper Case: {{student.fullName() | uppercase}}
```

Lowercase Filter

This adds lowercase filter to an expression using pipe character. Here, we add lowercase filter to print student name in small letters.

```
Enter first name:<input type="text" ng-model="student.firstName">

Enter last name: <input type="text" ng-model="student.lastName">

Name in Lower Case: {{student.fullName() | lowercase}}
```

Currency Filter

This adds currency filter to an expression that returns a number. Here, we add currency filter to print fees using currency format.

```
Enter fees: <input type="text" ng-model="student.fees">

fees: {{student.fees | currency}}
```

Filter Filter

To display only required subjects, we use subjectName as filter.

```
Enter subject: <input type="text" ng-model="subjectName">

Subject:

<ul>

  <li ng-repeat="subject in student.subjects | filter: subjectName">

    {{ subject.name + ', marks:' + subject.marks }}

  </li>

</ul>
```

Orderby Filter

To order subjects by marks, we use orderBy marks.

```
Subject:

<ul>

  <li ng-repeat="subject in student.subjects | orderBy:'marks'">

    {{ subject.name + ', marks:' + subject.marks }}

  </li>

</ul>
```

Example

The following example shows use of all the above mentioned filters.

testAngularJS.htm

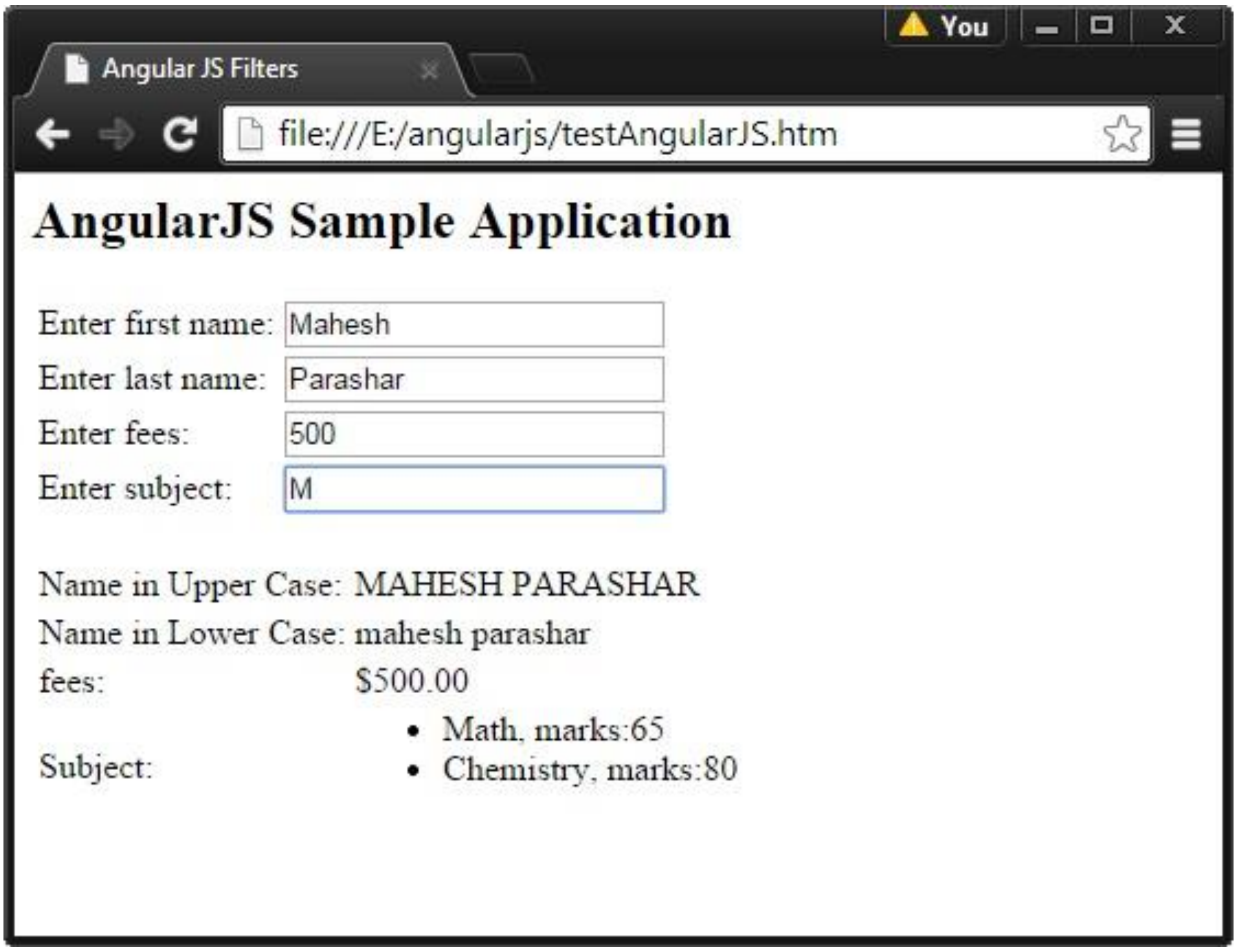
```
<html>

<head>
```

```
<title>Angular JS Filters</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="" ng-controller="studentController">

<table border="0">

<tr><td>Enter first name:</td><td><input type="text" ng-
model="student.firstName"></td></tr>

<tr><td>Enter last name: </td><td><input type="text" ng-
model="student.lastName"></td></tr>

<tr><td>Enter fees: </td><td><input type="text" ng-
model="student.fees"></td></tr>

<tr><td>Enter subject: </td><td><input type="text" ng-
model="subjectName"></td></tr>

</table>

<br/>

<table border="0">

<tr><td>Name in Upper Case: </td><td>{{student.fullName() |
uppercase}}</td></tr>

<tr><td>Name in Lower Case: </td><td>{{student.fullName() |
lowercase}}</td></tr>

<tr><td>fees: </td><td>{{student.fees | currency}}</td></tr>

<tr><td>Subject:</td><td>

<ul>

   <li ng-repeat="subject in student.subjects | filter: subjectName
|orderBy:'marks'">

      {{ subject.name + ', marks:' + subject.marks }}

   </li>

</ul>

</td></tr>

</table>
```

```
</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},

         {name:'Math',marks:65}

      ],

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser. See the result.



AngularJS Sample Application

Enter first name:	Mahesh
Enter last name:	Parashar
Enter fees:	500
Enter subject:	M

Name in Upper Case:	MAHESH PARASHAR
Name in Lower Case:	mahesh parashar
fees:	$500.00
Subject:	• Math, marks:65 • Chemistry, marks:80

9. TABLES

Table data is generally repeatable. The ng-repeat directive can be used to draw table easily. The following example shows the use of ng-repeat directive to draw a table:

```
<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">

      <td>{{ subject.name }}</td>

      <td>{{ subject.marks }}</td>

   </tr>

</table>
```

Table can be styled using CSS Styling.

```
<style>

table, th , td {

   border: 1px solid grey;

   border-collapse: collapse;

   padding: 5px;

}

table tr:nth-child(odd) {

   background-color: #f2f2f2;

}

table tr:nth-child(even) {

   background-color: #ffffff;
}

</style>
```

Example

The following example shows use of all the above mentioned directives.

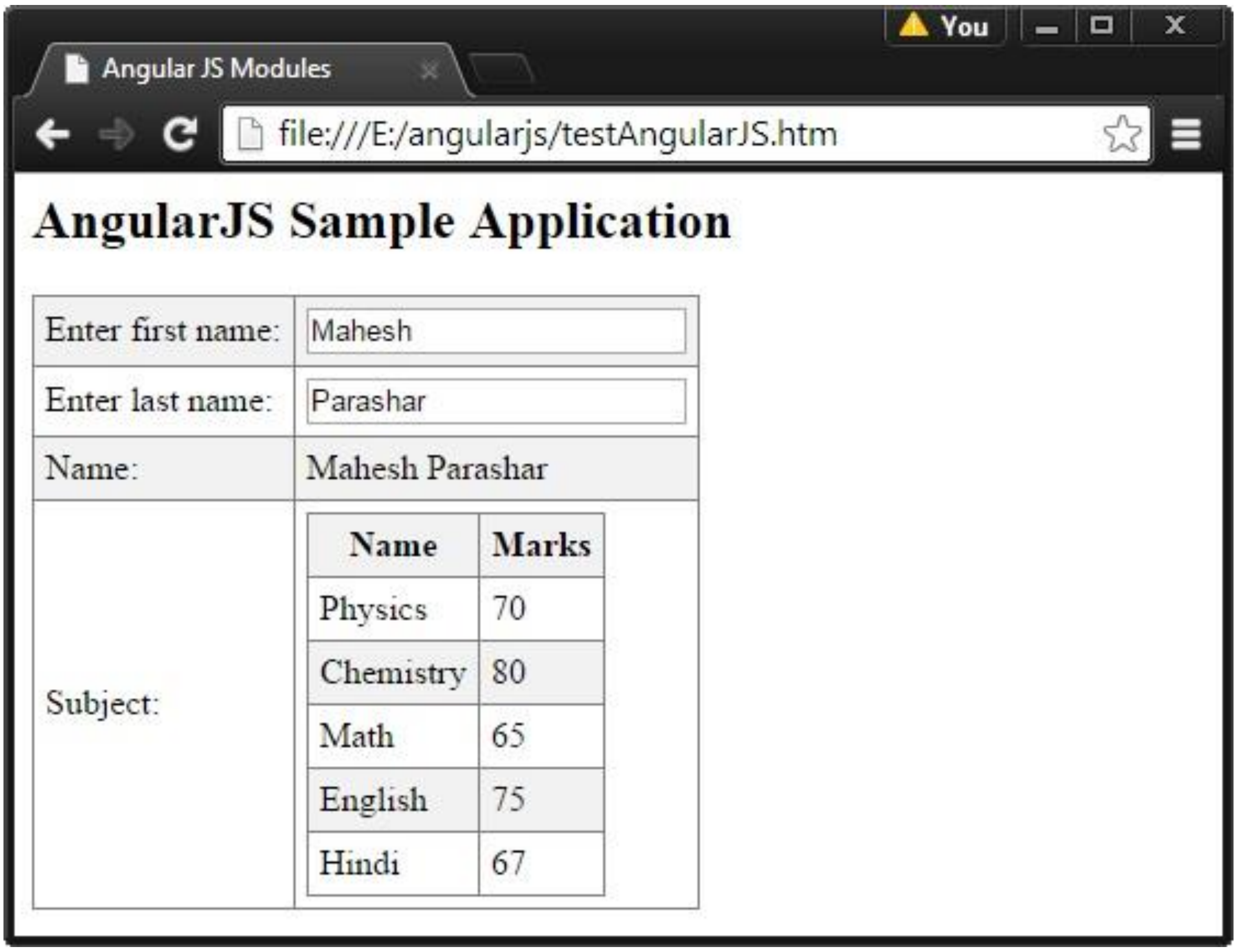
testAngularJS.htm

```
<html>

<head>

<title>Angular JS Table</title>

<style>

table, th , td {

  border: 1px solid grey;

  border-collapse: collapse;

  padding: 5px;

}

table tr:nth-child(odd) {

  background-color: #f2f2f2;

}

table tr:nth-child(even) {

  background-color: #ffffff;

}

</style>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="" ng-controller="studentController">

<table border="0">
```

```
<tr><td>Enter first name:</td><td><input type="text" ng-
model="student.firstName"></td></tr>

<tr><td>Enter last name: </td><td><input type="text" ng-
model="student.lastName"></td></tr>

<tr><td>Name: </td><td>{{student.fullName()}}</td></tr>

<tr><td>Subject:</td><td>

<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">

      <td>{{ subject.name }}</td>

      <td>{{ subject.marks }}</td>

   </tr>

</table>

</td></tr>

</table>

</div>

<script>

function studentController($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},
         {name:'Math',marks:65},

               {name:'English',marks:75},

               {name:'Hindi',marks:67}

      ],

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

}

</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



AngularJS Sample Application

Enter first name:	Mahesh
Enter last name:	Parashar
Name:	Mahesh Parashar
Subject:	

Name	Marks
Physics	70
Chemistry	80
Math	65
English	75
Hindi	67

10. HTML DOM

The following directives are used to bind application data to attributes of HTML DOM elements:

S.No.	Name	Description
1	ng-disabled	Disables a given control.
2	ng-show	Shows a given control.
3	ng-hide	Hides a given control.
4	ng-click	Represents a AngularJS click event.

ng-disabled Directive

Add ng-disabled attribute to an HTML button and pass it a model. Bind the model to a checkbox and see the variation.

```
<input type="checkbox" ng-model="enableDisableButton">Disable Button

<button ng-disabled="enableDisableButton">Click Me!</button>
```

ng-show Directive

Add ng-show attribute to an HTML button and pass it a model. Bind the model to a checkbox and see the variation.

```
<input type="checkbox" ng-model="showHide1">Show Button

<button ng-show="showHide1">Click Me!</button>
```

ng-hide Directive

Add ng-hide attribute to an HTML button and pass it a model. Bind the model to a checkbox and see the variation.

```
<input type="checkbox" ng-model="showHide2">Hide Button
```

```
<button ng-hide="showHide2">Click Me!</button>
```

ng-click Directive

Add ng-click attribute to an HTML button and update a model. Bind the model to HTML and see the variation.

```
<p>Total click: {{ clickCounter }}</p></td>

<button ng-click="clickCounter = clickCounter + 1">Click Me!</button>
```

Example

The following example shows use of all the above mentioned directives.

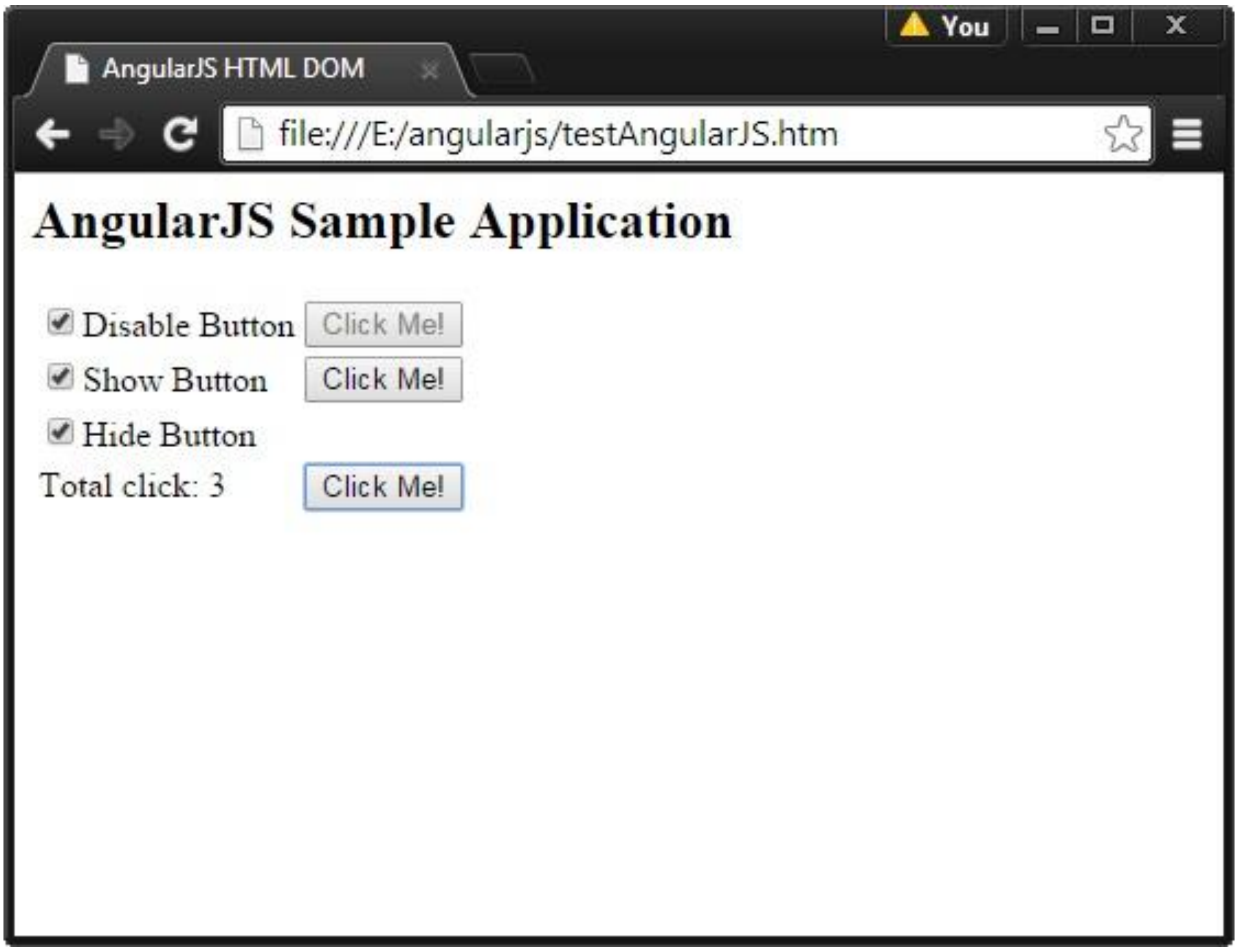
testAngularJS.htm

```
<html>

<head>

<title>AngularJS HTML DOM</title>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="">

<table border="0">

<tr>

   <td><input type="checkbox" ng-model="enableDisableButton">Disable
Button</td>

   <td><button ng-disabled="enableDisableButton">Click Me!</button></td>

</tr>

<tr>

   <td><input type="checkbox" ng-model="showHide1">Show Button</td>

   <td><button ng-show="showHide1">Click Me!</button></td>

</tr>
```

```
<tr>

   <td><input type="checkbox" ng-model="showHide2">Hide Button</td>

   <td><button ng-hide="showHide2">Click Me!</button></td>

</tr>

<tr>

   <td><p>Total click: {{ clickCounter }}</p></td>

   <td><button ng-click="clickCounter = clickCounter + 1">Click
Me!</button></td>

</tr>

</table>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



AngularJS Sample Application

Disable Button Click Me!

Show Button Click Me!

Hide Button

Total click: 3 Click Me!

11. MODULES

AngularJS supports modular approach. Modules are used to separate logic such as services, controllers, application etc. from the code and maintain the code clean. We define modules in separate js files and name them as per the module.js file. In the following example, we are going to create two modules:

- **Application Module** - used to initialize an application with controller(s).
- **Controller Module** - used to define the controller.

Application Module

Here is a file named *mainApp.js* that contains the following code:

```
var mainApp = angular.module("mainApp", []);
```

Here, we declare an application **mainApp** module using angular.module function and pass an empty array to it. This array generally contains dependent modules.

Controller Module

studentController.js

```
mainApp.controller("studentController", function($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},

         {name:'Math',marks:65},

         {name:'English',marks:75},

         {name:'Hindi',marks:67}
      ],

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

});
```

Here, we declare a controller **studentController** module using mainApp.controller function.

Use Modules

```
<div ng-app="mainApp" ng-controller="studentController">

..

<script src="mainApp.js"></script>

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

Here, we use application module using ng-app directive, and controller using ng-controller directive. We import the mainApp.js and studentController.js in the main HTML page.

Example

The following example shows use of all the above mentioned modules.

testAngularJS.htm

```
<html>

<head>

<title>Angular JS Modules</title>

<style>

table, th , td {

  border: 1px solid grey;
  border-collapse: collapse;

  padding: 5px;

}

table tr:nth-child(odd) {

  background-color: #f2f2f2;

}

table tr:nth-child(even) {

  background-color: #ffffff;

}

</style>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="mainApp" ng-controller="studentController">

<table border="0">

<tr><td>Enter first name:</td><td><input type="text" ng-
model="student.firstName"></td></tr>

<tr><td>Enter last name: </td><td><input type="text" ng-
model="student.lastName"></td></tr>

<tr><td>Name: </td><td>{{student.fullName()}}</td></tr>

<tr><td>Subject:</td><td>

<table>

   <tr>

      <th>Name</th>

      <th>Marks</th>

   </tr>

   <tr ng-repeat="subject in student.subjects">

      <td>{{ subject.name }}</td>
      <td>{{ subject.marks }}</td>

   </tr>

</table>

</td></tr>

</table>

</div>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

<script src="mainApp.js"></script>

<script src="studentController.js"></script>

</body>

</html>
```

mainApp.js

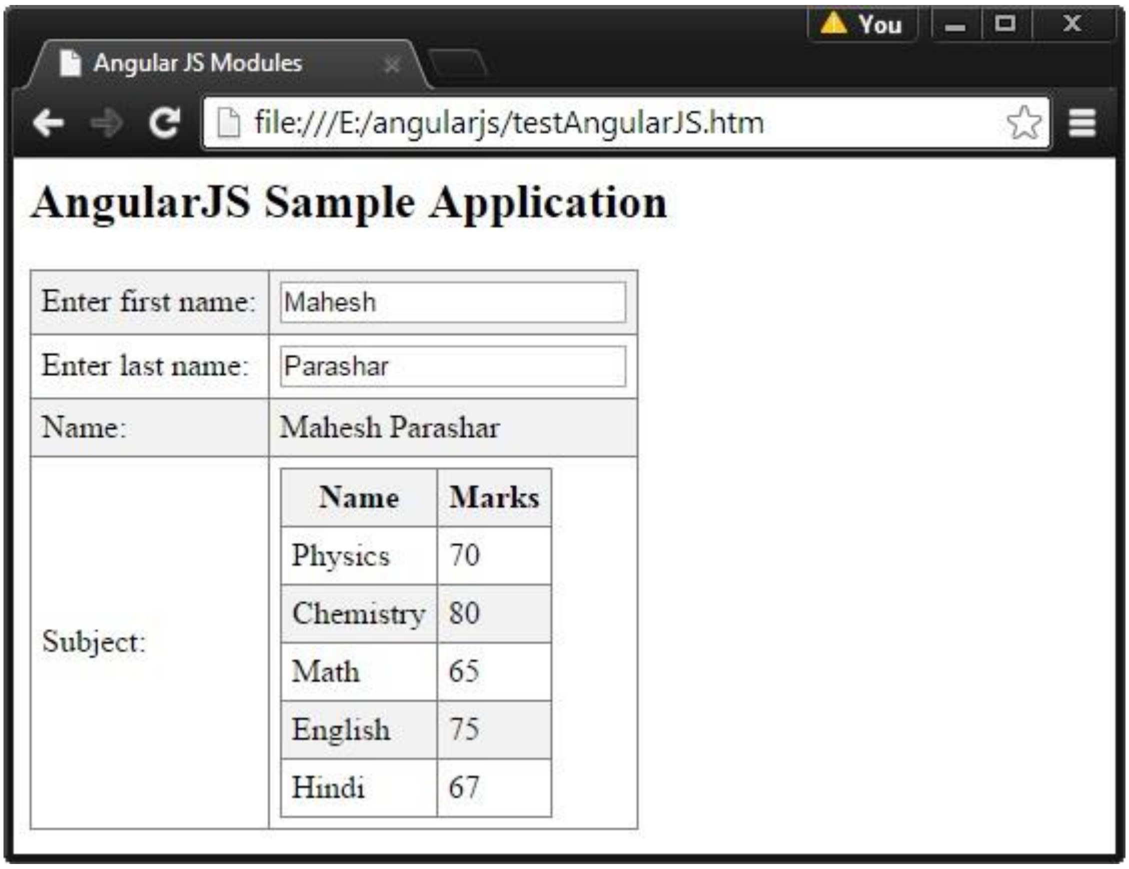
```
var mainApp = angular.module("mainApp", []);
```

studentController.js

```
mainApp.controller("studentController", function($scope) {

   $scope.student = {

      firstName: "Mahesh",

      lastName: "Parashar",

      fees:500,

      subjects:[

         {name:'Physics',marks:70},

         {name:'Chemistry',marks:80},

         {name:'Math',marks:65},

         {name:'English',marks:75},

         {name:'Hindi',marks:67}
      ],

      fullName: function() {

         var studentObject;

         studentObject = $scope.student;

         return studentObject.firstName + " " + studentObject.lastName;

      }

   };

});
```

Output

Open the file *textAngularJS.htm* in a web browser. See the result.



AngularJS Sample Application

Enter first name:	Mahesh
Enter last name:	Parashar
Name:	Mahesh Parashar
Subject:	

Name	Marks
Physics	70
Chemistry	80
Math	65
English	75
Hindi	67

12. FORMS

AngularJS enriches form filling and validation. We can use ng-click event to handle the click button and use $dirty and $invalid flags to do the validation in a seamless way. Use novalidate with a form declaration to disable any browser-specific validation. The form controls make heavy use of AngularJS events. Let us have a look at the events first.

Events

AngularJS provides multiple events associated with the HTML controls. For example, ng-click directive is generally associated with a button. AngularJS supports the following events:

- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress
- ng-change

Let us go through ng-click:

ng-click

Reset data of a form using on-click directive of a button.

```
<input name="firstname" type="text" ng-model="firstName" required>

<input name="lastname" type="text" ng-model="lastName" required>

<input name="email" type="email" ng-model="email" required>
```

```
<button ng-click="reset()">Reset</button>

<script>

   function studentController($scope) {

      $scope.reset = function(){

         $scope.firstName = "Mahesh";

         $scope.lastName = "Parashar";

         $scope.email = "MaheshParashar@tutorialspoint.com";

  }

   $scope.reset();

}

</script>
```

Validate Data

The following can be used to track error.

- **$dirty** - states that value has been changed.
- **$invalid**- states that value entered is invalid.
- **$error**- states the exact error.

Example

Following example will showcase all the above mentioned directives.

testAngularJS.htm

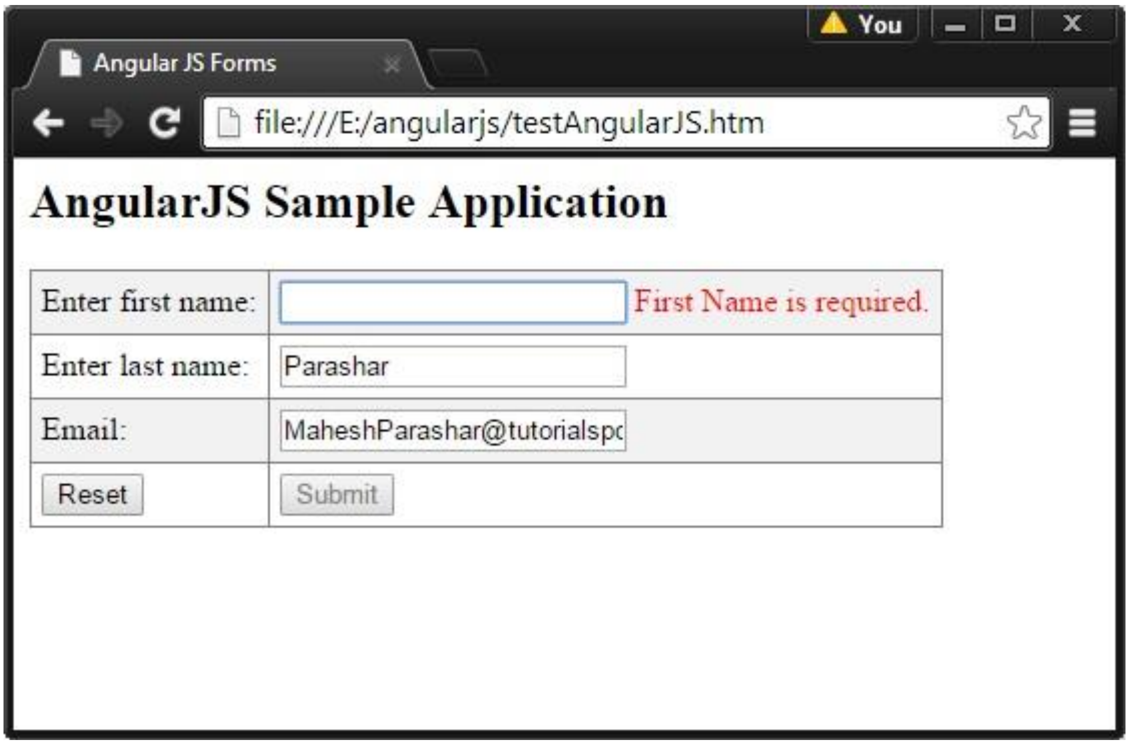
```
<html>

<head>

<title>Angular JS Forms</title>

<style>

table, th , td {

  border: 1px solid grey;

  border-collapse: collapse;
  padding: 5px;

}

table tr:nth-child(odd) {

  background-color: #f2f2f2;

}

table tr:nth-child(even) {

  background-color: #ffffff;

}

</style>

</head>

<body>

<h2>AngularJS Sample Application</h2>

<div ng-app="" ng-controller="studentController">

<form name="studentForm" novalidate>

<table border="0">

<tr><td>Enter first name:</td><td><input name="firstname" type="text" ng-
model="firstName" required>

   <span style="color:red" ng-show="studentForm.firstname.$dirty &&
studentForm.firstname.$invalid">

      <span ng-show="studentForm.firstname.$error.required">First Name is
required.</span>

   </span>

</td></tr>

<tr><td>Enter last name: </td><td><input name="lastname"  type="text" ng-
model="lastName" required>

   <span style="color:red" ng-show="studentForm.lastname.$dirty &&
studentForm.lastname.$invalid">

      <span ng-show="studentForm.lastname.$error.required">Last Name is
required.</span>

   </span>
```

```
</td></tr>

<tr><td>Email: </td><td><input name="email" type="email" ng-model="email"
length="100" required>

<span style="color:red" ng-show="studentForm.email.$dirty &&
studentForm.email.$invalid">

      <span ng-show="studentForm.email.$error.required">Email is
required.</span>

        <span ng-show="studentForm.email.$error.email">Invalid email
address.</span>

   </span>

</td></tr>

<tr><td><button ng-click="reset()">Reset</button></td><td><button

      ng-disabled="studentForm.firstname.$dirty &&
studentForm.firstname.$invalid ||

                        studentForm.lastname.$dirty &&
studentForm.lastname.$invalid ||

                        studentForm.email.$dirty &&
studentForm.email.$invalid"

      ng-click="submit()">Submit</button></td></tr>

</table>

</form>

</div>

<script>

function studentController($scope) {

   $scope.reset = function(){

              $scope.firstName = "Mahesh";

              $scope.lastName = "Parashar";

              $scope.email = "MaheshParashar@tutorialspoint.com";

   }

   $scope.reset();

}
```

```
</script>

<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

</body>

</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



AngularJS Sample Application

Enter first name:	First Name is required.
Enter last name:	Parashar
Email:	MaheshParashar@tutorialspo
Reset	Submit

17. SERVICES

AngularJS supports the concept of *Separation of Concerns* using services architecture. Services are JavaScript functions, which are responsible to perform only specific tasks. This makes them individual entities which are maintainable and testable. The controllers and filters can call them on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS.

AngularJS provides many inbuilt services. For example, $http, $route, $window, $location etc. Each service is responsible for a specific task such as the $http is used to make ajax call to get the server data, the $route is used to define the routing information, and so on. The inbuilt services are always prefixed with $ symbol.

There are two ways to create a service:

- Factory
- Service

Using Factory Method

In this method, we first define a factory and then assign method to it.

```
var mainApp = angular.module("mainApp", []);

mainApp.factory('MathService', function() {

   var factory = {};

   factory.multiply = function(a, b) {

      return a * b

   }

   return factory;

});
```

Using Service Method

In this method, we define a service and then assign method to it. We also inject an already available service to it.

```
mainApp.service('CalcService', function(MathService){
    this.square = function(a) {

              return MathService.multiply(a,a);

       }

});
```

Example

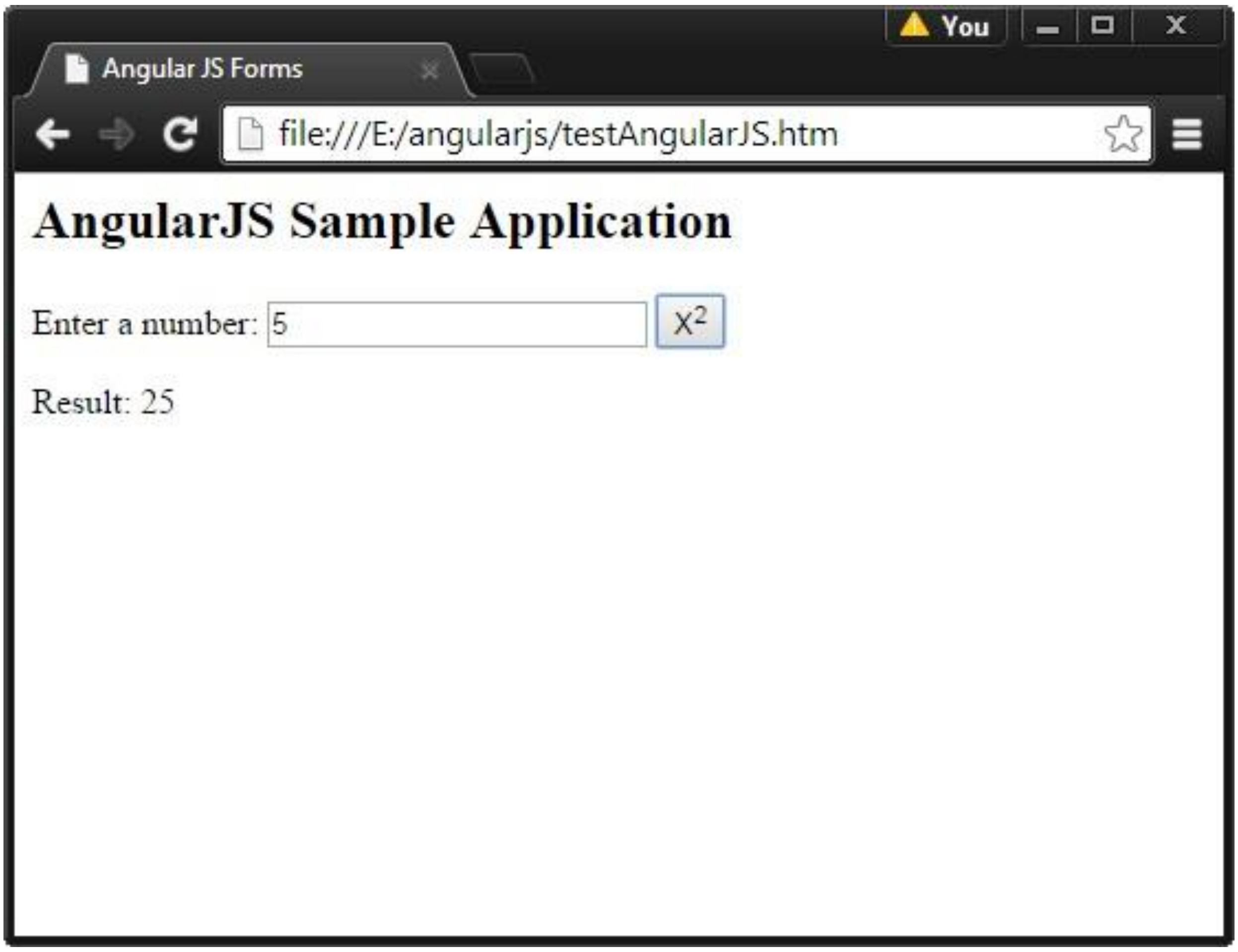
The following example shows use of all the above mentioned directives:

testAngularJS.htm

```
<html>

<head>

   <title>Angular JS Forms</title>

</head>

<body>

   <h2>AngularJS Sample Application</h2>

   <div ng-app="mainApp" ng-controller="CalcController">

      <p>Enter a number: <input type="number" ng-model="number" />

      <button ng-click="square()">X<sup>2</sup></button>

      <p>Result: {{result}}</p>

   </div>

   <script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js">
</script>

   <script>

      var mainApp = angular.module("mainApp", []);

      mainApp.factory('MathService', function() {

         var factory = {};

         factory.multiply = function(a, b) {

            return a * b
         }
         return factory;
      });

      mainApp.service('CalcService', function(MathService){
            this.square = function(a) {
            return MathService.multiply(a,a);
         }
      });

      mainApp.controller('CalcController', function($scope, CalcService) {
            $scope.square = function() {
            $scope.result = CalcService.square($scope.number);
         }
      });
   </script>
</body>
</html>
```

Output

Open the file *testAngularJS.htm* in a web browser and see the result.



AngularJS Sample Application

Enter a number: 5 X²

Result: 25

Chapter 1: Getting started with firebase

Remarks

Firebase is a Backend as a Service (Baas) very useful for mobile app development.

It provides many features like **Authentication & Security**, **Realtime Database & File Storage**, **Analytics**, **Push Notifications**, **AdMod** and many others

It provides the SDK for **Android, iOS, Web, NodeJS, C++ and Java Server**

Versions

Platform SDK	Version	Release date
Firebase JavaScript SDK	3.7.0	2017-03-01
Firebase C++ SDK	3.0.0	2107-02-27
Firebase Unity SDK	3.0.0	2107-02-27
Firebase iOS SDK	3.14.0	2017-02-23
Firebase Android SDK	10.2	2017-02-15
Firebase Admin Node.js SDK	4.1.1	2017-02-14
Firebase Admin Java SDK	4.1.2	2017-02-14

Examples

Add Firebase to Your Android Project

Here the steps required to create a Firebase project and to connect with an Android app.

Add Firebase to your app

1. Create a Firebase project in the Firebase console and click **Create New Project**.

2. Click **Add Firebase to your Android app** and follow the setup steps.

3. When prompted, enter your **app's package name**.
   It's important to enter the package name your app is using; this can only be set when you add an app to your Firebase project.

4. To add debug signing certificate SHA1 which is **required for Dynamic Links, Invites, and Google Sign-In support in Auth,** go to your project in Android Studio, click on `Gradle` tab on the right side of your window, click on `Refresh` button, go to `project(root)` -> `Tasks` -> `android` -> `signingReport`. This will generate **MD5** and **SHA1** both in `Run` tab. Copy paste SHA1 into firebase console.

5. At the end, you'll download a `google-services.json` file. You can download this file again at any time.

6. If you haven't done so already, copy this into your project's module folder, typically app/.

The next step is to Add the SDK to integrate the Firebase libraries in the project.

Add the SDK

To integrate the Firebase libraries into one of your own projects, you need to perform a few basic tasks to prepare your Android Studio project. You may have already done this as part of adding Firebase to your app.

1. Add rules to your root-level `build.gradle` file, to include the **google-services plugin**:

```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.0.0'
    }
}
```

Then, in your module Gradle file (usually the `app/build.gradle`), add the apply plugin line at the bottom of the file to enable the Gradle plugin:

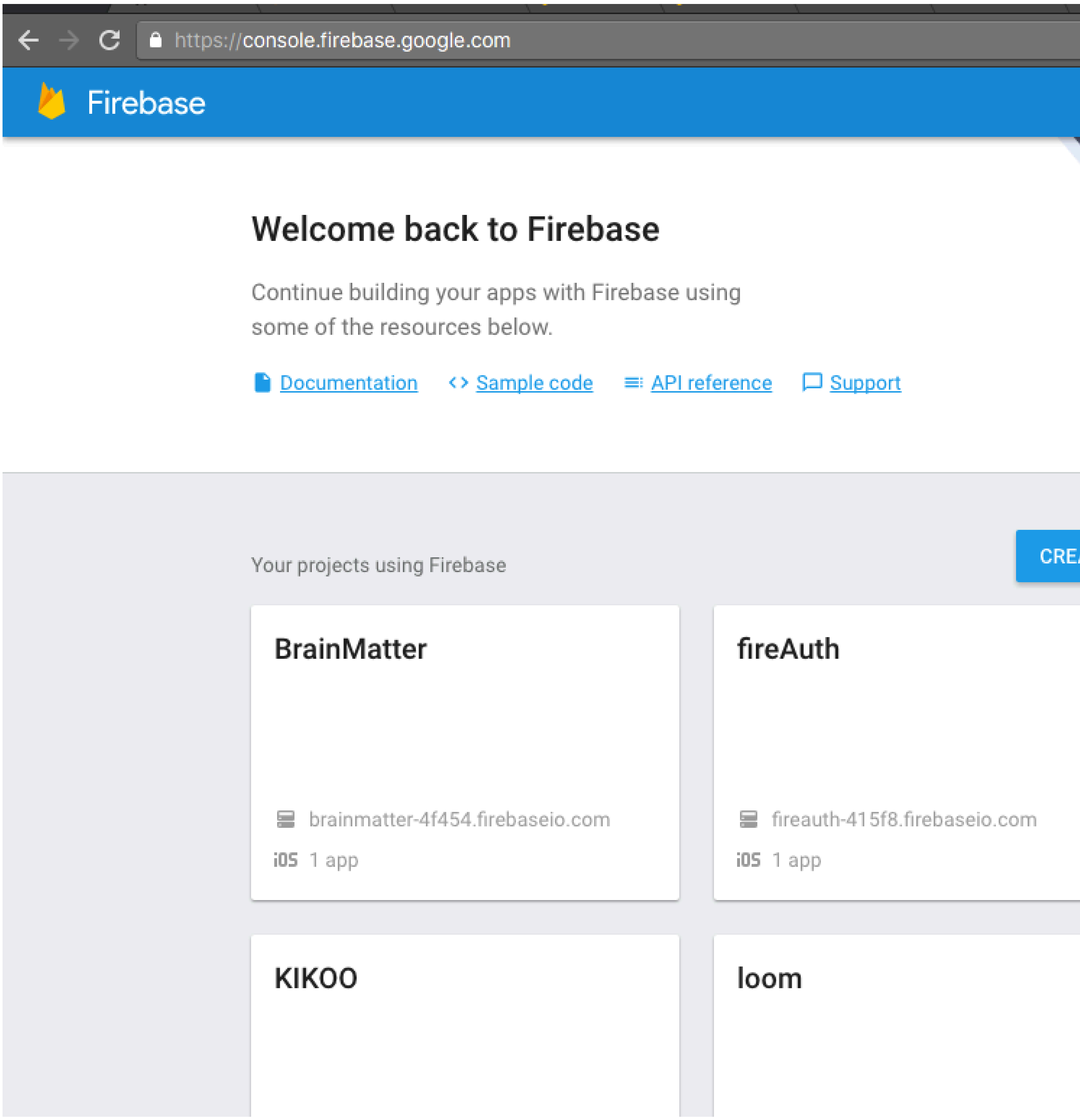
```
apply plugin: 'com.android.application'

android {
  // ...
}

dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:9.4.0'
}

// ADD THIS AT THE BOTTOM
apply plugin: 'com.google.gms.google-services'
```

The final step is to add the dependencies for the Firebase SDK using one or more **libraries available** for the different Firebase features.

Gradle Dependency Line	Service
com.google.firebase:firebase-core:9.4.0	Analytics
com.google.firebase:firebase-database:9.4.0	Realtime Database
com.google.firebase:firebase-storage:9.4.0	Storage
com.google.firebase:firebase-crash:9.4.0	Crash Reporting
com.google.firebase:firebase-auth:9.4.0	Authentication
com.google.firebase:firebase-messaging:9.4.0	Cloud Messaging / Notifications
com.google.firebase:firebase-config:9.4.0	Remote Config
com.google.firebase:firebase-invites:9.4.0	Invites / Dynamic Links
com.google.firebase:firebase-ads:9.4.0	AdMob
com.google.android.gms:play-services-appindexing:9.4.0	App Indexing

Setting up Firebase for IOS

1. Firstly, you want to go to firebase dashboard and create a new project using the 'Create New Project' button.


https://console.firebase.google.com

Firebase

Welcome back to Firebase

Continue building your apps with Firebase using some of the resources below.

Documentation   Sample code   API reference   Support

Your projects using Firebase

CRE

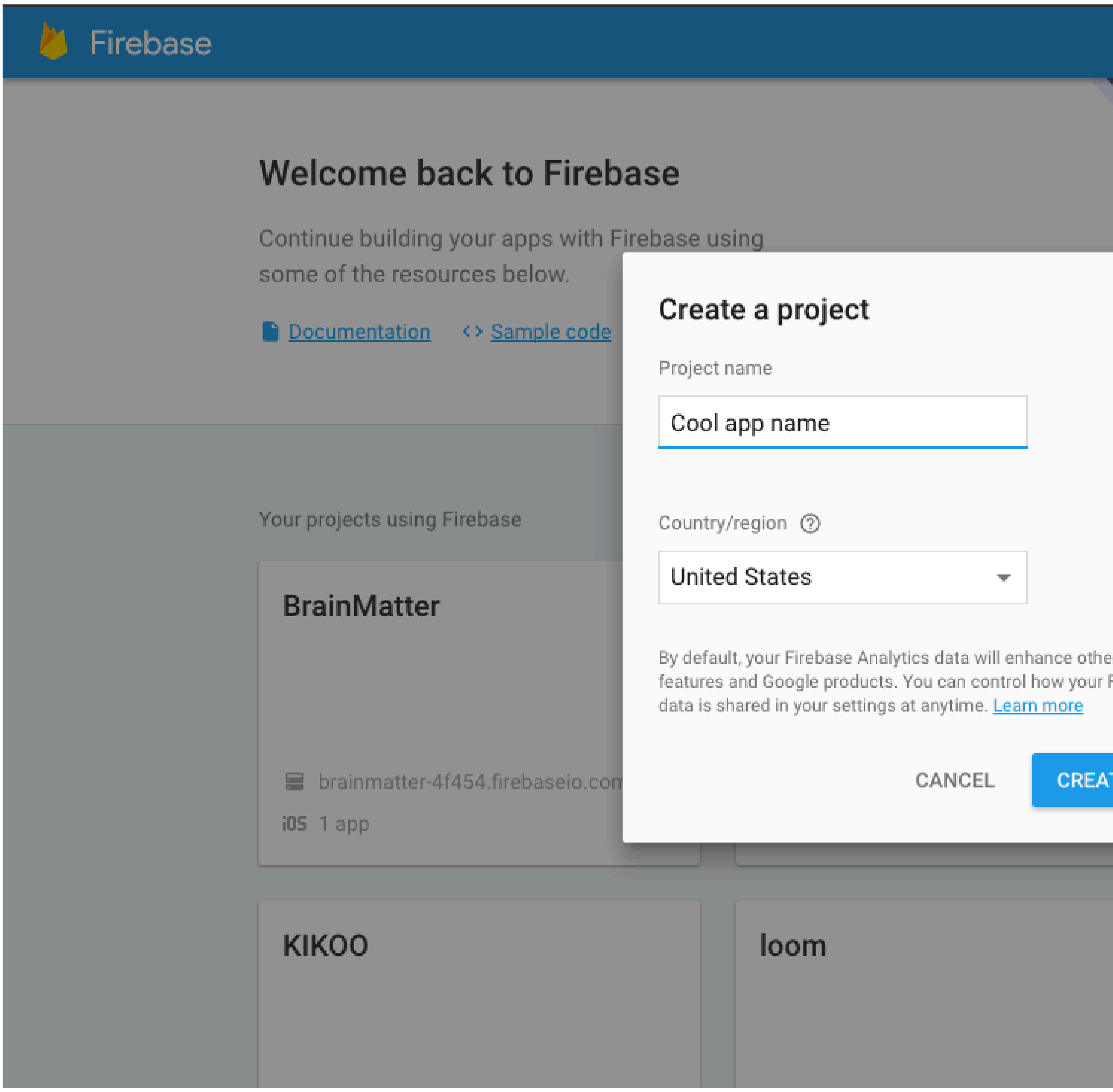
BrainMatter
brainmatter-4f454.firebaseio.com
iOS 1 app

fireAuth
fireauth-415f8.firebaseio.com
iOS 1 app

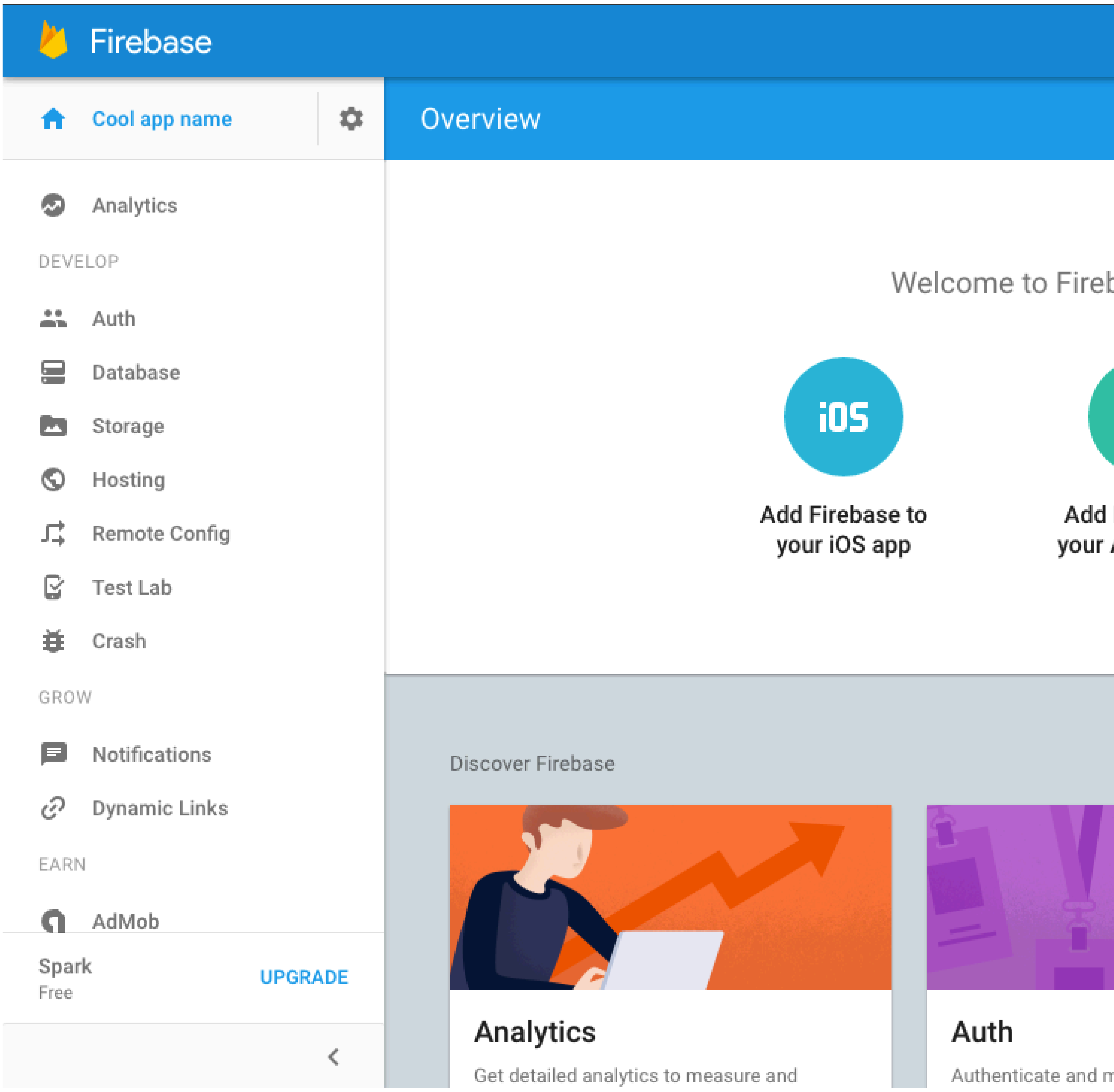
KIKOO

loom

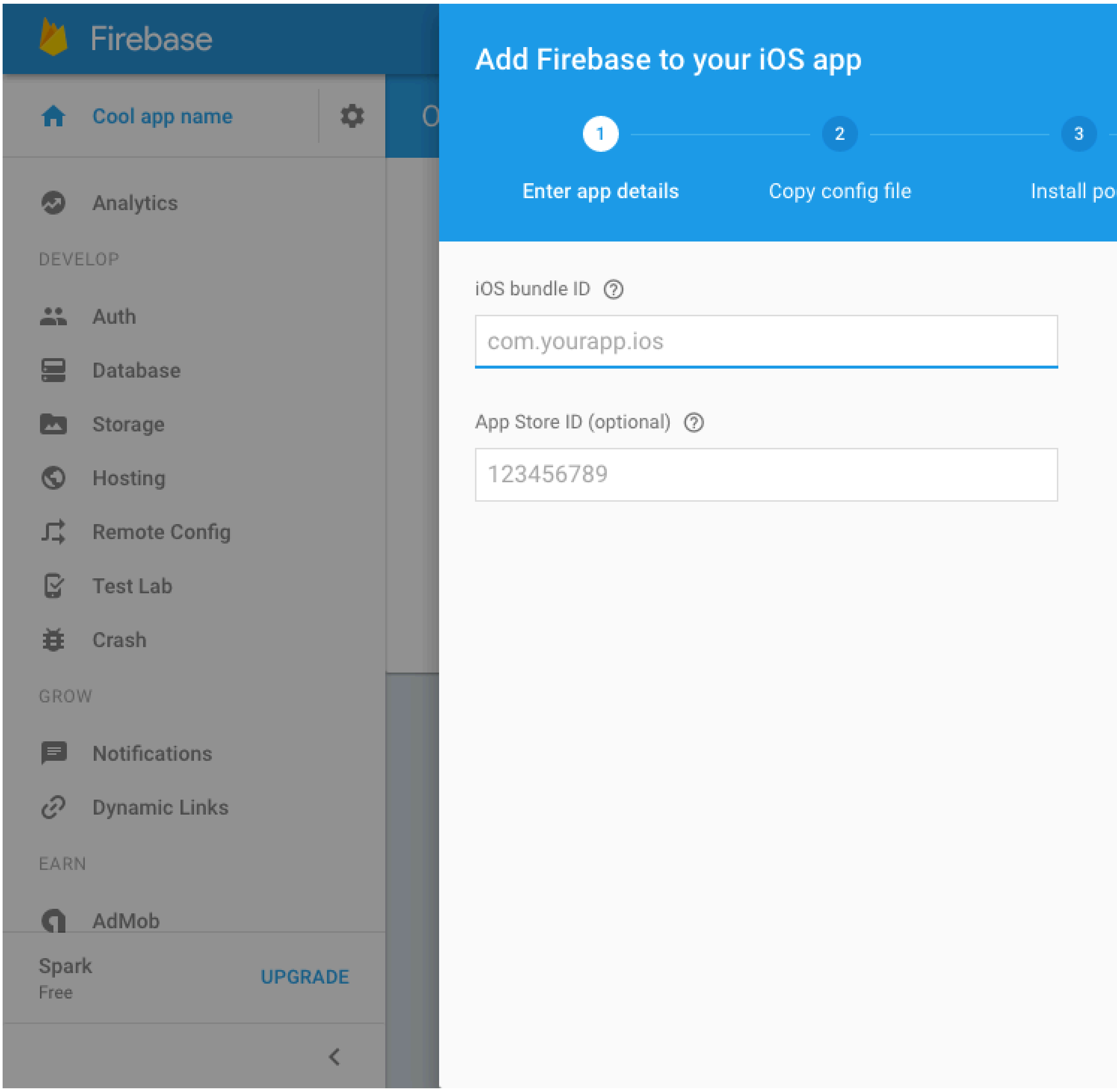
2. You want to create a new project by adding the name of your app for example I put mine as 'Cool app name' then choose your region and press 'Create Project'


Firebase
Welcome back to Firebase
Continue building your apps with Firebase using some of the resources below.
Documentation
Sample code
Your projects using Firebase
BrainMatter
brainmatter-4f454.firebaseio.com
iOS 1 app
KIKOO
loom
Create a project
Project name
Cool app name
Country/region
United States
By default, your Firebase Analytics data will enhance othe
features and Google products. You can control how your F
data is shared in your settings at anytime. Learn more
CANCEL
CREAT

3. After creating project you will be directed to this page which is the dashboard and from here you have to pick a platform which you want to install firebase to for this example we will choose IOS.



4. After selecting IOS you should see the same pop up as the one from the image below asking for the IOS Bundle and the app store id. You will only need to provide the IOS Bundle because our app isn't on the app store yet.



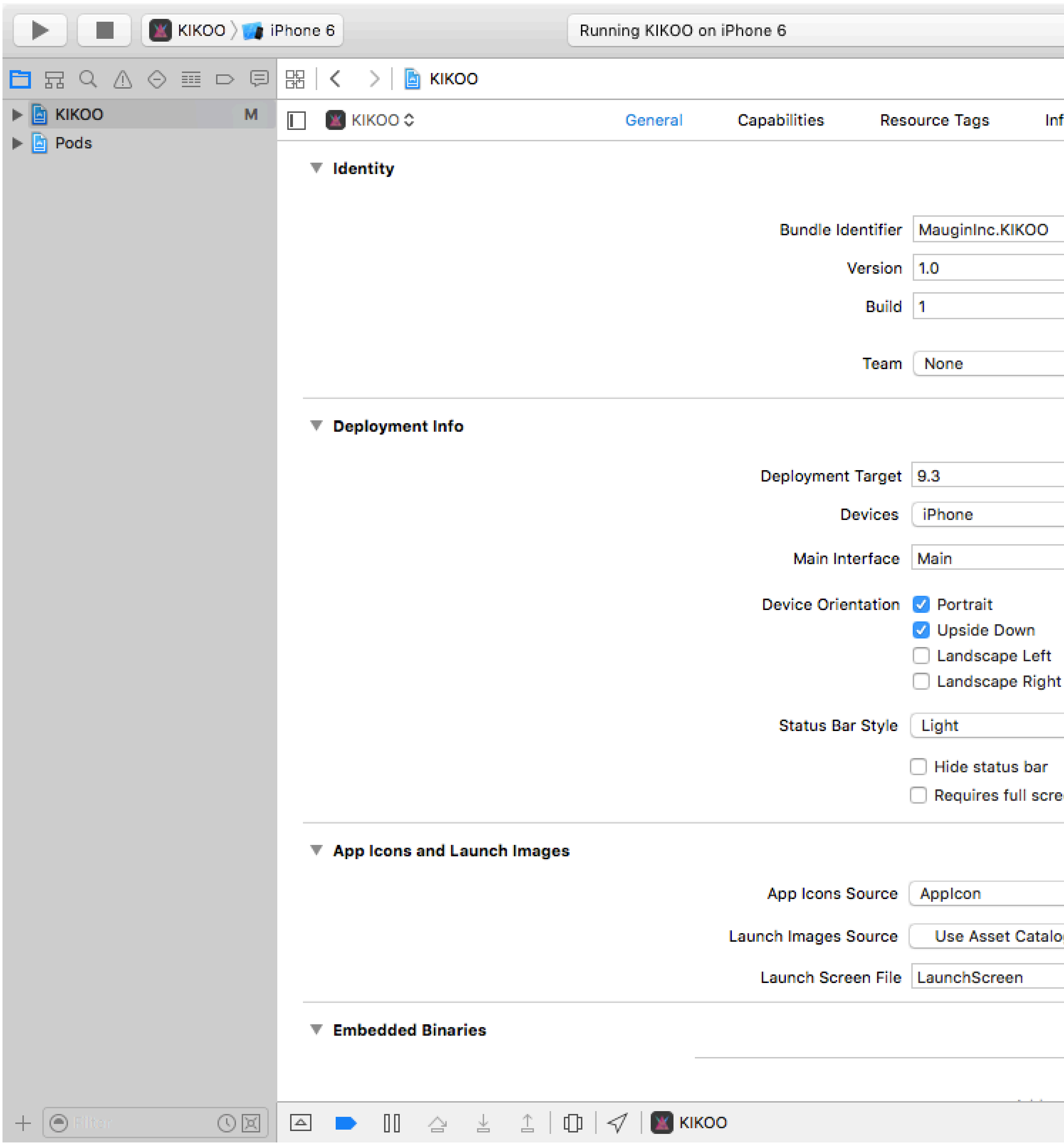
iOS bundle ID

com.yourapp.ios

App Store ID (optional)

123456789

5. Get the bundle ID from xcode after creating a xcode project anyway you usually would after that you can get the bundle id for your application on the app Genral view in xcode it will be the first field at the top and once you get it paste it into the Bundle field in firebase for example mine would be 'MauginInc.KIKOO'



Identity

Bundle Identifier: MauginInc.KIKOO
Version: 1.0
Build: 1
Team: None

Deployment Info

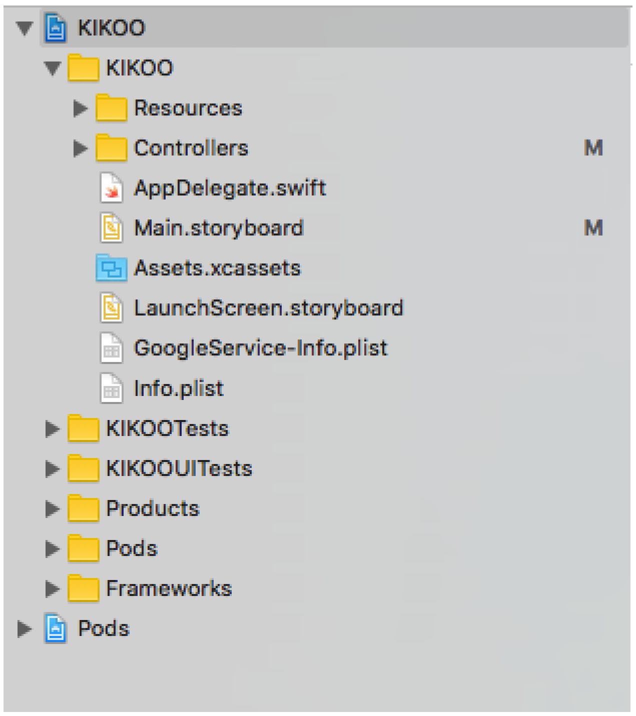
Deployment Target: 9.3
Devices: iPhone
Main Interface: Main
Device Orientation: [✓ Portrait] [✓ Upside Down] Landscape Left, Landscape Right
Status Bar Style: Light
Hide status bar
Requires full scre

App Icons and Launch Images

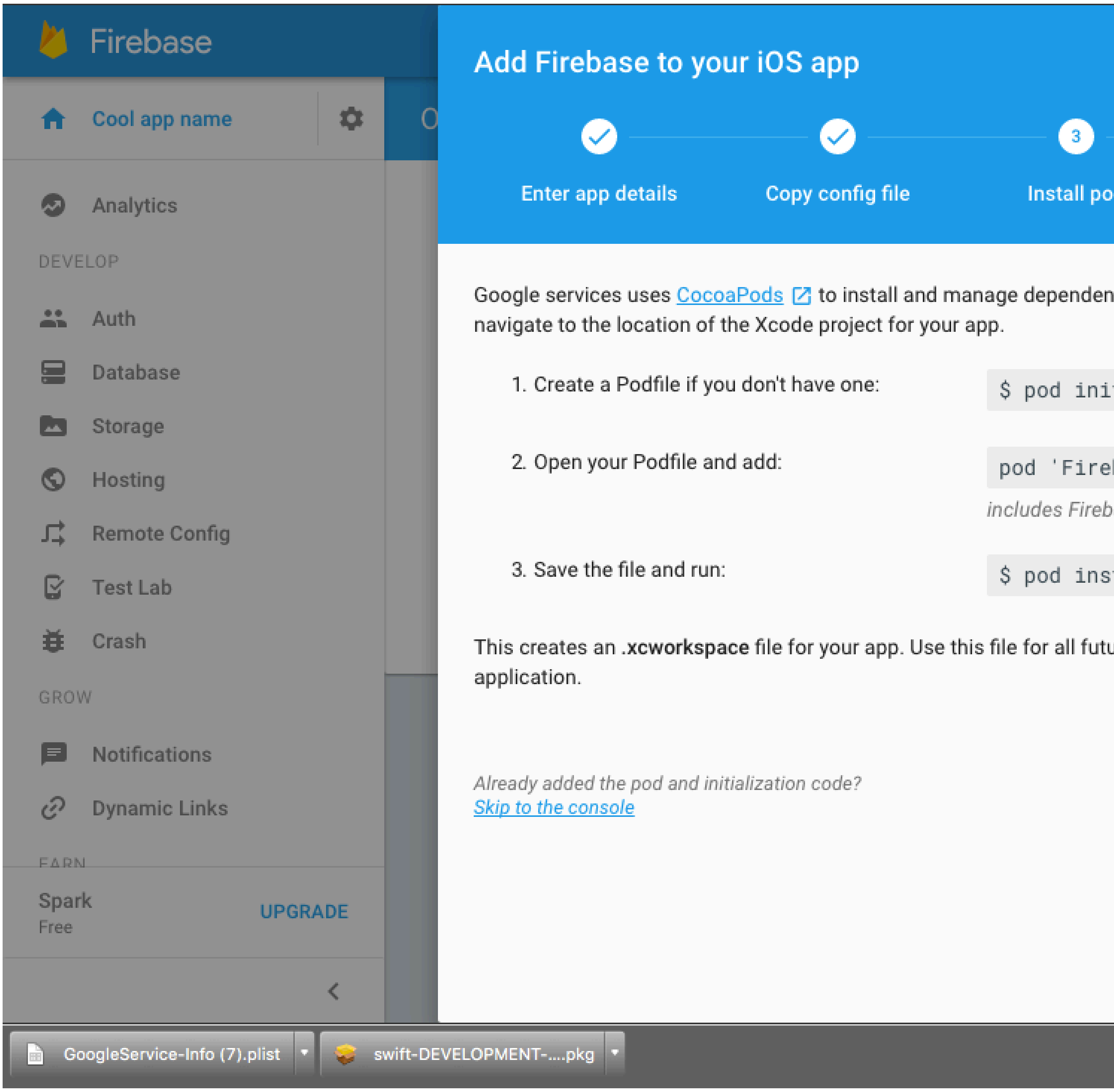
App Icons Source: AppIcon
Launch Images Source: Use Asset Catalo
Launch Screen File: LaunchScreen

Embedded Binaries

6. After you have done that and pressed 'Next' a 'GoogleService-Info.plist' file will download and what you will need to do is move that into the root folder of your app within xcode



7. You will want to initialise pods and install the firebase pods you need you cam do this by going into your terminal and navigate to your xcode project folder and follow these instructions given by firebase.



Add Firebase to your iOS app

Google services uses CocoaPods to install and manage dependen
navigate to the location of the Xcode project for your app.

1. Create a Podfile if you don't have one:

```
$ pod ini
```

2. Open your Podfile and add:

```
pod 'Fireb
```

includes Fireb

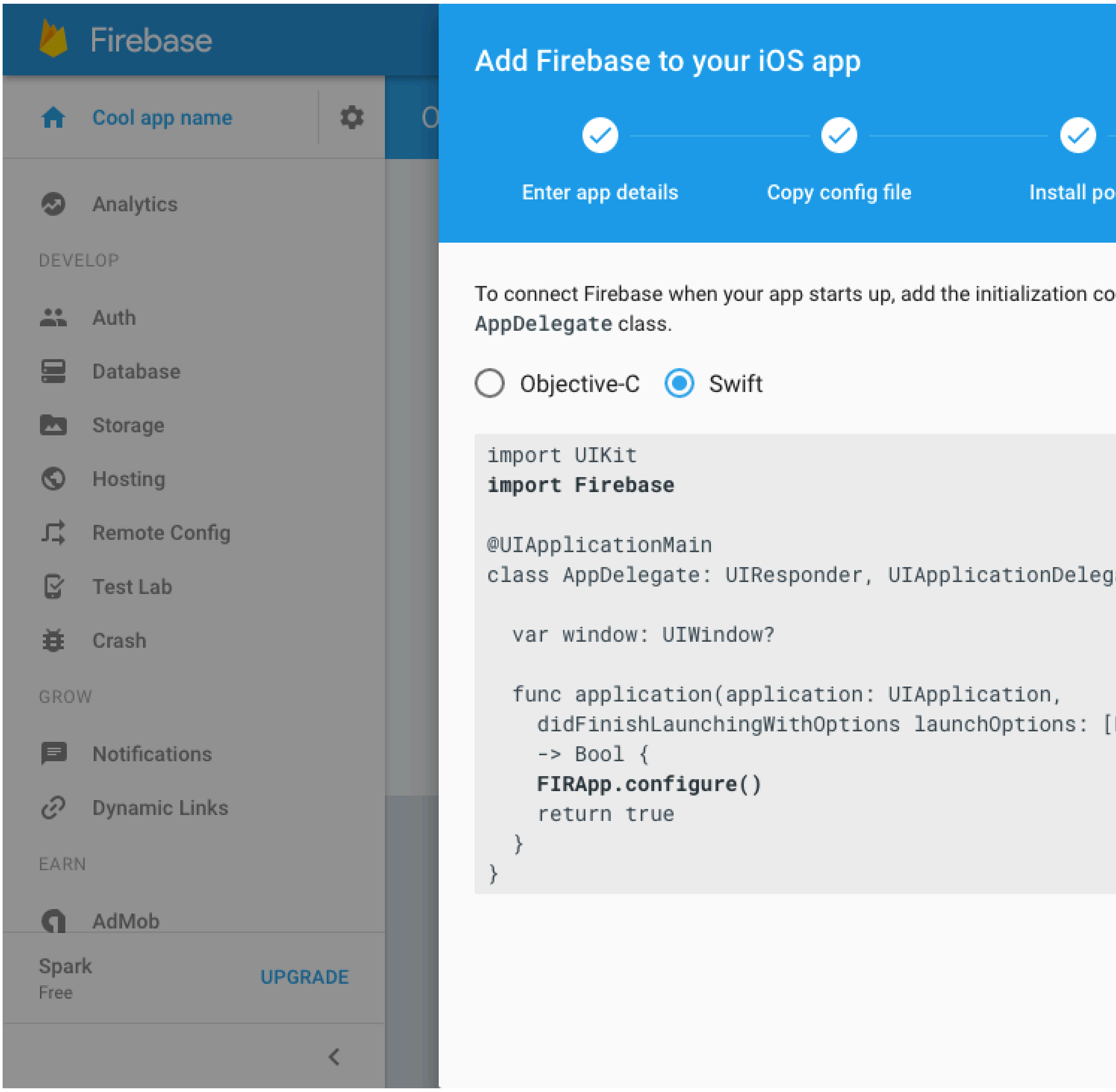
3. Save the file and run:

```
$ pod ins
```

This creates an **.xcworkspace** file for your app. Use this file for all futu
application.

Already added the pod and initialization code?
Skip to the console

8. Finally you want to configure you app to let swift do what it does best and that is making app development a whole lot more easier and efficient all you need to do is edit you AppDelegate.swift files the same the pop up shows you.



To connect Firebase when your app starts up, add the initialization co
AppDelegate class.

( ) Objective-C [✓ Swift]

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDeleg

  var window: UIWindow?

  func application(application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [
    -> Bool {
    FIRApp.configure()
    return true
  }
}
```

That's all you now have firebase installed in your xcode project for IOS

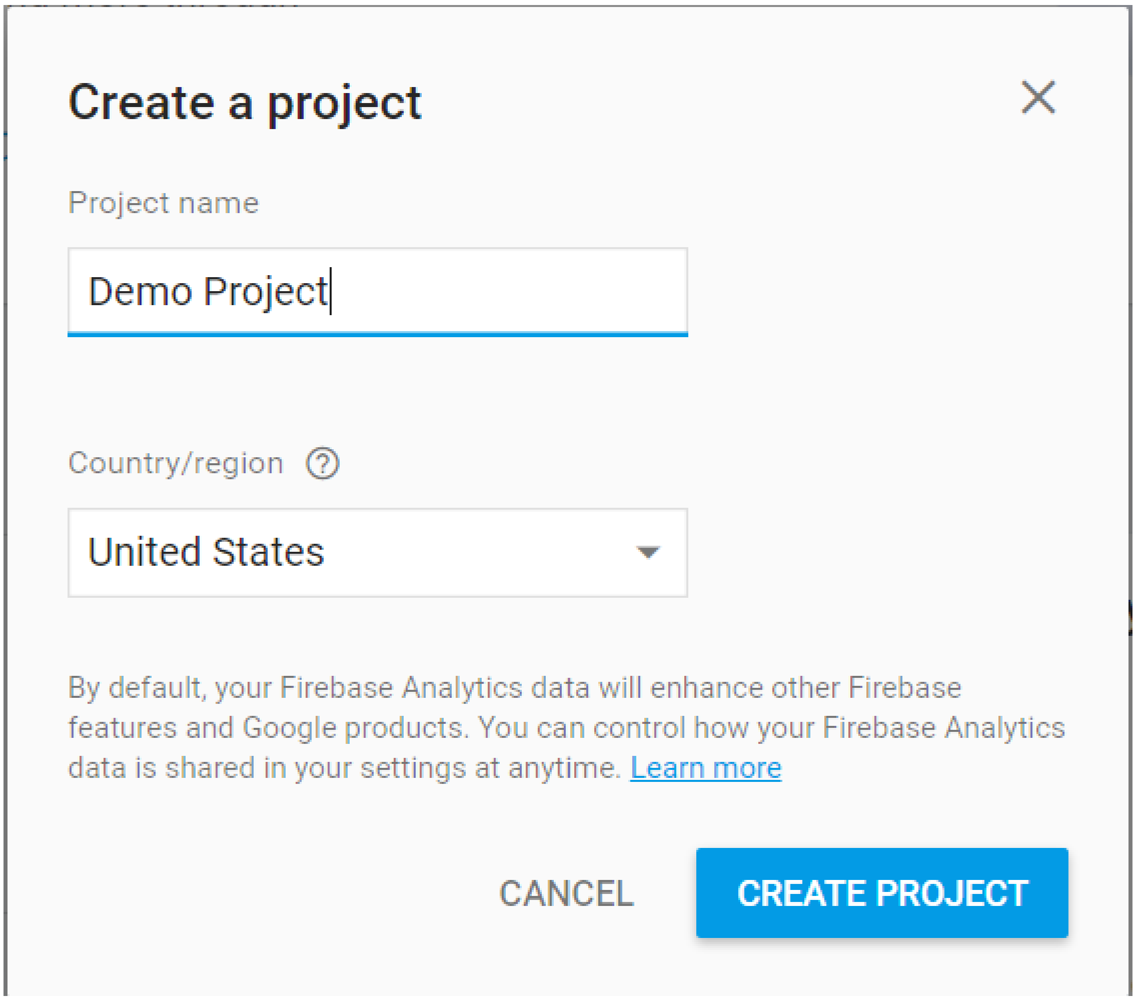
Getting started in Firebase with a simple Hello World web app in JavaScript

This example will demonstrate how to get started with Firebase in your web apps with JavaScript.

We are going to add a **text child** in our Firebase Database and display it in realtime on our web app.

Lets get started.

- Go to the Firebase Console - https://console.firebase.google.com and create a new project. Enter the project name, Country/region and click on **create project**.



Create a project

Project name: Demo Project

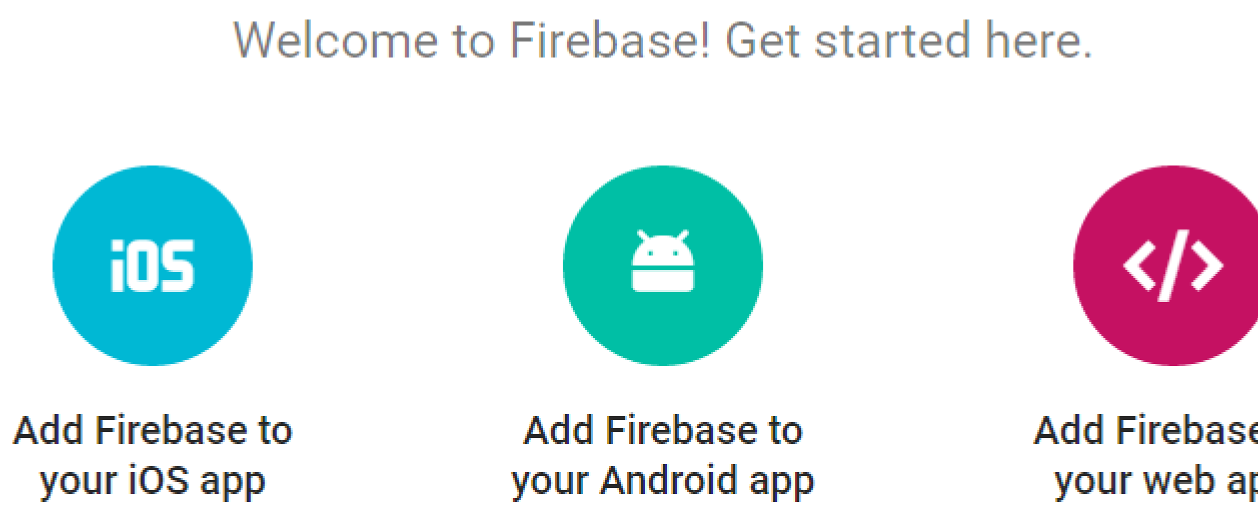
Country/region: United States

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at anytime. Learn more

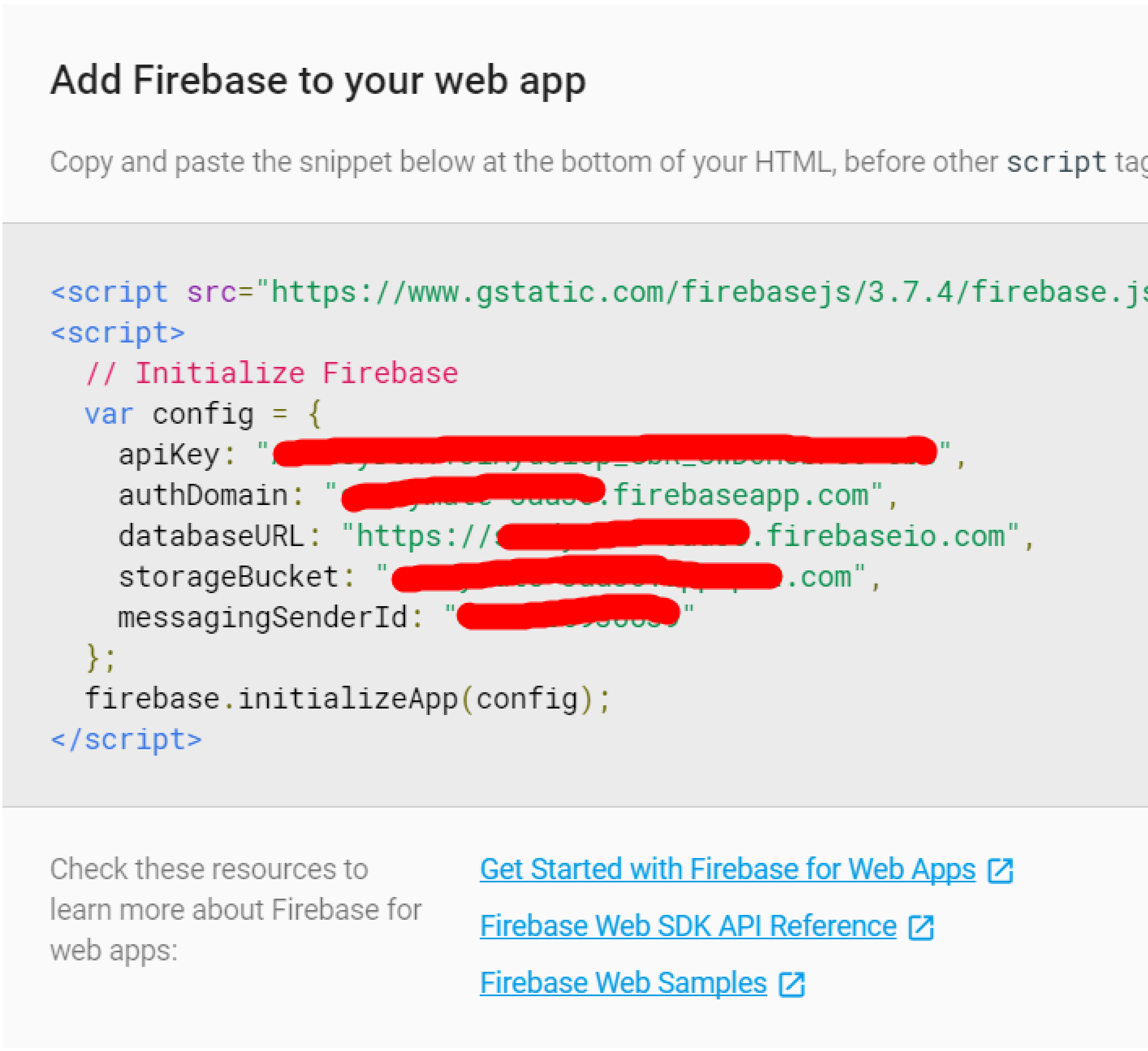
- Now create a file **index.html** on your computer. And add the following code to it.

```
<body>
    <p>Getting started with Firebase</p>
    <h1 id="bigOne"></h1>
    <script>
          // your firebase JavaScript code here
    </script>

</body>
```

- Now go to your project on Firebase Console and you can see this


Welcome to Firebase! Get started here.
iOS
Add Firebase to your iOS app
Add Firebase to your Android app
Add Firebase your web ap

- Now click on **Add Firebase to your web app**. You will the following pop up, click on copy button


Add Firebase to your web app

Copy and paste the snippet below at the bottom of your HTML, before other `script` tag

```
<script src="https://www.gstatic.com/firebasejs/3.7.4/firebase.js
<script>
  // Initialize Firebase
  var config = {
    apiKey: "[redacted]",
    authDomain: "[redacted].firebaseapp.com",
    databaseURL: "https://[redacted].firebaseio.com",
    storageBucket: "[redacted].com",
    messagingSenderId: "[redacted]"
  };
  firebase.initializeApp(config);
</script>
```

Check these resources to learn more about Firebase for web apps:

Get Started with Firebase for Web Apps

Firebase Web SDK API Reference

Firebase Web Samples

- Now go to your **index.html** file and add the snippet to the script section as following

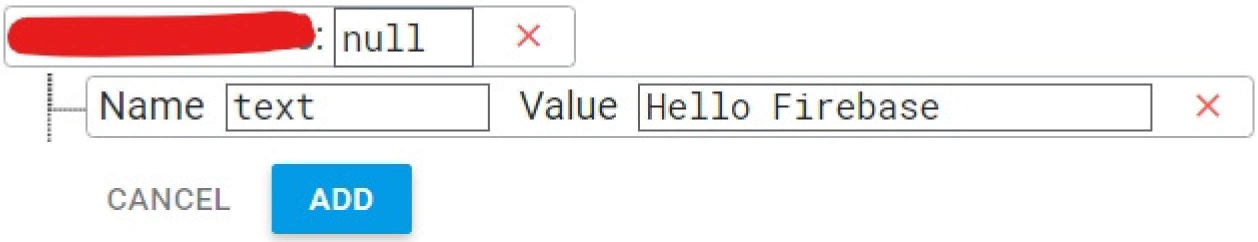
```
<body>

  <p>Getting started with Firebase</p>
  <h1 id="bigOne"></h1>

  <script src="https://www.gstatic.com/firebasejs/3.7.4/firebase.js"></script>
  <script>
    // Initialize Firebase
    var config = {
      apiKey: "apiKey",
      authDomain: "authDomain",
      databaseURL: "databaseURL",
      storageBucket: "storageBucket",
      messagingSenderId: "messagingSenderId"
    };
    firebase.initializeApp(config);
  </script>
```

```
</body>
```

- Now you have completed adding Firebase initialization code. Now we need to get our **text** value from the database.
- To do that add the following code (Initialize Firebase already added in last step. Don't re-add) inside the script in **index.html**

```
<script>

    // Initialize Firebase
    var config = {
      apiKey: "apiKey",
      authDomain: "authDomain",
      databaseURL: "databaseURL",
      storageBucket: "storageBucket",
      messagingSenderId: "messagingSenderId"
    };
    firebase.initializeApp(config);

    // getting the text value from the database
    var bigOne = document.getElementById('bigOne');
    var dbRef = firebase.database().ref().child('text');
    dbRef.on('value', snap => bigOne.innerText = snap.val());

</script>
```

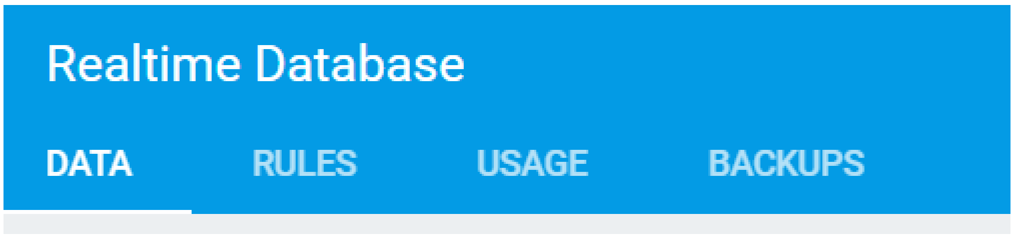
- Now we are all done with the **index.html** file and now let's go the **Database** in Firebase Console.
- You will see that its blank and empty right now. Lets add the a **text child** in the database and add any value to it.



[redacted]: null

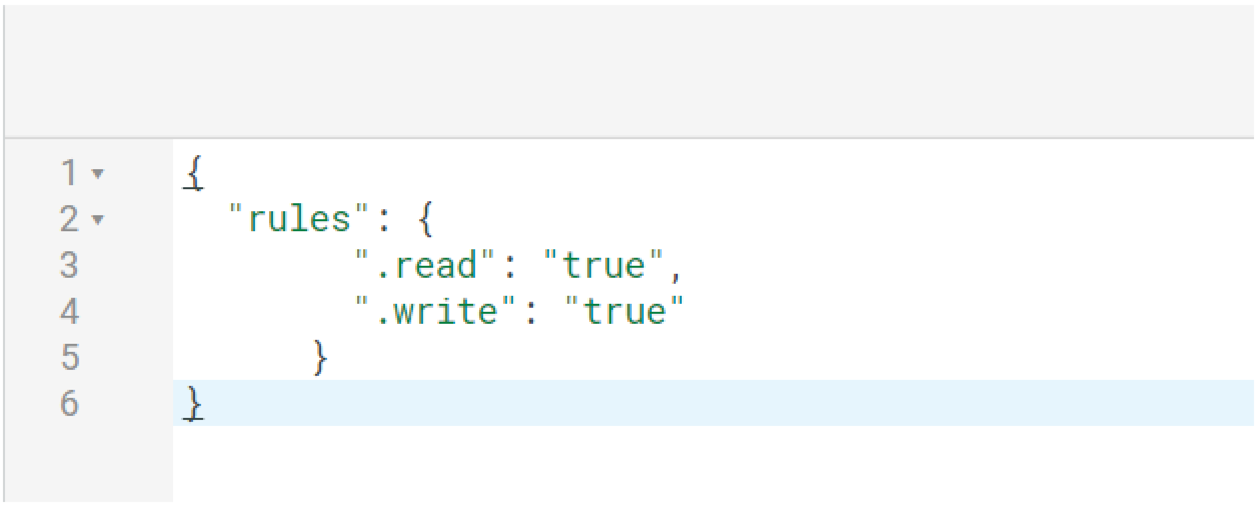
Name text Value Hello Firebase

- Now click on **ADD** button.
- Now go the **RULES** section in the Database.



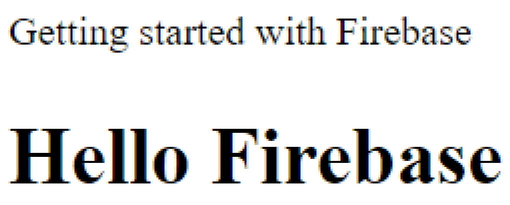
- For development purpose right now, we will right now enable all the **read and write** queries.

```
{
  "rules": {
      ".read": "true",
      ".write": "true"
    }
}
```



```
{
   "rules": {
         ".read": "true",
         ".write": "true"
      }
}
```

- Now open index.html in the browser
- You will see the text value on your page as following -



Getting started with Firebase

Hello Firebase

- Now if you go back to your database and change the **text child** value to something else, you will see that the text in the browser also changes without any refresh or reload. This is how **realtime database** works on Firebase.

Read Getting started with firebase online: https://riptutorial.com/firebase/topic/816/getting-started-with-firebase

Chapter 1: Getting started with Docker

Remarks

Docker is an open-source project that automates the deployment of applications inside software containers. These application containers are similar to lightweight virtual machines, as they can be run in isolation to each other and the running host.

Docker requires features present in recent linux kernels to function properly, therefore on Mac OSX and Windows host a virtual machine running linux is required for docker to operate properly. Currently the main method of installing and setting up this virtual machine is via Docker Toolbox that is using VirtualBox internally, but there are plans to integrate this functionality into docker itself, using the native virtualisation features of the operating system. On Linux systems docker run natively on the host itself.

Versions

Version	Release Date
17.05.0	2017-05-04
17.04.0	2017-04-05
17.03.0	2017-03-01
1.13.1	2016-02-08
1.12.0	2016-07-28
1.11.2	2016-04-13
1.10.3	2016-02-04
1.9.1	2015-11-03
1.8.3	2015-08-11
1.7.1	2015-06-16
1.6.2	2015-04-07
1.5.0	2015-02-10

Examples

Installing Docker on Mac OS X

Requirements: OS X 10.8 “Mountain Lion” or newer required to run Docker.

While the docker binary can run natively on Mac OS X, to build and host containers you need to run a Linux virtual machine on the box.

1.12.0

Since version 1.12 you don't need to have a separate VM to be installed, as Docker can use the native `Hypervisor.framework` functionality of OSX to start up a small Linux machine to act as backend.

To install docker follow the following steps:

1. Go to Docker for Mac
2. Download and run the installer.
3. Continue through installer with default options and enter your account credentials when requested.

Check here for more information on the installation.

1.11.2

Until version 1.11 the best way to run this Linux VM is to install Docker Toolbox, that installs Docker, VirtualBox and the Linux guest machine.

To install docker toolbox follow the following steps:

1. Go to Docker Toolbox
2. Click the link for Mac and run the installer.
3. Continue through installer with default options and enter your account credentials when requested.

This will install the Docker binaries in `/usr/local/bin` and update any existing Virtual Box installation. Check here for more information on the installation.

To Verify Installation:

1.12.0

1. Start `Docker.app` from the Applications folder, and make sure it is running. Next open up Terminal.

1.11.2

1. Open the `Docker Quickstart Terminal`, which will open a terminal and prepare it for use for Docker commands.

2. Once the terminal is open type

```
$ docker run hello-world
```

3. If all is well then this should print a welcome message verifying that the installation was successful.

Installing Docker on Windows

Requirements: 64-bit version of Windows 7 or higher on a machine which supports Hardware Virtualization Technology, and it is enabled.

While the docker binary can run natively on Windows, to build and host containers you need to run a Linux virtual machine on the box.

1.12.0

Since version 1.12 you don't need to have a separate VM to be installed, as Docker can use the native Hyper-V functionality of Windows to start up a small Linux machine to act as backend.

To install docker follow the following steps:

1. Go to Docker for Windows
2. Download and run the installer.
3. Continue through installer with default options and enter your account credentials when requested.

Check here for more information on the installation.

1.11.2

Until version 1.11 the best way to run this Linux VM is to install Docker Toolbox, that installs Docker, VirtualBox and the Linux guest machine.

To install docker toolbox follow the following steps:

1. Go to Docker Toolbox
2. Click the link for Windows and run the installer.
3. Continue through installer with default options and enter your account credentials when requested.

This will install the Docker binaries in Program Files and update any existing Virtual Box installation. Check here for more information on the installation.

To Verify Installation:

1.12.0

1. Start `Docker` from the Start menu if it hasn't been started yet, and make sure it is running. Next upen up any terminal (either `cmd` or PowerShell)

1.11.2

1. On your Desktop, find the Docker Toolbox icon. Click the icon to launch a Docker Toolbox terminal.

2. Once the terminal is open type

   ```
   docker run hello-world
   ```

3. If all is well then this should print a welcome message verifying that the installation was successful.

Installing docker on Ubuntu Linux

Docker is supported on the following *64-bit* versions of Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

A couple of notes:

> The following instructions involve installation using **Docker** packages only, and this ensures obtaining the latest official release of **Docker**. If you need to install only using `Ubuntu-managed` packages, consult the Ubuntu documentation (Not recommended otherwise for obvious reasons).
>
> Ubuntu Utopic 14.10 and 15.04 exist in Docker's APT repository but are no longer officially supported due to known security issues.

Prerequisites

- Docker only works on a 64-bit installation of Linux.
- Docker requires Linux kernel version 3.10 or higher (Except for `Ubuntu Precise 12.04`, which requires version 3.13 or higher). Kernels older than 3.10 lack some of the features required to run Docker containers and contain known bugs which cause data loss and frequently panic under certain conditions. Check current kernel version with the command `uname -r`. Check this post if you need to update your `Ubuntu Precise (12.04 LTS)` kernel by scrolling further down. Refer to this WikiHow post to obtain the latest version for other Ubuntu installations.

Update APT sources

This needs to be done so as to access packages from Docker repository.

1. Log into your machine as a user with `sudo` or `root` privileges.
2. Open a terminal window.
3. Update package information, ensure that APT works with the https method, and that CA certificates are installed.

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

4. Add Docker’s official GPG key:

   ```
   $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
   ```

   Verify that the key fingerprint is **9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88** .

   ```
   $ sudo apt-key fingerprint 0EBFCD88
   ```

```
    pub   4096R/0EBFCD88 2017-02-22
          Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
    uid                  Docker Release (CE deb) <docker@docker.com>
    sub   4096R/F273FCD8 2017-02-22
```

5. Find the entry in the table below which corresponds to your Ubuntu version. This determines where APT will search for Docker packages. When possible, run a long-term support (LTS) edition of Ubuntu.

Ubuntu Version	Repository
Precise 12.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Wily 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

   Note: Docker does not provide packages for all architectures. Binary artifacts are built nightly, and you can download them from `https://master.dockerproject.org`. To install docker on a multi-architecture system, add an `[arch=...]` clause to the entry. Refer to Debian Multiarch wiki for details.

6. Run the following command, substituting the entry for your operating system for the placeholder `<REPO>`.

   $ echo "" | sudo tee /etc/apt/sources.list.d/docker.list

7. Update the `APT` package index by executing `sudo apt-get update`.

8. Verify that `APT` is pulling from the right repository.

When you run the following command, an entry is returned for each version of Docker that is available for you to install. Each entry should have the URL `https://apt.dockerproject.org/repo/`. The version currently installed is marked with `***`.See the below example's output.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
       1.12.0-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

From now on when you run `apt-get upgrade`, `APT` pulls from the new repository.

Prerequisites by Ubuntu Version

For Ubuntu Trusty (14.04) , Wily (15.10) , and Xenial (16.04) , install the `linux-image-extra-*` kernel packages, which allows you use the `aufs` storage driver.

To install the `linux-image-extra-*` packages:

1. Open a terminal on your Ubuntu host.
2. Update your package manager with the command `sudo apt-get update`.
3. Install the recommended packages.

   ```
   $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
   ```

4. Proceed to Docker installation

For Ubuntu Precise (12.04 LTS), Docker requires the 3.13 kernel version. If your kernel version is older than 3.13, you must upgrade it. Refer to this table to see which packages are required for your environment:

Package	Description
`linux-image-generic-lts-trusty`	Generic Linux kernel image. This kernel has `AUFS` built in. This is required to run Docker.
`linux-headers-generic-lts-trusty`	Allows packages such as `ZFS` and `VirtualBox guest additions` which depend on them. If you didn’t install the headers for your existing kernel, then you can skip these headers for the `trusty` kernel. If you’re unsure, you should include this package for safety.
`xserver-xorg-lts-trusty`	Optional in non-graphical environments without Unity/Xorg. **Required** when running Docker on machine with a graphical environment.

Package	Description
`ligbl1-mesa-glx-lts-trusty`	To learn more about the reasons for these packages, read the installation instructions for backported kernels, specifically the LTS Enablement Stack. Refer to note 5 under each version.

To upgrade your kernel and install the additional packages, do the following:

1. Open a terminal on your Ubuntu host.
2. Update your package manager with the command `sudo apt-get update`.
3. Install both the required and optional packages.

```
$ sudo apt-get install linux-image-generic-lts-trusty
```

4. Repeat this step for other packages you need to install.
5. Reboot your host to use the updated kernel using the command `sudo reboot`.
6. After reboot, go ahead and install Docker.

Install the latest version

Make sure you satisfy the prerequisites, only then follow the below steps.

> **Note:** For production systems, it is recommended that you install a specific version so that you do not accidentally update Docker. You should plan upgrades for production systems carefully.

1. Log into your Ubuntu installation as a user with `sudo` privileges. (Possibly running `sudo -su`).
2. Update your APT package index by running `sudo apt-get update`.
3. Install Docker Community Edition with the command `sudo apt-get install docker-ce`.
4. Start the `docker` daemon with the command `sudo service docker start`.
5. Verify that `docker` is installed correctly by running the hello-world image.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Manage Docker as a non-root user

If you don’t want to use `sudo` when you use the docker command, create a Unix group called `docker` and add users to it. When the `docker` daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.

To create the `docker` group and add your user:

1. Log into Ubuntu as a user with `sudo` privileges.
2. Create the `docker` group with the command `sudo groupadd docker`.
3. Add your user to the `docker` group.

   ```
   $ sudo usermod -aG docker $USER
   ```

4. Log out and log back in so that your group membership is re-evaluated.
5. Verify that you can `docker` commands without `sudo` permission.

   ```
   $ docker run hello-world
   ```

If this fails, you will see an error:

```
    Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Check whether the `DOCKER_HOST` environment variable is set for your shell.

```
    $ env | grep DOCKER_HOST
```

If it is set, the above command will return a result. If so, unset it.

```
    $ unset DOCKER_HOST
```

You may need to edit your environment in files such as `~/.bashrc` or `~/.profile` to prevent the `DOCKER_HOST` variable from being set erroneously.

Installing Docker on Ubuntu

Requirements: Docker can be installed on any Linux with a kernel of at least version 3.10. Docker is supported on the following 64-bit versions of Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Easy Installation

Note: Installing Docker from the default Ubuntu repository will install an old version of Docker.

To install the latest version of Docker using the Docker repository, use `curl` to grab and run the installation script provided by Docker:

```
$ curl -sSL https://get.docker.com/ | sh
```

Alternatively, `wget` can be used to install Docker:

```
$ wget -qO- https://get.docker.com/ | sh
```

Docker will now be installed.

Manual Installation

If, however, running the installation script is not an option, the following instructions can be used to manually install the latest version of Docker from the official repository.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

Add the GPG key:

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 \
  --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Next, open the `/etc/apt/sources.list.d/docker.list` file in your favorite editor. If the file doesn’t exist, create it. Remove any existing entries. Then, depending on your version, add the following line:

- Ubuntu Precise 12.04 (LTS):

  ```
  deb https://apt.dockerproject.org/repo ubuntu-precise main
  ```

- Ubuntu Trusty 14.04 (LTS)

  ```
  deb https://apt.dockerproject.org/repo ubuntu-trusty main
  ```

- Ubuntu Wily 15.10

  ```
  deb https://apt.dockerproject.org/repo ubuntu-wily main
  ```

- Ubuntu Xenial 16.04 (LTS)

  ```
  deb https://apt.dockerproject.org/repo ubuntu-xenial main
  ```

Save the file and exit, then update your package index, uninstall any installed versions of Docker, and verify `apt` is pulling from the correct repo:

```
$ sudo apt-get update
$ sudo apt-get purge lxc-docker
$ sudo apt-cache policy docker-engine
```

Depending on your version of Ubuntu, some prerequisites may be required:

- Ubuntu Xenial 16.04 (LTS), Ubuntu Wily 15.10, Ubuntu Trusty 14.04 (LTS) sudo apt-get update && sudo apt-get install linux-image-extra-$(uname -r)

- Ubuntu Precise 12.04 (LTS)

  This version of Ubuntu requires kernel version 3.13. You may need to install additional packages depending on your environment:

  `linux-image-generic-lts-trusty`

  Generic Linux kernel image. This kernel has AUFS built in. This is required to run Docker.

  `linux-headers-generic-lts-trusty`

  Allows packages such as ZFS and VirtualBox guest additions which depend on them. If you didn’t install the headers for your existing kernel, then you can skip these headers for the `trusty` kernel. If you’re unsure, you should include this package for safety.

  `xserver-xorg-lts-trusty`

  `libgl1-mesa-glx-lts-trusty`

  These two packages are optional in non-graphical environments without Unity/Xorg. Required when running Docker on machine with a graphical environment.

  To learn more about the reasons for these packages, read the installation instructions for backported kernels, specifically the LTS Enablement Stack — refer to note 5 under each version.

  Install the required packages then reboot the host:

  `$ sudo apt-get install linux-image-generic-lts-trusty`

  `$ sudo reboot`

Finally, update the `apt` package index and install Docker:

```
$ sudo apt-get update
$ sudo apt-get install docker-engine
```

Start the daemon:

```
$ sudo service docker start
```

Now verify that docker is running properly by starting up a test image:

```
$ sudo docker run hello-world
```

This command should print a welcome message verifying that the installation was successful.

Create a docker container in Google Cloud

You can use docker, without using docker daemon (engine), by using cloud providers. In this example, you should have a `gcloud` (Google Cloud util), that connected to your account

```
docker-machine create --driver google --google-project `your-project-name` google-machine-type
f1-large fm02
```

This example will create a new instance, in your Google Cloud console. Using machine time `f1-large`

Install Docker on Ubuntu

Docker is supported on the following *64-bit* versions of Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

A couple of notes:

> The following instructions involve installation using **Docker** packages only, and this ensures obtaining the latest official release of **Docker**. If you need to install only using `Ubuntu-managed` packages, consult the Ubuntu documentation (Not recommended otherwise for obvious reasons).
>
> Ubuntu Utopic 14.10 and 15.04 exist in Docker's APT repository but are no longer officially supported due to known security issues.

Prerequisites

- Docker only works on a 64-bit installation of Linux.
- Docker requires Linux kernel version 3.10 or higher (Except for `Ubuntu Precise 12.04`, which requires version 3.13 or higher). Kernels older than 3.10 lack some of the features required to run Docker containers and contain known bugs which cause data loss and frequently panic under certain conditions. Check current kernel version with the command `uname -r`. Check this post if you need to update your `Ubuntu Precise (12.04 LTS)` kernel by scrolling further down. Refer to this WikiHow post to obtain the latest version for other Ubuntu installations.

Update APT sources

This needs to be done so as to access packages from Docker repository.

1. Log into your machine as a user with `sudo` or `root` privileges.
2. Open a terminal window.
3. Update package information, ensure that APT works with the https method, and that CA certificates are installed.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

4. Add the new `GPG` key. This commands downloads the key with the ID `58118E89F3A912897C070ADBF76221572C52609D` from the keyserver `hkp://ha.pool.sks-keyservers.net:80` and adds it to the `adv keychain`. For more information, see the output of `man apt-key`.

   ```
   $ sudo apt-key adv \
          --keyserver hkp://ha.pool.sks-keyservers.net:80 \
          --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
   ```

5. Find the entry in the table below which corresponds to your Ubuntu version. This determines where APT will search for Docker packages. When possible, run a long-term support (LTS) edition of Ubuntu.

Ubuntu Version	Repository
Precise 12.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Wily 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

   Note: Docker does not provide packages for all architectures. Binary artifacts are built nightly, and you can download them from `https://master.dockerproject.org`. To install docker on a multi-architecture system, add an `[arch=...]` clause to the entry. Refer to Debian Multiarch wiki for details.

6. Run the following command, substituting the entry for your operating system for the placeholder `<REPO>`.

   $ echo "" | sudo tee /etc/apt/sources.list.d/docker.list

7. Update the `APT` package index by executing `sudo apt-get update`.

8. Verify that `APT` is pulling from the right repository.

When you run the following command, an entry is returned for each version of Docker that is available for you to install. Each entry should have the URL `https://apt.dockerproject.org/repo/`. The version currently installed is marked with `***`.See the below example's output.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
```

```
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
     1.12.0-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

From now on when you run `apt-get upgrade`, `APT` pulls from the new repository.

Prerequisites by Ubuntu Version

For Ubuntu Trusty (14.04) , Wily (15.10) , and Xenial (16.04) , install the `linux-image-extra-*` kernel packages, which allows you use the `aufs` storage driver.

To install the `linux-image-extra-*` packages:

1. Open a terminal on your Ubuntu host.
2. Update your package manager with the command `sudo apt-get update`.
3. Install the recommended packages.

   ```
   $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
   ```

4. Proceed to Docker installation

For Ubuntu Precise (12.04 LTS), Docker requires the 3.13 kernel version. If your kernel version is older than 3.13, you must upgrade it. Refer to this table to see which packages are required for your environment:

Package	Description
`linux-image-generic-lts-trusty`	Generic Linux kernel image. This kernel has `AUFS` built in. This is required to run Docker.
`linux-headers-generic-lts-trusty`	Allows packages such as `ZFS` and `VirtualBox guest additions` which depend on them. If you didn't install the headers for your existing kernel, then you can skip these headers for the `trusty` kernel. If you're unsure, you should include this package for safety.
`xserver-xorg-lts-trusty`	Optional in non-graphical environments without Unity/Xorg. **Required** when running Docker on machine with a graphical environment.
`ligbl1-mesa-glx-lts-trusty`	To learn more about the reasons for these packages, read the installation instructions for backported kernels, specifically the LTS Enablement Stack. Refer to note 5 under each version.

To upgrade your kernel and install the additional packages, do the following:

1. Open a terminal on your Ubuntu host.
2. Update your package manager with the command `sudo apt-get update`.

3. Install both the required and optional packages.

   ```
   $ sudo apt-get install linux-image-generic-lts-trusty
   ```

4. Repeat this step for other packages you need to install.

5. Reboot your host to use the updated kernel using the command `sudo reboot`.

6. After reboot, go ahead and install Docker.

Install the latest version

Make sure you satisfy the prerequisites, only then follow the below steps.

> **Note:** For production systems, it is recommended that you install a specific version so that you do not accidentally update Docker. You should plan upgrades for production systems carefully.

1. Log into your Ubuntu installation as a user with `sudo` privileges. (Possibly running `sudo -su`).

2. Update your APT package index by running `sudo apt-get update`.

3. Install Docker with the command `sudo apt-get install docker-engine`.

4. Start the `docker` daemon with the command `sudo service docker start`.

5. Verify that `docker` is installed correctly by running the hello-world image.

   ```
   $ sudo docker run hello-world
   ```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Manage Docker as a non-root user

If you don’t want to use `sudo` when you use the docker command, create a Unix group called `docker` and add users to it. When the `docker` daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.

To create the `docker` group and add your user:

1. Log into Ubuntu as a user with `sudo` privileges.

2. Create the `docker` group with the command `sudo groupadd docker`.

3. Add your user to the `docker` group.

   ```
   $ sudo usermod -aG docker $USER
   ```

4. Log out and log back in so that your group membership is re-evaluated.

5. Verify that you can `docker` commands without `sudo` permission.

   ```
   $ docker run hello-world
   ```

If this fails, you will see an error:

```
    Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Check whether the `DOCKER_HOST` environment variable is set for your shell.

```
   $ env | grep DOCKER_HOST
```

If it is set, the above command will return a result. If so, unset it.

```
   $ unset DOCKER_HOST
```

You may need to edit your environment in files such as `~/.bashrc` or `~/.profile` to prevent the `DOCKER_HOST` variable from being set erroneously.

Installating Docker-ce OR Docker-ee on CentOS

Docker has announced following editions:

-Docker-ee (Enterprise Edition) along with Docker-ce(Community Edition) and Docker (Commercial Support)

This document will help you with installation steps of Docker-ee and Docker-ce edition in CentOS

Docker-ce Installation

Following are steps to install docker-ce edition

1. Install yum-utils, which provides yum-config-manager utility:

   ```
   $ sudo yum install -y yum-utils
   ```

2. Use the following command to set up the stable repository:

   ```
   $ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
   ```

3. Optional: Enable the edge repository. This repository is included in the docker.repo file above but is disabled by default. You can enable it alongside the stable repository.

   ```
    $ sudo yum-config-manager --enable docker-ce-edge
   ```

- You can disable the edge repository by running the `yum-config-manager` command with the `--disable` flag. To re-enable it, use the `--enable` flag. The following command disables the edge repository.

  ```
  $ sudo yum-config-manager --disable docker-ce-edge
  ```

4. Update the yum package index.

   ```
   $ sudo yum makecache fast
   ```

5. Install the docker-ce using following command:

   ```
   $ sudo yum install docker-ce-17.03.0.ce
   ```

6. Confirm the Docker-ce fingerprint

   ```
   060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35
   ```

   If you want to install some other version of docker-ce you can use following command:

   ```
   $ sudo yum install docker-ce-VERSION
   ```

   Specify the `VERSION` number

7. If everything went well the docker-ce is now installed in your system, use following command to start:

   ```
   $ sudo systemctl start docker
   ```

8. Test your docker installation:

   ```
   $ sudo docker run hello-world
   ```

   you should get following message:

   ```
   Hello from Docker!
   This message shows that your installation appears to be working correctly.
   ```

-Docker-ee (Enterprise Edition) Installation

For Enterprise Edition (EE) it would be required to signup, to get your <DOCKER-EE-URL>.

1. To signup go to https://cloud.docker.com/. Enter your details and confirm your email id. After confirmation you would be given a <DOCKER-EE-URL>, which you can see in your dashboard after clicking on setup.

2. Remove any existing Docker repositories from `/etc/yum.repos.d/`

3. Store your Docker EE repository URL in a yum variable in `/etc/yum/vars/`. Replace <DOCKER-EE-URL> with the URL you noted down in the first step.

```
$ sudo sh -c 'echo "<DOCKER-EE-URL>" > /etc/yum/vars/dockerurl'
```

4. Install yum-utils, which provides the yum-config-manager utility:

```
$ sudo yum install -y yum-utils
```

5. Use the following command to add the stable repository:

```
$ sudo yum-config-manager \
--add-repo \
<DOCKER-EE-URL>/docker-ee.repo
```

6. Update the yum package index.

```
$ sudo yum makecache fast
```

7. Install docker-ee

```
sudo yum install docker-ee
```

8. You can start the docker-ee using following command:

```
$ sudo systemctl start docker
```

1. ReactJS — Introduction

ReactJS is a simple, feature rich, component based JavaScript UI library. It can be used to develop small applications as well as big, complex applications. ReactJS provides minimal and solid feature set to kick-start a web application. React community compliments React library by providing large set of ready-made components to develop web application in a record time. React community also provides advanced concept like state management, routing, etc., on top of the React library.

React versions

The initial version, *0.3.0* of React is released on May, 2013 and the latest version, *17.0.1* is released on October, 2020. The major version introduces breaking changes and the minor version introduces new feature without breaking the existing functionality. Bug fixes are released as and when necessary. React follows the *Sematic Versioning (semver)* principle.

Features

The salient features of *React library* are as follows:

- Solid base architecture
- Extensible architecture
- Component based library
- JSX based design architecture
- Declarative UI library

Benefits

Few benefits of using *React library* are as follows:

- Easy to learn
- Easy to adept in modern as well as legacy application
- Faster way to code a functionality
- Availability of large number of ready-made component
- Large and active community

Applications

Few popular websites powered by *React library* are listed below:

- *Facebook*, popular social media application
- *Instagram*, popular photo sharing application
- *Netflix*, popular media streaming application

- *Code Academy*, popular online training application
- *Reddit*, popular content sharing application

As you see, most popular application in every field is being developed by *React Library*.

2. ReactJS — Installation

This chapter explains the installation of React library and its related tools in your machine. Before moving to the installation, let us verify the prerequisite first.

React provides CLI tools for the developer to fast forward the creation, development and deployment of the React based web application. React CLI tools depends on the *Node.js* and must be installed in your system. Hopefully, you have installed Node.js on your machine. We can check it using the below command:

```
node --version
```

You could see the version of *Nodejs* you might have installed. It is shown as below for me,

```
v14.2.0
```

If *Nodejs* is not installed, you can download and install by visiting https://nodejs.org/en/download/.

Toolchain

To develop lightweight features such as form validation, model dialog, etc., React library can be directly included into the web application through content delivery network (CDN). It is similar to using jQuery library in a web application. For moderate to big application, it is advised to write the application as multiple files and then use bundler such as webpack, parcel, rollup, etc., to compile and bundle the application before deploying the code.

React toolchain helps to create, build, run and deploy the React application. React toolchain basically provides a starter project template with all necessary code to bootstrap the application.

Some of the popular toolchain to develop React applications are:

- Create React App - SPA oriented toolchain
- Next.js - server-side rendering oriented toolchain
- Gatsby - Static content oriented toolchain

Tools required to develop a React application are:

- The *serve*, a static server to serve our application during development
- Babel compiler
- Create React App CLI

Let us learn the basics of the above mentioned tools and how to install those in this chapter.

The *serve* static server

The *serve* is a lightweight web server. It serves static site and single page application. It loads fast and consume minimum memory. It can be used to serve a React application. Let us install the tool using *npm* package manager in our system.

```
npm install serve -g
```

Let us create a simple static site and serve the application using *serve* app.

Open a command prompt and go to your workspace.

```
cd /go/to/your/workspace
```

Create a new folder, *static_site* and change directory to newly created folder.

```
mkdir static_site
cd static_site
```

Next, create a simple webpage inside the folder using your favorite html editor.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Static website</title>
    </head>
    <body>
        <div><h1>Hello!</h1></div>
    </body>
</html>
```

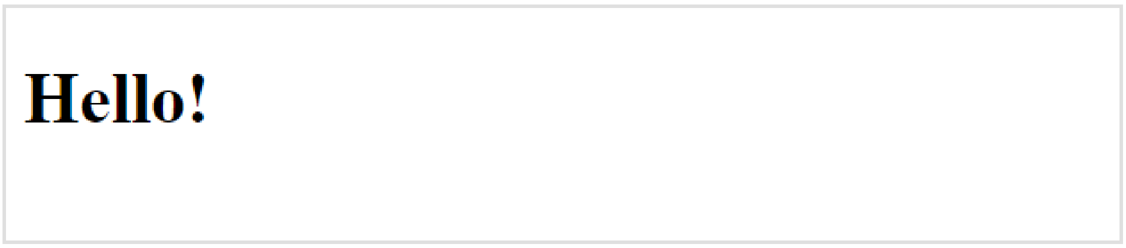
Next, run the *serve* command.

```
serve .
```

We can also serve single file, *index.html* instead of the whole folder.

```
serve ./index.html
```

Next, open the browser and enter *http://localhost:5000* in the address bar and press enter. *serve* application will serve our webpage as shown below.


Hello!

The *serve* will serve the application using default port, 5000. If it is not available, it will pick up a random port and specify it.

```
│   Serving!                                      │
    │                                                   │
    │   - Local:            http://localhost:57311      │
    │   - On Your Network:  http://192.168.56.1:57311   │
    │                                                   │
    │   This port was picked because 5000 is in use.    │
    │                                                   │
    │   Copied local address to clipboard!
```

Babel compiler

Babel is a JavaScript compiler which compiles many variant (es2015, es6, etc.,) of JavaScript into standard JavaScript code supported by all browsers. React uses JSX, an extension of JavaScript to design the user interface code. Babel is used to compile the *JSX* code into JavaScript code.

To install *Babel* and it's React companion, run the below command:

```
npm install babel-cli@6 babel-preset-react-app@3 -g

...
...

+ babel-cli@6.26.0
+ babel-preset-react-app@3.1.2
updated 2 packages in 8.685s
```

Babel helps us to write our application in next generation of advanced JavaScript syntax.

Create React App toolchain

Create React App is a modern CLI tool to create single page React application. It is the standard tool supported by React community. It handles babel compiler as well. Let us install *Create React App* in our local system.

```
> npm install -g create-react-app

+ create-react-app@4.0.1
added 6 packages from 4 contributors, removed 37 packages and updated 12
packages in 4.693s
```

Updating the toolchain

React Create App toolchain uses the *react-scripts* package to build and run the application. Once we started working on the application, we can update the react-script to the latest version at any time using *npm* package manager.

```
npm install react-scripts@latest
```

Advantages of using React toolchain

React toolchain provides lot of features out of the box. Some of the advantages of using React toolchain are:

- Predefined and standard structure of the application.
- Ready-made project template for different type of application.
- Development web server is included.
- Easy way to include third party React components.
- Default setup to test the application.

3. ReactJS — Architecture

React library is built on a solid foundation. It is simple, flexible and extensible. As we learned earlier, React is a library to create user interface in a web application. React's primary purpose is to enable the developer to create user interface using pure JavaScript. Normally, every user interface library introduces a new template language (which we need to learn) to design the user interface and provides an option to write logic, either inside the template or separately.

Instead of introducing new template language, React introduces three simple concepts as given below:

React elements

JavaScript representation of HTML DOM. React provides an API, ***React.createElement*** to create *React Element*.

JSX

A JavaScript extension to design user interface. JSX is an XML based, extensible language supporting HTML syntax with little modification. JSX can be compiled to *React Elements* and used to create user interface.

React component

React component is the primary building block of the React application. It uses *React elements* and *JSX* to design its user interface. React component is basically a JavaScript class (extends the ***React.component*** class) or pure JavaScript function. React component has properties, state management, life cycle and event handler. React component can be able to do simple as well as advanced logic.

Let us learn more about components in the *React Component* chapter.

Workflow of a React application

Let us understand the workflow of a React application in this chapter by creating and analyzing a simple React application.

Open a command prompt and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *static_site* and change directory to newly created folder.

```
mkdir static_site
cd static_site
```

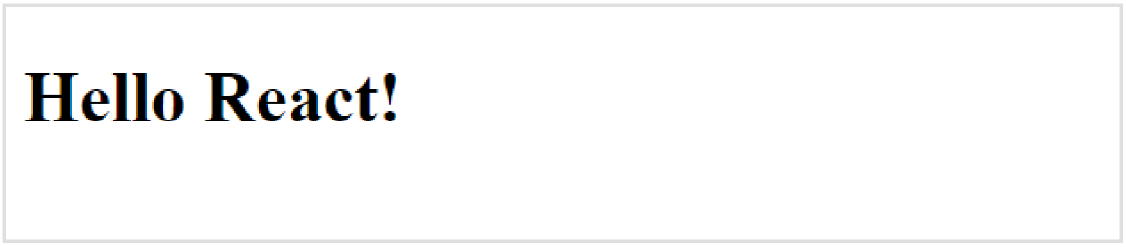
Next, create a file, *hello.html* and write a simple React application.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>React Application</title>
    </head>
    <body>
        <div id="react-app"></div>

        <script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>
        <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js" crossorigin></script>
        <script language="JavaScript">
            element = React.createElement('h1', {}, 'Hello React!')
            ReactDOM.render(element, document.getElementById('react-app'));
        </script>
    </body>
</html>
```

Next, serve the application using serve web server.

```
serve ./hello.html
```

Next, open your favorite browser. Enter *http://localhost:5000* in the address bar and then press enter.



Hello React!

Let us analyse the code and do little modification to better understand the React application.

Here, we are using two API provided by the React library.

React.createElement

Used to create React elements. It expects three parameters:

- Element tag
- Element attributes as object
- Element content - It can contain nested React element as well

ReactDOM.render

Used to render the element into the container. It expects two parameters:

- React Element OR JSX
- Root element of the webpage

Nested React element

As ***React.createElement*** allows nested React element, let us add nested element as shown below:

```
<script language="JavaScript">
    element = React.createElement('div', {},
        React.createElement('h1', {}, 'Hello React!'));
    ReactDOM.render(element, document.getElementById('react-app'));
</script>
```

It will generate the below content:

```
<div><h1>Hello React!</h1></div>
```

Use JSX

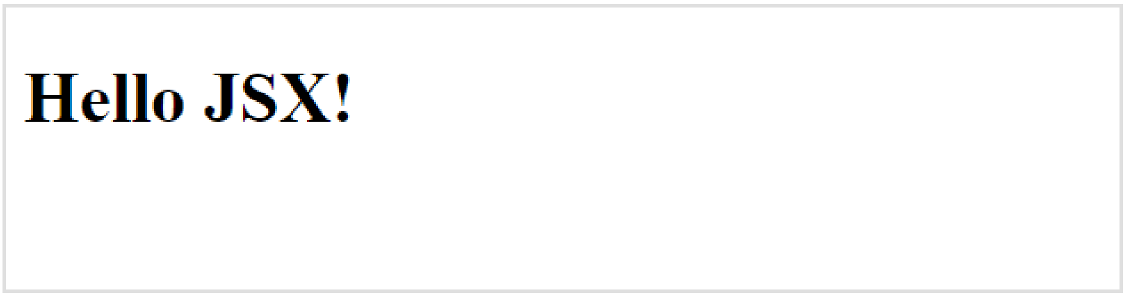
Next, let us remove the React element entirely and introduce JSX syntax as shown below:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>React Application</title>
    </head>
    <body>
        <div id="react-app"></div>

        <script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>
        <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js" crossorigin></script>
        <script
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
        <script type="text/babel">
            ReactDOM.render(
                <div><h1>Hello React!</h1></div>,
                document.getElementById('react-app') );
        </script>
    </body>
</html>
```

Here, we have included babel to convert JSX into JavaScript and added *type="text/babel"* in the script tag.

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script type="text/babel">
...
...
</script>
```

Next, run the application and open the browser. The output of the application is as follows:

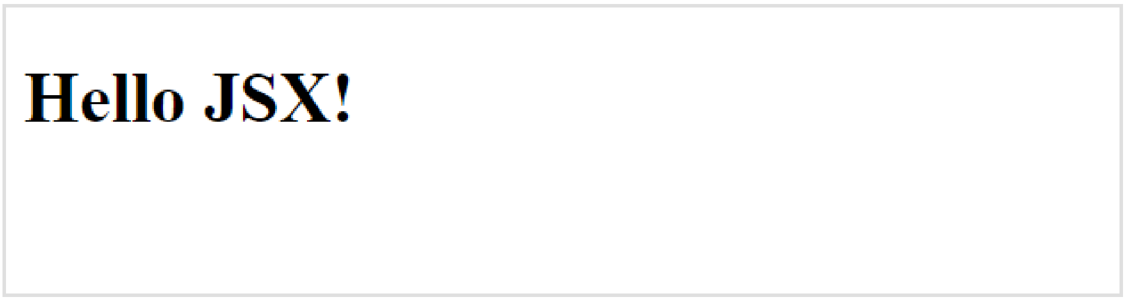


Hello JSX!

Next, let us create a new React component, *Greeting* and then try to use it in the webpage.

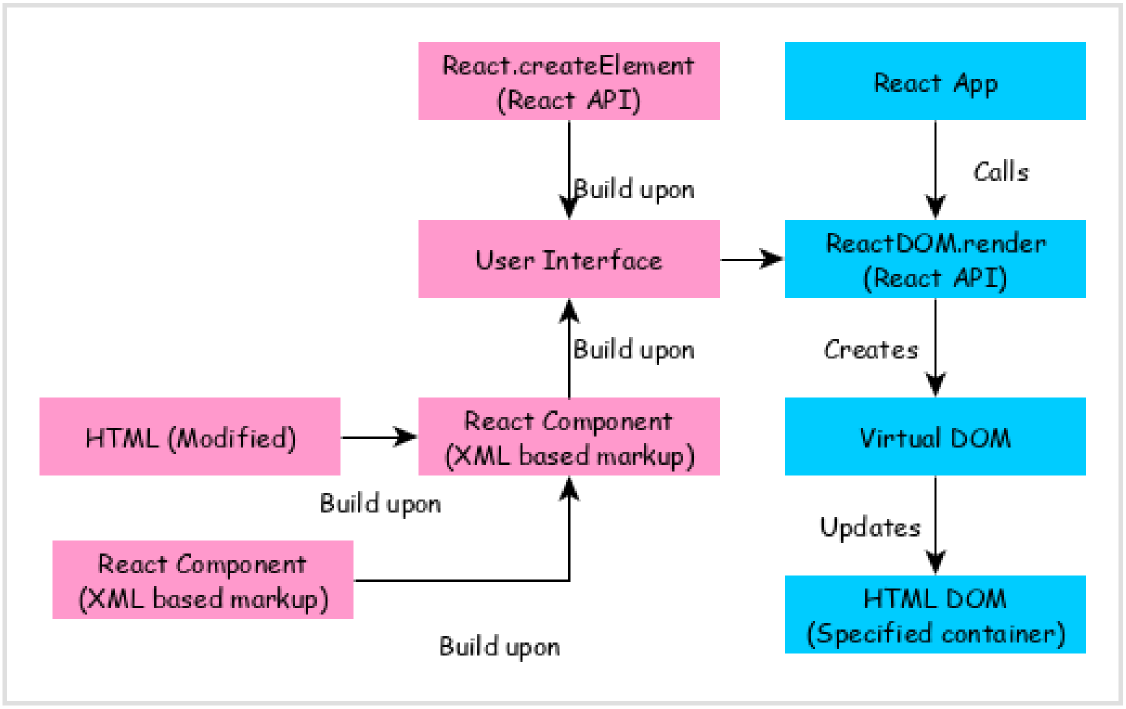
```
<script type="text/babel">
    function Greeting() {
        return <div><h1>Hello JSX!</h1></div>
    }
    ReactDOM.render(
        <Greeting />,
        document.getElementById('react-app') );
</script>
```

The result is same and as shown below:



Hello JSX!

By analyzing the application, we can visualize the workflow of the React application as shown in the below diagram.


React.createElement (React API)
React App
Build upon
Calls
User Interface
ReactDOM.render (React API)
Build upon
Creates
HTML (Modified)
React Component (XML based markup)
Virtual DOM
Build upon
Updates
React Component (XML based markup)
HTML DOM (Specified container)
Build upon

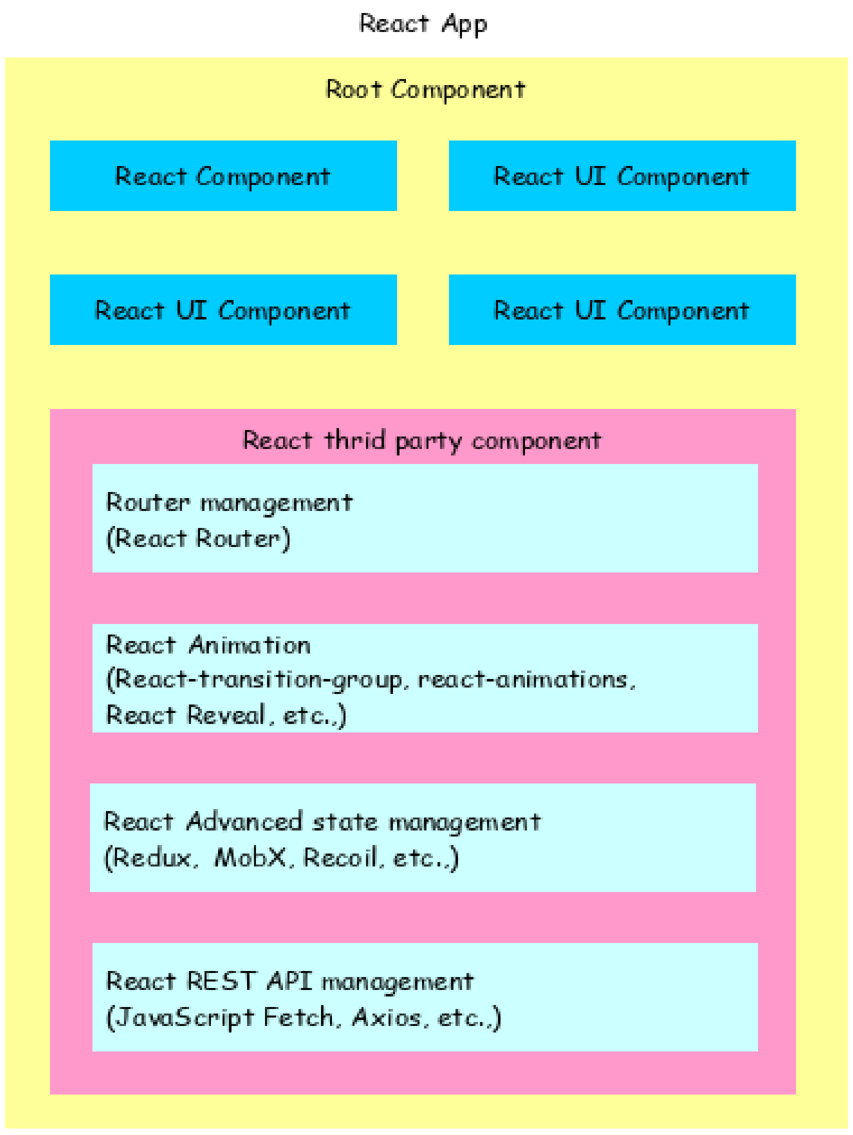
React app calls ***ReactDOM.render*** method by passing the user interface created using React component (coded in either JSX or React element format) and the container to render the user interface.

ReactDOM.render processes the JSX or React element and emits Virtual DOM.

Virtual DOM will be merged and rendered into the container.

Architecture of the React Application

React library is just UI library and it does not enforce any particular pattern to write a complex application. Developers are free to choose the design pattern of their choice. React community advocates certain design pattern. One of the patterns is *Flux* pattern. React library also provides lot of concepts like Higher Order component, Context, Render props, Refs etc., to write better code. React Hooks is evolving concept to do state management in big projects. Let us try to understand the high level architecture of a React application.


React App
Root Component
React Component
React UI Component
React UI Component
React UI Component
React thrid party component
Router management
(React Router)
React Animation
(React-transition-group, react-animations,
React Reveal, etc.)
React Advanced state management
(Redux, MobX, Recoil, etc.)
React REST API management
(JavaScript Fetch, Axios, etc.)

- React app starts with a single root component.
- Root component is build using one or more component.
- Each component can be nested with other component to any level.
- Composition is one of the core concepts of React library. So, each component is build by composing smaller components instead of inheriting one component from another component.
- Most of the components are user interface components.
- React app can include third party component for specific purpose such as routing, animation, state management, etc.

4. React — Creating a React Application

As we learned earlier, React library can be used in both simple and complex application. Simple application normally includes the React library in its script section. In complex application, developers have to split the code into multiple files and organize the code into a standard structure. Here, React toolchain provides pre-defined structure to bootstrap the application. Also, developers are free to use their own project structure to organize the code.

Let us see how to create simple as well as complex React application:

- Simple application using CDN
- Complex application using *React Create App* cli
- Complex application using customized method

Using CDN

Let us learn how to use content delivery network to include React in a simple web page.

Open a terminal and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *static_site* and change directory to newly created folder.

```
mkdir static_site
cd static_site
```

Next, create a new HTML file, *hello.html.*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Simple React app</title>
    </head>
    <body>
    </body>
</html>
```

Next, include *React library.*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Simple React app</title>
    </head>
    <body>
```

```
        <script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>
        <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js" crossorigin></script>
    </body>
</html>
```

Here,

- We are using ***unpkg*** CDN. ***unpkg*** is an open source, global content delivery network supporting ***npm*** packages.
- ***@17*** represent the version of the *React library*
- This is the development version of the *React library* with debugging option. To deploy the application in the production environment, use below scripts.

```
<script src="https://unpkg.com/react@17/umd/react.production.min.js"
crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js"
crossorigin></script>
```

Now, we are ready to use *React library* in our webpage.

Next, introduce a ***div*** tag with id ***react-app***.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>React based application</title>
    </head>
    <body>
        <div id="react-app"></div>

        <script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>
        <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js" crossorigin></script>
    </body>
</html>
```

The ***react-app*** is a placeholder container and React will work inside the container. We can use any name for the placeholder container relevant to our application.

Next, create a script section at the end of the document and use React feature to create an element.

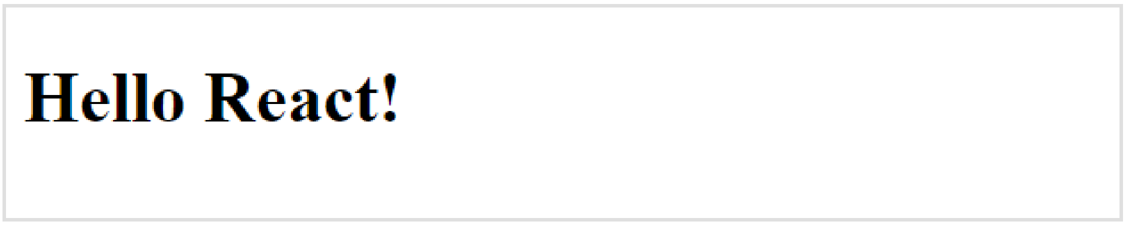
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>React based application</title>
    </head>
    <body>
        <div id="react-app"></div>

        <script src="https://unpkg.com/react@17/umd/react.development.js" 
crossorigin></script>
        <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js" crossorigin></script>
        <script language="JavaScript">
            element = React.createElement('h1', {}, 'Hello React!')
            ReactDOM.render(element, document.getElementById('react-app'));
        </script>
    </body>
</html>
```

Here, the application uses *React.createElement* and *ReactDOM.render* methods provided by *React Library* to dynamically create a HTML element and place it inside the **react-app** section.

Next, serve the application using *serve* web server.

```
serve ./hello.html
```

Next, open the browser and enter *http://localhost:5000* in the address bar and press enter. *serve* application will serve our webpage as shown below.


Hello React!

We can use the same steps to use React in the existing website as well. This method is very easy to use and consume React library. It can be used to do simple to moderate feature in a website. It can be used in new as well as existing application along with other libraries. This method is suitable for static website with few dynamic section like contact form, simple payment option, etc., To create advanced single page application (SPA), we need to use React tools. Let us learn how to create a SPA using React tools in upcoming chapter.

Using Create React App tool

Let us learn to create an expense management application using *Create React App* tool.

Open a terminal and go to your workspace.

```
> cd /go/to/your/workspace
```

Next, create a new React application using *Create React App* tool.

```
> create-react-app expense-manager
```

It will a create new folder ***expense-manager*** with startup template code.

Next, go to ***expense-manager*** folder and install the necessary library.

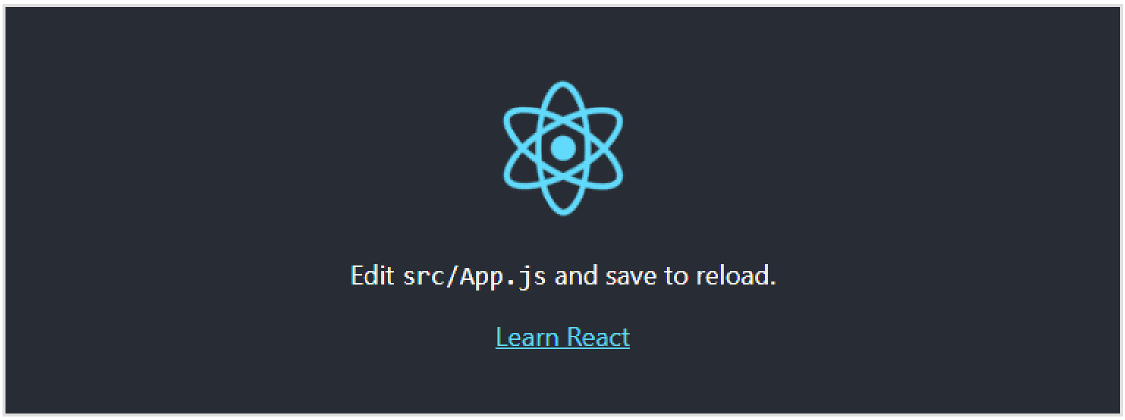
```
cd expense-manager
npm install
```

The *npm install* will install the necessary library under *node_modules* folder.

Next, start the application.

```
npm start
Compiled successfully!

You can now view react-cra-web-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Next, open the browser and enter *http://localhost:3000* in the address bar and press enter. The development web server will serve our webpage as shown below.


Edit src/App.js and save to reload.

Learn React

Let us analyse the structure of our React application.

Files and folders

The content of the React application is as follows:

```
|-- README.md
|-- node_modules
|-- package-lock.json
|-- package.json
|-- public
|    |-- favicon.ico
|    |-- index.html
|    |-- logo192.png
|    |-- logo512.png
|    |-- manifest.json
|    `-- robots.txt
`-- src
```

```
    |-- App.css
    |-- App.js
    |-- App.test.js
    |-- index.css
    |-- index.js
    |-- logo.svg
    |-- reportWebVitals.js
    `-- setupTests.js
```

Here,

The *package.json* is the core file representing the project. It configures the entire project and consists of project name, project dependencies, and commands to build and run the application.

```
{
  "name": "expense-manager",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.6",
    "@testing-library/react": "^11.2.2",
    "@testing-library/user-event": "^12.6.0",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-scripts": "4.0.1",
    "web-vitals": "^0.2.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

The *package.json* refers the below React library in its dependency section.

- *react* and *react-dom* are core react libraries used to develop web application.
- *web-vitals* are general library to support application in different browser.
- *react-scripts* are core react scripts used to build and run application.
- *@testing-library/jest-dom*, *@testing-library/react* and *@testing-library/user-event* are testing libary used to test the application after development.

- The ***public*** **folder** - Contains the core file, *index.html* and other web resources like images, logos, robots, etc., *index.html* loads our react application and render it in user's browser.
- The *src* folder - Contains the actual code of the application. We will check it next section.

Source code of the application

Let us check the each and every source code document of the application in this chapter.

- The *index.js* - Entry point of our application. It uses *ReactDOM.render* method to kick-start and start the application. The code is as follows:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Here,

React.StrictMode is a build-in component used to prevent unexpected bugs by analysing the component for unsafe lifecycle, unsafe API usage, depreciated API usage, etc., and throwing the relevant warning.

- *App* is our first custom and root component of the application. All other components will be rendered inside the *App* component.

The *index.css* - Used to styles of the entire application. Let us remove all styles and start with fresh code.

App.js - Root component of our application. Let us replace the existing JSX and show simple hello react message as shown below:

```
import './App.css';

function App() {
  return (
    <h1>Hello React!</h1>
  );
}

export default App;
```

- ***App.css* -** Used to style the *App* component. Let us remove all styles and start with fresh code.
- ***App.test.js* -** Used to write unit test function for our component.
- ***setupTests.js* -** Used to setup the testing framework for our application.
- ***reportWebVitals.js* -** Generic web application startup code to support all browsers.
- ***logo.svg* -** Logo in SVG format and can be loaded into our application using *import* keyword. Let us remove it from the project.

Customize the code

Let us remove the default source code of the application and bootstrap the application to better understand the internals of React application.

Delete all files under *src* and *public* folder.

Next, create a folder, *components* under *src* to include our React components. The idea is to create two files, *<component>.js* to write the component logic and *<component.css>* to include the component specific styles.

The final structure of the application will be as follows:

```
|-- package-lock.json
|-- package.json
`-- public
    |-- index.html
`-- src
    |-- index.js
    `-- components
    |   |-- mycom.js
    |   |-- mycom.css
```

Let us create a new component, HelloWorld to confirm our setup is working fine. Create a file, HelloWorld.js under components folder and write a simple component to emit Hello World message.

```
import React from "react";

class HelloWorld extends React.Component {
  render() {
    return (
      <div>
```

```
            <h1>Hello World!</h1>
         </div>
      );
   }
}
export default HelloWorld;
```

Next, create our main file, *index.js* under *src* folder and call our newly created component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import HelloWorld from './components/HelloWorld';

ReactDOM.render(
  <React.StrictMode>
    <HelloWorld />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Next, create a html file, *index.html* (under *public* folder*), which will be our entry point of the application.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Expense Manager</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Run the application

Let us run the application by invoking the *start* script configured in *package.json* file.

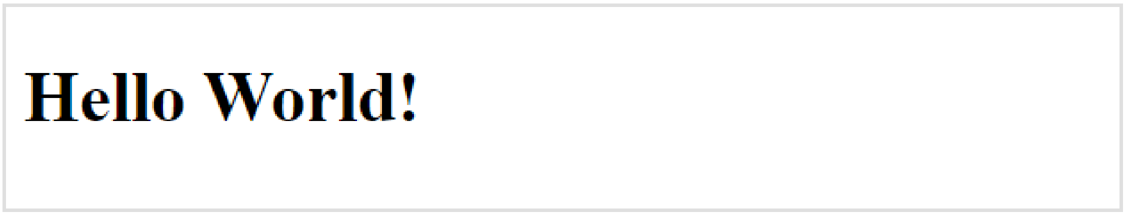
```
> npm start
```

It will start the application in the local system and can be accessed through browser @ http://localhost:3000/.

```
> expense-manager@0.1.0 start D:\path\to\expense-manager
> react-scripts start

i ｢wds｣: Project is running at http://192.168.56.1/
i ｢wds｣: webpack output is served from
i ｢wds｣: Content not from webpack is served from D:\path\to\expense-
manager\public
i ｢wds｣: 404s will fallback to /
```

```
Starting the development server...
Compiled successfully!

You can now view expense-manager in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Open your favorite browser and go to http://localhost:3000. The result of the application is as shown below:



Hello World!

Using custom solution

As we learned earlier, *Create react app* is the recommended tool to kick-start the React application. It includes everything to develop React application. But sometimes, application does not need all the feature provided by *Crzzeate React App* and we want our application to be small and tidy. Then, we can use our own customized solution to create React application with just enough dependency to support our application.

To create a custom project, we need to have basic knowledge about four items.

- **Package manager -** High level management of application. We are using *npm* as our default package manager.

- **Compiler -** Compiles the JavaScript variants into standard JavaScript supported by browser. We are using *Babel* as our default compiler.

- **Bundler -** Bundles the multiple sources (JavaScript, html and css) into a single deployable code. *Create React App* uses webpack as its bundler. Let us learn how to use *Rollup* and *Parcel* bundler in the upcoming section.

- **Webserver -** Starts the development server and launch our application. *Create React App* uses an internal webserver and we can use *serve* as our development server.

Using Rollup bundler

Rollup is one of the small and fast JavaScript bundlers. Let us learn how to use rollup bundler in this chapter.

Open a terminal and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *expense-manager-rollup* and move to newly created folder. Also, open the folder in your favorite editor or IDE.

```
mkdir expense-manager-rollup
cd expense-manager-rollup
```

Next, create and initialize the project.

```
npm init -y
```

Next, install React libraries (*react* and *react-dom*).

```
npm install react@^17.0.0 react-dom@^17.0.0 --save
```

Next, install babel and its preset libraries as development dependency.

```
npm install @babel/preset-env @babel/preset-react @babel/core @babel/plugin-
proposal-class-properties -D
```

Next, install rollup and its plugin libraries as development dependency.

```
npm i -D rollup postcss@8.1 @rollup/plugin-babel @rollup/plugin-commonjs
@rollup/plugin-node-resolve @rollup/plugin-replace rollup-plugin-livereload
rollup-plugin-postcss rollup-plugin-serve postcss@8.1 postcss-modules@4 rollup-
plugin-postcss
```

Next, install corejs and regenerator runtime for async programming.

```
npm i regenerator-runtime core-js
```

Next, create a babel configuration file, *.babelrc* under the root folder to configure the babel compiler.

```
{
    "presets": [
        [
            "@babel/preset-env",
            {
                "useBuiltIns": "usage",
                "corejs": 3,
                "targets": "> 0.25%, not dead"
            }
        ],
        "@babel/preset-react"
    ],
    "plugins": [
        "@babel/plugin-proposal-class-properties"
    ]
}
```

Next, create a *rollup.config.js* file in the root folder to configure the rollup bundler.

```
import babel from '@rollup/plugin-babel';
import resolve from '@rollup/plugin-node-resolve';
import commonjs from '@rollup/plugin-commonjs';
import replace from '@rollup/plugin-replace';

import serve from 'rollup-plugin-serve';
import livereload from 'rollup-plugin-livereload';

import postcss from 'rollup-plugin-postcss'

export default {
    input: 'src/index.js',
    output: {
        file: 'public/index.js',
        format: 'iife',
    },
    plugins: [
        commonjs({
            include: [
                'node_modules/**',
            ],
            exclude: [
                'node_modules/process-es6/**',
            ],
        }),
        resolve(),
        babel({
            exclude: 'node_modules/**'
        }),
        replace({
            'process.env.NODE_ENV': JSON.stringify('production'),
        }),
        postcss({
            autoModules: true
        }),
        livereload('public'),
        serve({
            contentBase: 'public',
            port: 3000,
            open: true,
        }), // index.html should be in root of project
    ]
}
```

Next, update the *package.json* and include our entry point (*public/index.js* and *public/styles.css*) and command to build and run the application.

```
...
"main": "public/index.js",
"style": "public/styles.css",
"files": [
  "public"
],
"scripts": {
  "start": "rollup -c -w",
  "build": "rollup"
},
...
```

Next, create a *src* folder in the root directory of the application, which will hold all the source code of the application.

Next, create a folder, *components* under *src* to include our React components. The idea is to create two files, *<component>.js* to write the component logic and *<component.css>* to include the component specific styles.

The final structure of the application will be as follows:

```
|-- package-lock.json
|-- package.json
|-- rollup.config.js
|-- .babelrc
`-- public
    |-- index.html
`-- src
    |-- index.js
    `-- components
    |   |-- mycom.js
    |   |-- mycom.css
```

Let us create a new component, *HelloWorld* to confirm our setup is working fine. Create a file, *HelloWorld.js* under components folder and write a simple component to emit *Hello World* message.

```
import React from "react";

class HelloWorld extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello World!</h1>
      </div>
    );
  }
}
export default HelloWorld;
```

Next, create our main file, *index.js* under *src* folder and call our newly created component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import HelloWorld from './components/HelloWorld';

ReactDOM.render(
  <React.StrictMode>
    <HelloWorld />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Next, create a *public* folder in the root directory.

Next, create a html file, *index.html* (under *public* folder*), which will be our entry point of the application.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Expense Manager :: Rollup version</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/JavaScript" src="./index.js"></script>
  </body>
</html>
```

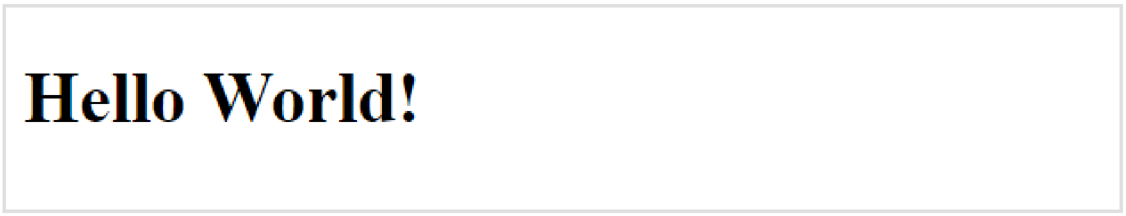
Next, build and run the application.

```
npm start
```

The *npm* build command will execute the *rollup* and bundle our application into a single file, *dist/index.js* file and start serving the application. The *dev* command will recompile the code whenever the source code is changed and also reload the changes in the browser.

```
> expense-manager-rollup@1.0.0 build /path/to/your/workspace/expense-manager-
rollup
> rollup -c


rollup v2.36.1
bundles src/index.js → dist\index.js...
LiveReload enabled
http://localhost:10001 ->  /path/to/your/workspace/expense-manager-rollup/dist
created dist\index.js in 4.7s

waiting for changes...
```

Next, open the browser and enter *http://localhost:3000* in the address bar and press enter. *serve* application will serve our webpage as shown below.


Hello World!

Using Parcel bundler

Parcel is fast bundler with zero configuration. It expects just the entry point of the application and it will resolve the dependency itself and bundle the application. Let us learn how to use parcel bundler in this chapter.

First, install the parcel bundler.

```
npm install -g parcel-bundler
```

Open a terminal and go to your workspace.

```
cd /go/to/your/workspace
```

Next, create a folder, *expense-manager-parcel* and move to newly created folder. Also, open the folder in your favorite editor or IDE.

```
mkdir expense-manager-parcel
cd expense-manager-parcel
```

Next, create and initialize the project.

```
npm init -y
```

Next, install React libraries (*react* and *react-dom*).

```
npm install react@^17.0.0 react-dom@^17.0.0 --save
```

Next, install babel and its preset libraries as development dependency.

```
npm install @babel/preset-env @babel/preset-react @babel/core @babel/plugin-
proposal-class-properties -D
```

Next, create a babel configuration file, *.babelrc* under the root folder to configure the babel compiler.

```
{
    "presets": [
        "@babel/preset-env",
        "@babel/preset-react"
    ],
    "plugins": [
        "@babel/plugin-proposal-class-properties"
    ]
}
```

Next, update the *package.json* and include our entry point (*src/index.js*) and commands to build and run the application.

```
...
"main": "src/index.js",
"scripts": {
    "start": "parcel public/index.html",
    "build": "parcel build public/index.html --out-dir dist"
},
...
```

Next, create a *src* folder in the root directory of the application, which will hold all the source code of the application.

Next, create a folder, *components* under *src* to include our React components. The idea is to create two files, *<component>.js* to write the component logic and *<component.css>* to include the component specific styles.

The final structure of the application will be as follows:

```
|-- package-lock.json
|-- package.json
|-- .babelrc
`-- public
    |-- index.html
`-- src
    |-- index.js
    `-- components
    |   |-- mycom.js
    |   |-- mycom.css
```

Let us create a new component, *HelloWorld* to confirm our setup is working fine. Create a file, *HelloWorld.js* under components folder and write a simple component to emit *Hello World* message.

```
import React from "react";

class HelloWorld extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello World!</h1>
      </div>
    );
  }
}
export default HelloWorld;
```

Next, create our main file, *index.js* under *src* folder and call our newly created component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import HelloWorld from './components/HelloWorld';

ReactDOM.render(
  <React.StrictMode>
    <HelloWorld />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Next, create a *public* folder in the root directory.

Next, create a html file, *index.html* (in the *public* folder), which will be our entry point of the application.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Expense Manager :: Parcel version</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/JavaScript" src="../src/index.js"></script>
  </body>
</html>
```

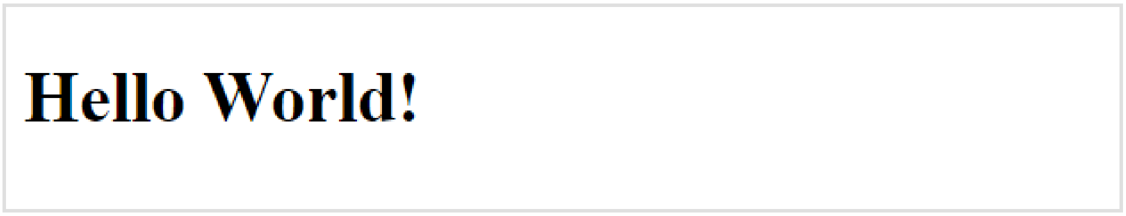
Next, build and run the application.

```
npm start
```

The *npm* build command will execute the *parcel* command. It will bundle and serve the application on the fly. It recompiles whenever the source code is changed and also reload the changes in the browser.

```
> expense-manager-parcel@1.0.0 dev /go/to/your/workspace/expense-manager-parcel
> parcel index.html

Server running at http://localhost:1234
√  Built in 10.41s.
```

Next, open the browser and enter *http://localhost:1234* in the address bar and press enter.


Hello World!

To create the production bundle of the application to deploy it in production server, use *build* command. It will generate a *index.js* file with all the bundled source code under *dist* folder.

```
npm run build

> expense-manager-parcel@1.0.0 build /go/to/your/workspace/expense-manager-
parcel
> parcel build index.html --out-dir dist

√  Built in 6.42s.

dist\src.80621d09.js.map    270.23 KB     79ms
```

```
dist\src.80621d09.js        131.49 KB    4.67s
dist\index.html                 221 B    1.63s
```