Web Technologies Laboratory 01

Laboratory Continuous Assessment (LCA) [As perRubrics]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim: Design and develop a web application using HTML and CSS

Objectives:

1. To understand HTML tags
2. To learn the styling of web pages using CSS

Theory:

1. Basic structure of HTML program
2. CSS Box Model

FAQ:

1. What is a responsive Website?
2. What is the difference between HTML and HTML5?
3. What is a marquee?
4. What are the advantages of using Cascading Style Sheets?

Output: Screenshots of the output to be attached.

HTML: HTML is the standard markup language for creating Web pages. It stands for Hyper Text Markup Language. It describes the structure of Web pages using markup. HTML elements are the building blocks of HTML pages. HTML elements are represented by tags. Hypertext is text that contains hyperlinks. A hyperlink is an automated cross-reference to another location on the same document or to another document which, when selected by a user, causes the computer to display the linked location or document within a very short period of time.

A markup language is a set of tags that can be embedded in digital text to provide additional information about it, including its content, structure and appearance. This information facilitates automated operations on the text, including formatting it for display, searching it and even modifying it.

HTML consists of a set of predefined tags that can be embedded in text by web site designers in order to indicate the details of how web pages are rendered (i.e., converted into a final, easily usable, form) by web browsers. These details include paragraphing, margins, fonts (including style and size), columns, colors (background and text), links, the location of images, text flow around images, tables and user input form elements (such as spaces for adding text and submit buttons).

HTML tags: This is a list of tags used in the HTML language. Each tag starts with a tag opener (a less than sign) and ends with a tag closer (a greater than sign). Many tags have corresponding closing tags which identical except for a slash after the tag opener. Some tags take parameters, called attributes. The attributes are given after the tag, separated by spaces. Certain attributes have an effect simply by their presence, others are followed by an equals sign and a value.

The basic HTML layout is as follows:

```
<html>
 <head>
  <title> …document title… </title>
 </head>
 <body>
  …your page content…
 </body>
</html>
```

The page components are described as follows:

Examples of HTML tags are as follows:

Heading Tags: It defines the heading size in the HTML sheet.

<h1>, <h2>, <h3>, <h4>, <h5>, and <h6>

```
<html>

  <head>
    <title>Heading Example</title>
  </head>

  <body>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

First line of code

Declaration of version of HTML

<html>…</html>

Container for the document

<head>…</head>

<title> Title of page </title>

<body>…</body>

Content of page
```

```
    <h1>This is heading 1</h1>
    <h2>This is heading 2</h2>
    <h3>This is heading 3</h3>
    <h4>This is heading 4</h4>
    <h5>This is heading 5</h5>
    <h6>This is heading 6</h6>
  </body>

</html>
```

Paragraph Tag: Inserts a line space before and after a paragraph

```
<!DOCTYPE html>
<html>

  <head>
    <title>Paragraph Example</title>
  </head>

  <body>
    <p>Here is a first paragraph of text.</p>
    <p>Here is a second paragraph of text.</p>
    <p>Here is a third paragraph of text.</p>
  </body>

</html>
```

Line Break Tag:

```
<!DOCTYPE html>
<html>

  <head>
    <title>Line Break  Example</title>
  </head>

  <body>
    <p>Hello<br />
       You delivered your assignment ontime.<br />
       Thanks<br />
       Mahnaz</p>
  </body>

</html>
```

Image Tag: 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*">

The alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader). The value of the alt attribute should describe the image:

<img src="img_chania.jpg" alt="Flowers in Chania">

Link Tag: HTML links are hyperlinks. In HTML, links are defined with the <a> tag:

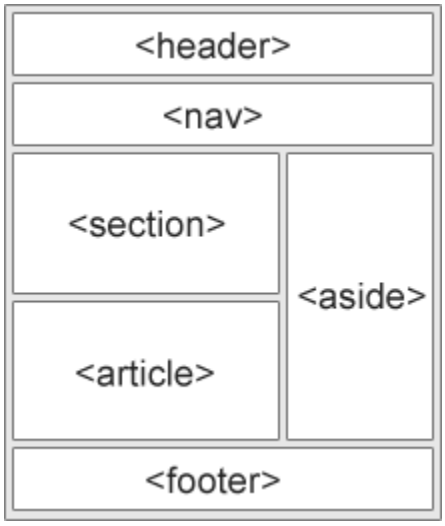
link text

The href attribute specifies the destination address of the link. The link text is the visible part.

Visit our HTML tutorial

HTML5 tags: 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>**



Difference between HTML & HTML5: The major differences in HTML and HTML5 are as follows:

- HTML5 has over its unnumbered predecessor is that it has high-level audio and video support which was not a part of the version specifications in previous HTMLs.
- HTML doesn't allow JavaScript to run within the web browser (it instead runs in the browser interface thread) whereas HTML5 provides full support for JavaScript to run in the background.
- HTML5 supports new kinds of form controls, for example: dates and times, email, number, range, tel, url, search etc.
- HTML5 uses web SQL databases, application cache for temporary storing data, meanwhile, in HTML, only browser cache could be utilized for this purpose.

CSS: Cascading Style Sheets (**CSS**): CSS is the acronym for "Cascading Style Sheet". HTML is used to control the structure of web document. CSS is used to control the style of a web document in a simple and easy way.

CSS stands for **C**ascading **S**tyle **S**heets.

CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**.

CSS **saves a lot of work**. It can control the layout of multiple web pages all at once.

CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using an external CSS file

Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

This example sets the text color of the <h1> element to blue:

Example

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

```
<h1 style="color:blue;">This is a Blue Heading</h1>


</body>

</html>
```

Output:

Internal CSS

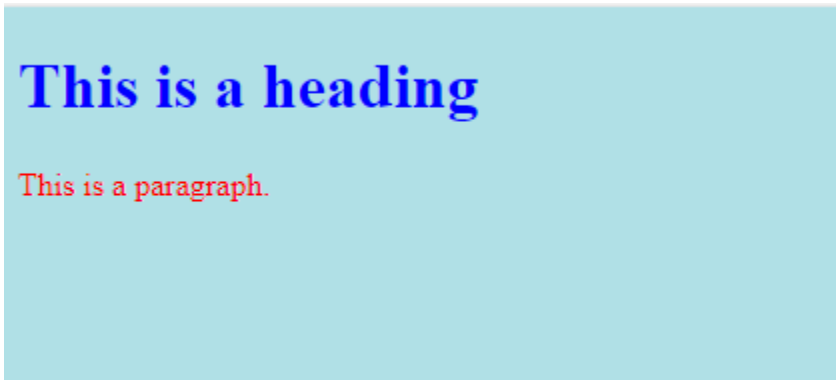
An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the `<head>` section of an HTML page, within a `<style>` element:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Output:



External CSS

An external style sheet is used to define the style for many HTML pages.

With an external style sheet, you can change the look of an entire web site, by changing one file!

To use an external style sheet, add a link to it in the `<head>` section of the HTML page:

Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

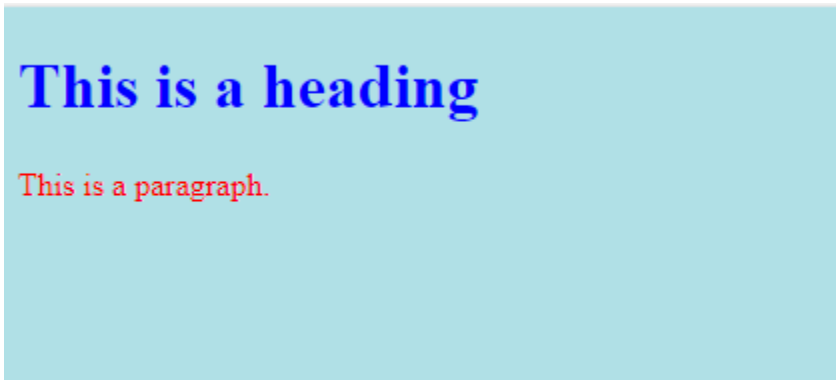
An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is how the "styles.css" looks:

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
```

```
p {
  color: red;
}
```

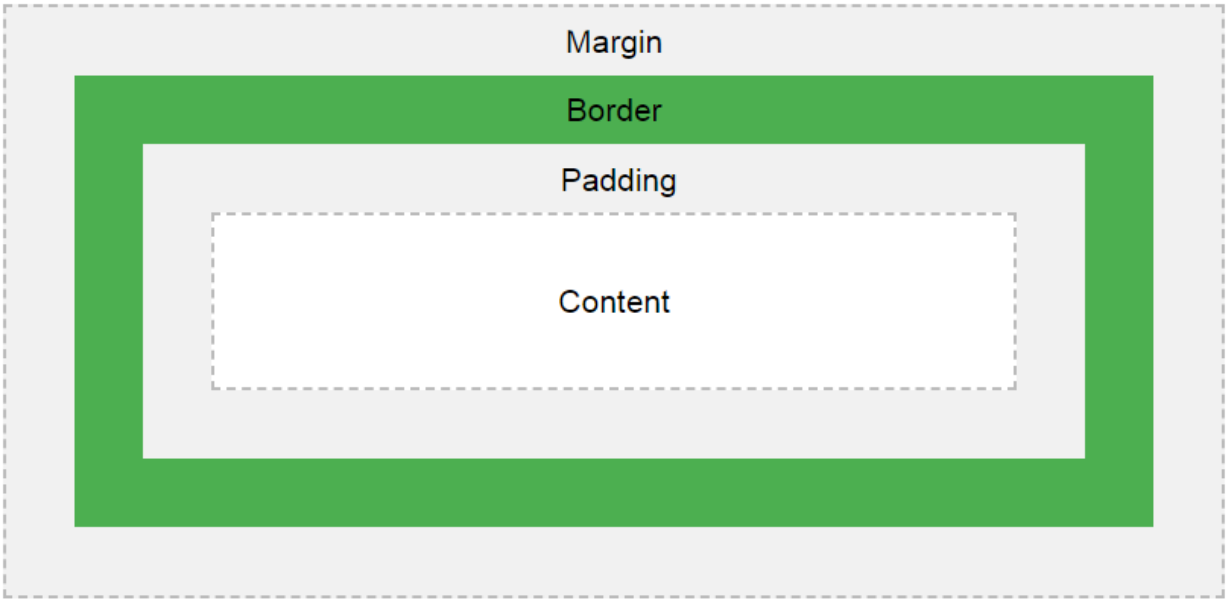
Output:



CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



Explanation of the different parts:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content

- **Margin** - Clears an area outside the border. The margin is transparent

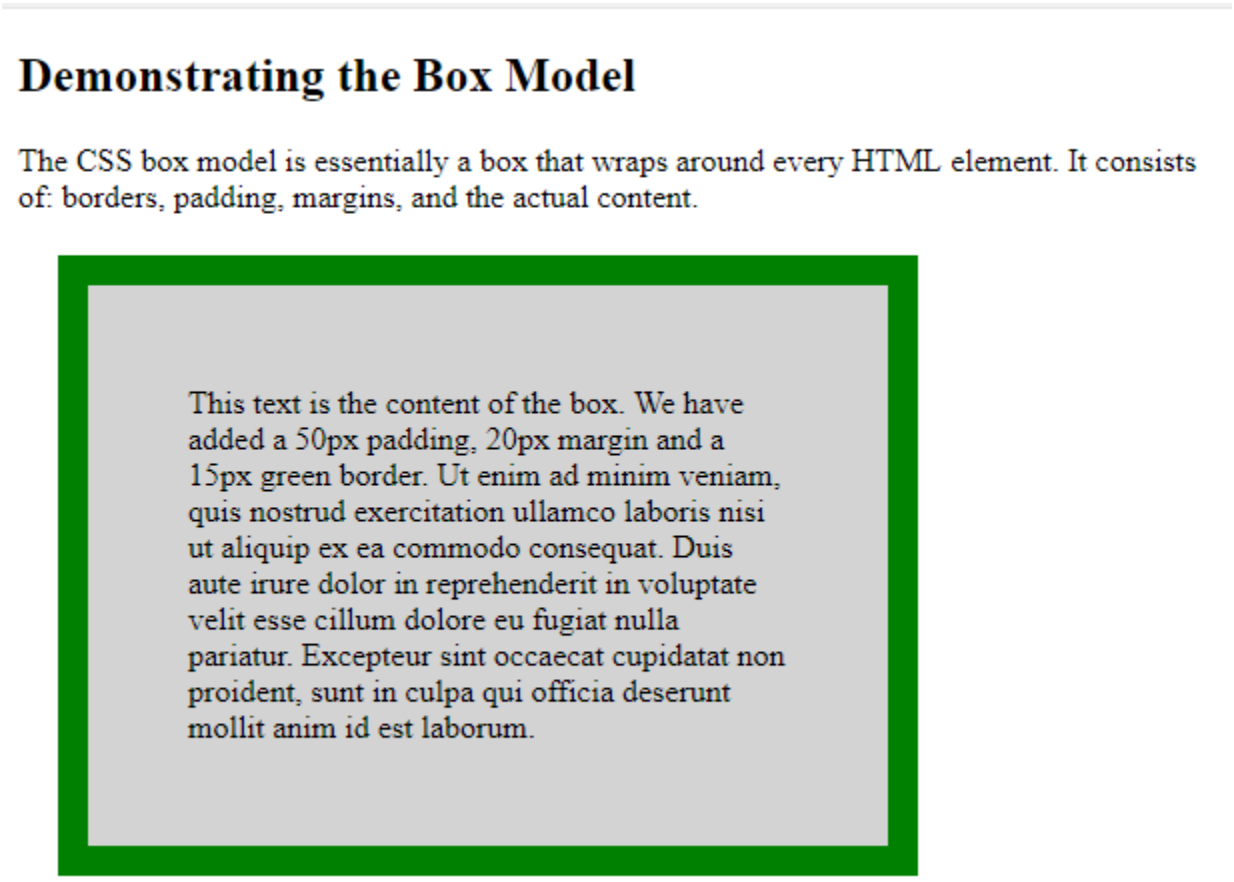
The box model allows us to add a border around elements, and to define space between elements.

Example

Demonstration of the box model:

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

Output:



CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

The CSS element Selector

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

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

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p>Every paragraph will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

Output:

Every paragraph will be affected by the style.

Me too!

And me!

CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

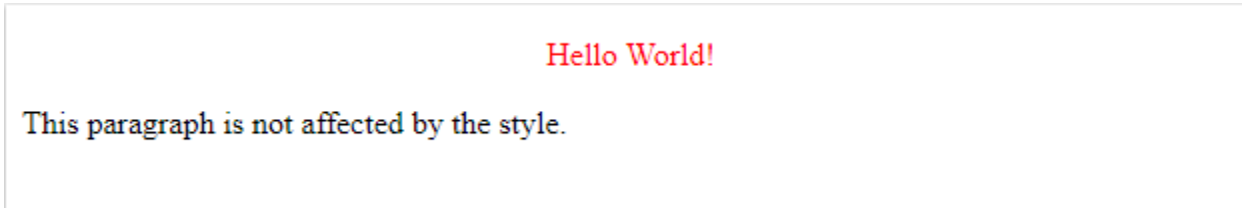
The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: red;
}
</style>
```

```
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

Output:



CSS Class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

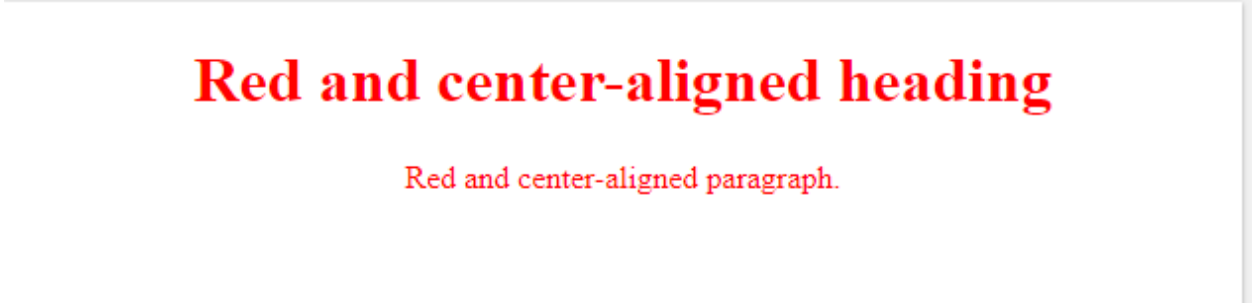
Example

In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {
  text-align: center;
  color: red;
}
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  text-align: center;
  color: red;
```

```
}
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

Output:



CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
<!DOCTYPE html>
<html>
<head>
<style>
* {
```

```
 text-align: center;
 color: blue;
}
</style>
</head>
<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

Output:

Hello world!

Every element on the page will be affected by the style.

Me too!

And me!

CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1 {
  text-align: center;
  color: red;
```

```
}

h2 {
  text-align: center;
  color: red;
}

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

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

Example

In this example we have grouped the selectors from the code above:

```
h1, h2, p {
  text-align: center;
  color: red;
}
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>


<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
```

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


</body>

</html>
```

Output:

Design / execution steps:

- Open Notepad.
- Write the HTML code
- Save the HTML page with the extension .html or .htm
- Write the CSS code in a separate file. Save using the extension .css
- Link both files using the link tag in html file
- View the HTML page in the browser

Input: HTML file and CSS File

Output: Webpage is displayed on browser

Conclusion: Hence, we have designed static web pages using HTML and CSS.

Web Technologies Laboratory 02

Laboratory Continuous Assessment (LCA) [As per Rubrics]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim: Encode the given information using XML document. Construct an external DTD (Document Type Definition) for the XML document. Convert DTD to XML Schema and validate the document. Write a stylesheet to style the data using XSLT.

Objectives:

1. To understand XML
2. To learn validation of XML using DTD and XML Schema
3. To design a stylesheet to style the XML document

Theory:

1. XML
2. DTD and XML
3. XSLT

FAQ:

1. What is the difference between HTML and XML?
2. What are advantages of XML schema over the DTD?
3. What is meant by a well formed XML document?
4. How to validate a XML document using DTD or XML Schema?
5. What are XML namespaces and why are they used?
6. What is the use of eXtensible stylesheet and how to associate it with a XML document?
7. What do the minOccurs and maxOccurs attributes specify?

Output: Screenshots of the output to be attached.

eXtensible Markup Language (XML)

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine readable. It was was developed to overcome the limitations of html. It is used to represent and model data. XML tags are case-sensitive. Major components of XML: Elements, Attributes, Entities (Character, general, unparsed). Unlike html, Element tags in XML can be created by the user and they are not predefined by the language. Each document contains one root element. Attributes can provide parameters for an element. XML must be ***well-formed*** *(*correct syntax , tags match, tags nest, all characters legal)

Advantages of XML:

- Multiple delivery formats
- Information reuse
- Multiple sources of information
- Consistent presentation
- Reduce time spent fiddling with presentation

XML declaration at the start of XML file <?xml version="1.0" ?>

```
<?xml version=”1.0” ?>
<Customer Name=“Bill Smith” Age=“32” MaritalStatus=“Married”>
        <wife>
                <name>Angela</name>
                <age>25</age>
        </wife>
</customer>
```

Document Model:

A document model is used to enforce the structure within a document. Two types of document models can be used for XML validation. **Validation** is a process by which an XML document is validated.

- DTD – Document Type Definition
- XML Schema

DTD (Document Type Definition)

It is used to define the XML elements and attributes.It defines the relationships between different elements and attributes. Two types of DTD's exist:

Internal DTD: DTD exists as part of the XML document

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE greeting [
        <!ELEMENT greeting (#PCDATA)>
]

>

<greeting>Hello, world!</greeting>
```

External DTD: DTD exist as an external file.External DTD is most common. You have to add a DOCTYPE declaration in the xml document before the root element as shown in the example below. Also the standalone attribute of the xml tag needs to be set to "no" to indicate it is depending on the external DTD for validation.

DOCTYPE Declaration contains following fields:

- !DOCTYPE
- Name of root element
- SYSTEM – tells parser that the DTD is external
- Location of DTD file

```
<!DOCTYPE books SYSTEM "book.dtd">
```

ELEMENT Declarations in DTD

- ANY – element can contain child elements or raw text data
- #PCDATA –Parsed Character data -Raw text
- Sequences (sequence of elements)
- Choices (a | b | c)
- Mixed Content (PC data and elements)
- EMPTY

The + suffix indicates that one or more of that element is required at that point

```
<!ELEMENT cataloging_info (abstract, keyword+)>
```

The * suffix indicates that zero, one, or more of that element is required at that point

```
<!ELEMENT catalog (category, cataloging_info, last_updated, copyright, maintainer, composer*, composition*)>
```

Suffixing an element name with a question mark (?) in the content model indicates that either 0 or 1 (but not more than one) of that element are expected at that position

<!ELEMENT composition (title, date, length?, instruments, description?, publisher?)>

A choice indicates one element or another but not both

- A choice is signified by a vertical bar |
- There can be two or more elements in a choice

<!ELEMENT date (year | ISODate)>

Mixed content is both #PCDATA and child elements in a choice, followed by an asterisk

<!ELEMENT description (#PCDATA | ul | cite)*>

Empty Elements – element does not need a closing tag

<!ELEMENT BR EMPTY>

ATTRIBUTE DECLARATION in DTD:

- <!ATTLIST
- element-name
- attr-name
- attr-type
- attr-default >

attr types:

- CDATA: any value is allowed
- (value | ….) enumeration of allowed values
- ID: must be a unique value within the current document
- IDREF, IDREFS: must match an already declared id (used to reference other elements in the document)

attr-default:

- **#REQUIRED:** The attribute must be explicitly provided
- **#IMPLIED:** Attribute is optional, no default provided
- **#FIXED "value" :** Attribute is set to this value always

XML Schema

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.

- **XML Schema Features:**
  - Pattern matching
  - Rich set of data types
  - Attribute grouping
  - Supports XML namespaces
  - Follows XML syntax
- **The XML Schema specification consists of two parts:**
  - XML Schema: **Structures**. This specification consists of a definition language for describing and constraining the content of XML documents
  - XML Schema: **Datatypes**. This specification defines the datatypes to be used in XML schemas.

The XMLSchema language itself is a schema that is located at:

http://www.w3.org/2001/XMLSchema

Defining XMLSchema elements

- **Simple types**

  XML Schema has a lot of built-in data types. The most common types are:

  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

```
<xsd:element name="fruit" type="xsd:string"/>
```

New simple types can be created by specifying values for one or more of the optional facets for the base type. Facets such as length, minLength , maxLength, minInclusive , maxInclusive, totalDigits , fractionDigits, pattern, enumeration

Example

```
<xsd:simpleType name="monthOfYear">
        <xsd:restriction base="xsd:integer">
```

```
<xsd:minInclusive value="1" />
<xsd:maxInclusive value="12" />
</xsd:restriction>
</xsd:simpleType>
```

ComplexType

A complex type element may be considered to be one of four kinds according to the content that it contains:

- Element-only
- Text only
- Empty (with attributes)
- Mixed-content

```
<xs:complexType mixed="true">
```

All elements defined in a complex type must be part of either: Sequence, Choice, Unordered group

Sequence

```
<xs:element name="note">
 <xs:complexType>
  <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="heading" type="xs:string"/>
 <xs:element name="body" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Choice: Only one element that is part of choice can appear once in a XML document

Example

```
<!-- create a complex type element called 'lang' -->
<xsd:complexType name="lang">
 <xsd:choice>
  <xsd:element name="english" type=xsd:string/>
<xsd:element name="italian" type=xsd:string/>
<xsd:element name="german" type=xsd:string/>
```

```
</xsd:choice>
</xsd:complexType>
```

Unordered group: Allows a set of elements to appear once within the XML document in any order

Example

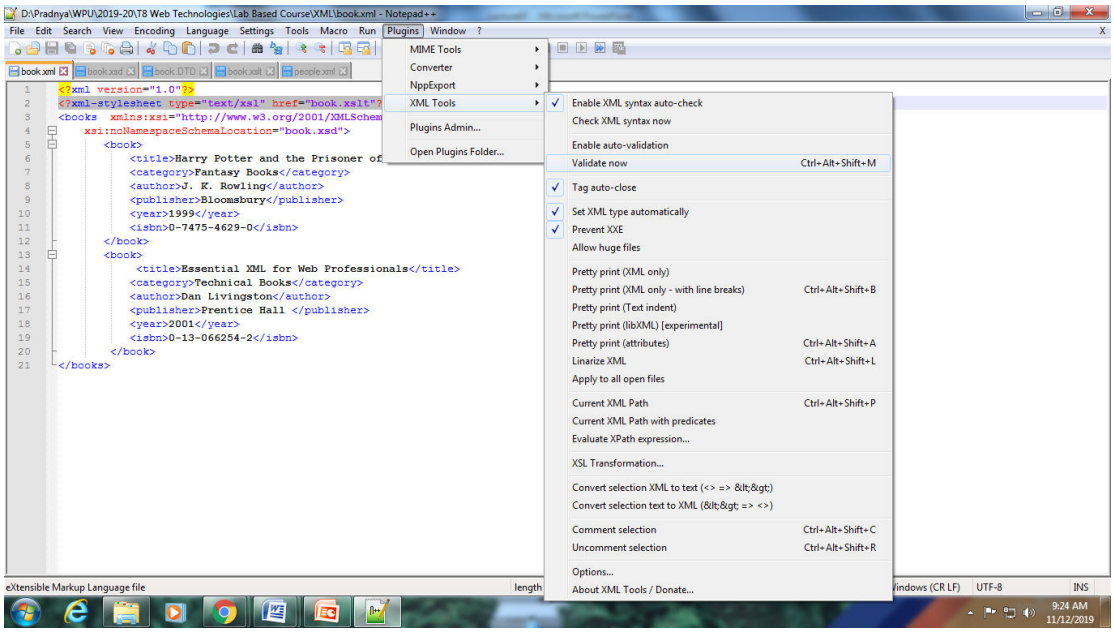
```
<!-- create a complexType element called 'activity' -->
<xsd:complexType name=”activity”>
  <xsd:any>
   <xsd:element name=”activity_type” type=xsd:string/>
 <xsd:element name=”category” type=xsd:string/>
 <xsd:element name=”length” type=xsd:duration/>
  </xsd:any>
</xsd:complexType>
```

Validating an XML document using Notepad++

Open the XML file which you want to validate in Notepad++

Either a DTD or XSD file needs to connected to the XML file using proper declaration.

Then go to Plugins->XMLTools->Validate now



XML Namespaces

As XML tags are created by the users and not defined by the language, there is a possibility of more than one user creating the tag with same name but meaning different depending on their requirement.

e.g. xyz company can create a tag <cost> where the cost of the product is including the GST

abc company might create a tag <cost> but here the cost data may be excluding the GST

Therefore we need some way of differentiating between the tags in the XML document created by different users.

This is achieved via namespaces. These are nothing but unique identifiers

Every XML document has a Namespace declaration at the top which looks like this:

xmlns:prefix=”sample”

xmlns:mark = “http://www.joe.com/ns/cost”

- URL (Uniform Resource Locator)
- URI (Uniform Resource Identifier)
- URN (Universal Resource Name)

The prefix attribute tells that all the tags in this namespaces will have the prefix sample. It is usually a practice to have the unique identifier which is URL, URI or URN as the mark attribute. It is not necessary that the URL or URI or URN even exist in the world.

Default Namespace: In many cases you will find that one namespace is being used most of the time. In such cases you can make that namespace default as follows

```
<myelement
        xmlns=“http://www.ballarat.edu.au/psmith/page1”
        xmlns:n1=“http://www.ballarat.edu.au/psmith/page2"
    <child1>
        I am from default namespace some data </child1>
    <n1:child1>
        I am from the other namespace
     </n1:child1>
</myelement>
```

XSL (eXtensible Stylesheet Language)

It is a styling language for XML. XSLT stands for XSL Transformations. It is used to transform the XML document into a HTML document which can then be rendered. XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically (html, pdf or another XML document). It is far more powerful than CSS. **XSLT style sheet is an XML document and as such must have a root element and an XML declaration**

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

<xsl:stylesheet version="1.0"

xmlns:xsl=http://www.w3c.org/1999/XSL/Transform >

<!-- style sheet rules here -->

</xsl:stylesheet>

Template is formed and the source xml tree is traversed starting from the root node

<xsl:value-of select="."/> Returns value of current node

<xsl:value-of select=".."/> Returns parent

<xsl:value-of select="blah"/> Returns value of "blah" (assuming "blah" is a child of the current node)

<xsl:value-of select ="../sibling"/> Returns value of "sibling" (assuming "sibling" is a sibling of the current node).

EXAMPLE of XML, DTD, XSD and XSLT

Book.xml with external DTD for validation

```
<?xml version="1.0" standalone="no"?>
<books>
    <book>
                <title>Harry Potter and the Prisoner of Azkaban</title>
                <category>Fantasy Books</category>
                <author>J. K. Rowling</author>
                <publisher>Bloomsbury</publisher>
                <year>1999</year>
                <isbn>0-7475-4629-0</isbn>
        </book>
        <book>
                <title>Essential XML for Web professionals</title>
                <category>Technical Books</category>
                <author>Dan Livingston</author>
                <publisher>Prentice Hall </publisher>
                <year>2001</year>
                <isbn>0-13-066254-2</isbn>
              </book>
</books>
```

Corresponding Document Type Definition (book.dtd)

```
<!ELEMENT books (book*)>
<!ELEMENT book (title,category,author,publisher,year,isbn)>
```

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

XML Schema document (book.xsd) corresponding to book.xml

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="books">
   <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" ref="book"/>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

<xsd:element name="book">
  <xsd:complexType>
            <xsd:sequence>
                    <xsd:element name="title" type="xsd:string" />
                    <xsd:element name="category" type="xsd:string" />
                    <xsd:element name="author" type="xsd:string" />
                    <xsd:element name="publisher" type="xsd:string" />
                    <xsd:element name="year" type="xsd:positiveInteger" />
                    <xsd:element name="isbn" type="xsd:string" />
            </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

eXtensible Style sheet (book.xslt) associated with the book.xml

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
 <html>
 <body>
  <h2>My Book Collection</h2>
  <table border="1">
   <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Category</th>
            <th>Author</th>
```

```
                <th>Publisher</th>
                <th>Year</th>
                <th>Isbn</th>
            </tr>
            <xsl:for-each select="books/book">
              <tr>
                <td><xsl:value-of select="title"/></td>
                <td><xsl:value-of select="category"/></td>
                        <td><xsl:value-of select="author"/></td>
                        <td><xsl:value-of select="publisher"/></td>
                <td><xsl:value-of select="year"/></td>
                        <td><xsl:value-of select="isbn"/></td>
              </tr>
            </xsl:for-each>
          </table>
        </body>
        </html>
      /xsl:template>

</xsl:stylesheet>
```

In order to attach a particular DTD to a XML document add the following line before the root element in xml file

```
<!DOCTYPE books SYSTEM "book.dtd">
```

In order to associate a xml schema file to the XML document add the xsd file as attribute value of xsi:noNameSpaceSchemaLocation of the root element in xml file as follows

```
<books xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="book.xsd">
```

In order to associate a xslt stylesheet to the xml document add following line before the root element in xml file

```
<?xml-stylesheet type="text/xsl" href="book.xslt"?>
```

Web Technologies Laboratory 03

Laboratory Continuous Assessment (LCA) [As per Rubrics]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim: Write a program to design Student registration form by using HTML, CSS and perform following validations using JavaScript:
a) All fields mandatory,
b) Phone Number, Email Address, Zip code validation etc.

Objectives:

1. To understand what is form validation.
2. To know its importance.
3. To learn how to apply various techniques to implement it.

Theory:

1. Different types of form validations.
2. Different client side validations that can be performed on the form.

FAQ:

1. What is JavaScript?
2. Enumerate the differences between Java and JavaScript?
3. What are JavaScript Data Types?
4. Write 3 reasons why Form validations are important.
5. Explain the "required" attribute in HTML5.

Output: Screenshots of the output to be attached.

JavaScript 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.

Why to Learn Javascript

Javascript is a MUST for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. Some of the key advantages of learning Javascript:

- Javascript is the most popular **programming language** in the world. Once learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
- Javascript is everywhere, it comes installed on every modern web browser.
- Javascript helps you create really beautiful and crazy fast websites.
- JavaScript usage has now extended to mobile app development, desktop app development, and game development.
- Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.
- Great thing about Javascript is that it has tons of frameworks and Libraries already developed which can be used directly in software development to reduce your time to market.

Advantages of JavaScript

The merits of using JavaScript are −

- **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

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- 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 multi-threading or multiprocessor capabilities.

Applications:

- **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
- **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

JavaScript Development Tools

Some of them are listed here −

- **Microsoft FrontPage** − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX** − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5** − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Javascript Syntax:

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 your web page, but it is normally recommended that you should keep it within the **<head>** tags.

A simple syntax:

```
<script ...>
   JavaScript code
</script>
```

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 −

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

Hello World using Javascript

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

What is Form Validation?

Go to any popular site with a registration form, and you will notice that they provide feedback when you don't enter your data in the format they are expecting. You'll get messages such as:

- "This field is required" (You can't leave this field blank.)
- "Please enter your phone number in the format xxx-xxxx" (The form enforces three numbers followed by a dash, followed by four numbers.)
- "Please enter a valid email address" (Used if your entry is not in the format of "somebody@example.com.")
- "Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number."

Why do we insist on validating our forms? There are three main reasons:

1. **We want to get the right data, in the right format.** Our applications won't work properly if our users' data is stored in the incorrect format, if they don't enter the correct information, or if they omit information altogether.
2. **We want to protect our users' accounts**. Forcing our users to enter secure passwords makes it easier to protect their account information.
3. **We want to protect ourselves**.

Different types of form validation:

There are two different types of form validations on the web:

- **Client-side validation**: is validation that occurs in the browser before the data has been submitted to the server. Client-side validation is more user-friendly than server-side validation because it gives an instant response. Client-side validation is further subdivided into the following categories:
  - **JavaScript** validation is coded using JavaScript. This validation is completely customizable.
  - **Built-in form validation** uses HTML5 form validation features. This validation generally doesn't require JavaScript. Built-in form validation has better performance than JavaScript. And, while highly customizable, native validation is not as customizable as JavaScript.
- **Server-side validation:** is validation that occurs on the server after the data has been submitted. Server-side code is used to validate the data before the data is saved in the database or otherwise used by the application. If the data fails validation, a response is sent back to the client with corrections that the user needs to make. Server-side validation is your application's last line of defense against incorrect or malicious data.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

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

Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>
   <head>
      <title>Form Validation</title>
      <script type = "text/javascript">
         <!--
            // Form validation code will come here.
         //-->
      </script>
   </head>

   <body>
      <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate()
         <table cellspacing = "2" cellpadding = "2" border = "1">

            <tr>
               <td align = "right">Name</td>
               <td><input type = "text" name = "Name" /></td>
            </tr>

            <tr>
               <td align = "right">EMail</td>
               <td><input type = "text" name = "EMail" /></td>
            </tr>

            <tr>
               <td align = "right">Zip Code</td>
               <td><input type = "text" name = "Zip" /></td>
            </tr>

            <tr>
               <td align = "right">Country</td>
               <td>
                  <select name = "Country">
                     <option value = "-1" selected>[choose yours]</option>
                     <option value = "1">USA</option>
                     <option value = "2">UK</option>
                     <option value = "3">INDIA</option>
                  </select>
               </td>
            </tr>

            <tr>
               <td align = "right"></td>
               <td><input type = "submit" value = "Submit" /></td>
            </tr>

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

Basic Form Validation

In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
   <!--
      // Form validation code will come here.
      function validate() {

         if( document.myForm.Name.value == "" ) {
            alert( "Please provide your name!" );
            document.myForm.Name.focus() ;
            return false;
         }
         if( document.myForm.EMail.value == "" ) {
            alert( "Please provide your Email!" );
            document.myForm.EMail.focus() ;
            return false;
         }
         if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
            document.myForm.Zip.value.length != 5 ) {

            alert( "Please provide a zip in the format #####." );
            document.myForm.Zip.focus() ;
            return false;
         }
         if( document.myForm.Country.value == "-1" ) {
            alert( "Please provide your country!" );
            return false;
         }
         return( true );
      }
   //-->
</script>
```

Data Format Validation

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

```
<script type = "text/javascript">
   <!--
      function validateEmail() {
         var emailID = document.myForm.EMail.value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");

         if (atpos < 1 || ( dotpos - atpos < 2 )) {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
         }
         return( true );
      }
   //-->
</script>
```

Alphanumeric validation

Javascript function to check if a field input contains letters and numbers only:

```
function alphanumeric(inputtxt)
{
 var letterNumber = /^[0-9a-zA-Z]+$/;
 if((inputtxt.value.match(letterNumber))
 {
   return true;
 }
else
 {
   alert("message");
   return false;
 }
}
```

Conclusion:

Written and successfully executed Client Side validations on a web page using JavaScript.

Web Technologies Laboratory 04

Laboratory Continuous Assessment (LCA) [As per Rubrics]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim: Write a client-side script with JavaScript to access and manipulate Document Object Model (DOM) objects in an HTML web page.

Objectives:

1. To learn basic functioning of DOM objects.
2. To understand how to retrieve and manipulate DOM objects in an HTML web page.

Theory:

1. HTML Document Object Model
2. DOM methods, properties
3. Javascript Event Handlers
4. Linking external Javascript file to HTML file

FAQ:

1. How do you access an element of DOM having id=”abc”?
2. Provide a sample code for accessing DOM element having class=”blue”.
3. How to access all the <p> elements using DOM methods?
4. Give an example of how to modify an attribute value using DOM.
5. How do you modify the CSS properties using HTML DOM?
6. Give an example code of adding and removing a node from the DOM tree.

Output: Screenshots of the output to be attached.

Document Object Model

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.

- The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.
- A Web page is a document. This document can be either displayed in the browser window or as the HTML source. But it is the same document in both cases.
- The Document Object Model (DOM) represents that same document so it can be manipulated. The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.

For example, following html code represents sample code,

HTML Snippet

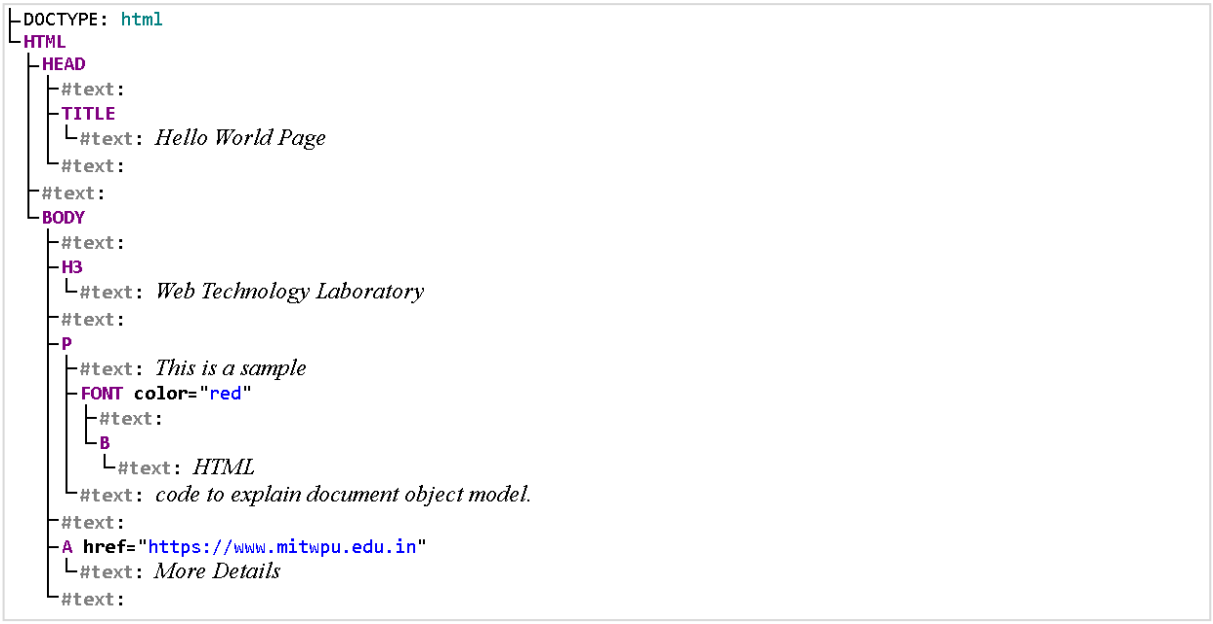
```
<!DOCTYPE html>
<html>
<head> <title> Hello World Page </title> </head>
<body>
<h3> Web Technology Laboratory </h3>
<p>
This is a sample <font color="red"> <b> HTML </b></font>
code to explain document object model.
</p>
<a href="https://www.mitwpu.edu.in"> More Details </a>
</body>
</html>
```

Rendered View (Web Browser)

Web Technology Laboratory

This is a sample HTML code to explain document object model.

More Details



DOM and JavaScript

- The DOM is not a programming language, but without it, the JavaScript language wouldn't have any model or notion of web pages, HTML documents, XML documents, and their component parts (e.g. elements).
- Every element in a document—the document as a whole, the head, tables within the document, table headers, text within the table cells—is part of the document object model for that document, so they can all be accessed and manipulated using the DOM and a scripting language like JavaScript.
- The page content is stored in the DOM and may be accessed and manipulated via JavaScript, so that we may write this approximate equation:

  API (HTML or XML page) = **DOM + JS** (scripting language)
- The DOM was designed to be independent of any particular programming language, making the structural representation of the document available from a single, consistent API.

  DOM defines the following

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

Common Properties in HTML DOM

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Common Methods Involved in HTML DOM

- HTML DOM methods are used as a way to generate the result without much of the written code and it can be easily used.
- HTML DOM methods provide a way to manipulate HTML elements without much efforts.
- In DOM context, Document and window objects are the objects whose interfaces you generally use most often in DOM programming.
- In simple terms, the window object represents something like the browser, and the document object is the root of the document itself.
- The following is a brief list of common APIs in web using the DOM:
  document.getElementById(id) –
  Allows to access HTML element associated with a specified id.
  document.getElementsByTagName(name) –
  Allows to get all elements using tag name that is associated with it.
  document.getElementsByClassName()
  Allows to get all elements using class name associated with it
  document.appendChild(node) –
  Allows to insert a child node to x and append the child node at the end of the tree.
  document.removeChild(node) –
  Allows to remove a child node from x and remove from the tree area also.

Events in HTML DOM

HTML DOM Events allow JavaScript to register different event handlers on elements in an HTML document.
Example 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

Accessing the DOM Elements

- Different browsers have different implementations of the DOM, and these implementations exhibit varying degrees of conformance to the actual DOM standard (a subject we try to avoid in this documentation), but every web browser uses some document object model to make web pages accessible via JavaScript.

Accessing Element Having particular id

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

<h2>My First Page</h2>

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

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

</body>
</html>
```

Accessing element not having an id attribute (getElementsByTagName())

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

<h2>Finding HTML Elements by Tag Name</h2>
<p>This is first p element on this html page</p>
<p>This is second p element on the page</p>
<p id="demo">some text</p>

<script>
var y = document.getElementsByTagName("p");
var a = document.getElementById("demo");
a.innerHTML = 'Accessing text in the second p tag: ' + y[1].innerHTML;
</script>

</body>
</html>
```

Accessing element not having a particular class (getElementsByClassName())

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

<div class="example">First div element with class="example".</div>
<div class="example">Second div element with class="example".</div>
<button onclick="myFunction()">Change Text</button>

<script>
function myFunction() {
  var x = document.getElementsByClassName("example");
  x[0].innerHTML = "Hello World!";
}
</script>

</body>
</html>
```

Modifying the CSS properties of Element

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

<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>
</body>
</html>
```

Adding Elements to HTML DOM Tree:

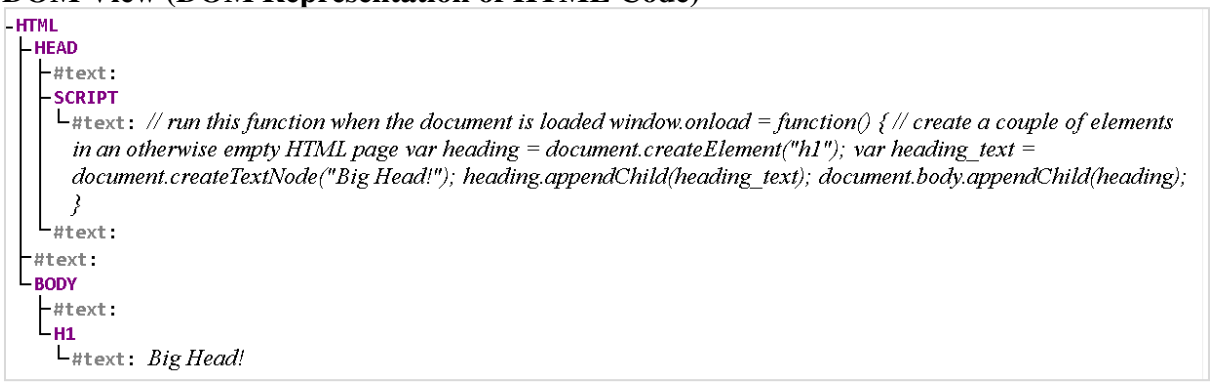
Another example. This function creates a new H1 element, adds text to that element, and then adds the H1 to the tree for this document:

HTML Code

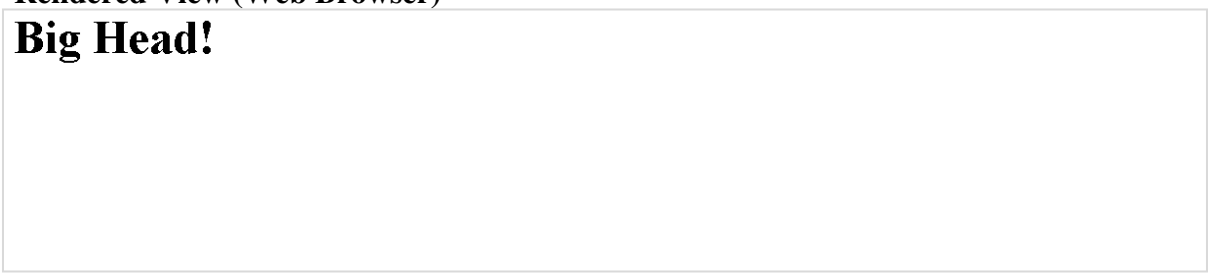
```
<html>
 <head>
  <script>
    // run this function when the document is loaded
    window.onload = function() {
```

```
    // create a couple of elements in an otherwise empty HTML page
    var heading = document.createElement("h1");
    var heading_text = document.createTextNode("Big Head!");
    heading.appendChild(heading_text);
    document.body.appendChild(heading);
  }
 </script>
</head>
<body>
</body>
</html>
```

DOM View (DOM Representation of HTML Code)



Rendered View (Web Browser)



Removing Elements from HTML DOM Tree:

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

<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>


<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
  var elmnt = document.getElementById("p1");
  elmnt.remove();
}
</script>

</body>
</html>
```

Javascript Event Handlers

Event handlers are the functions which get executed when a particular event occurs. Sample code for event handler using external Javascript file shown below.

bgcolor.html file code:

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <title>Learning the DOM</title>
  </head>

  <body>
    <h1>Document Object Model</h1>
    <button id="changeBackground">Change Background Color</button>

    <script src="script.js"></script>
  </body>

</html>
```

script.js file code:

```
b = document.getElementById('changeBackground');

b.addEventListener('click', myfunction);
function myfunction()
{
  document.body.style.backgroundColor = 'red';
}
```

Laboratory Assignment:

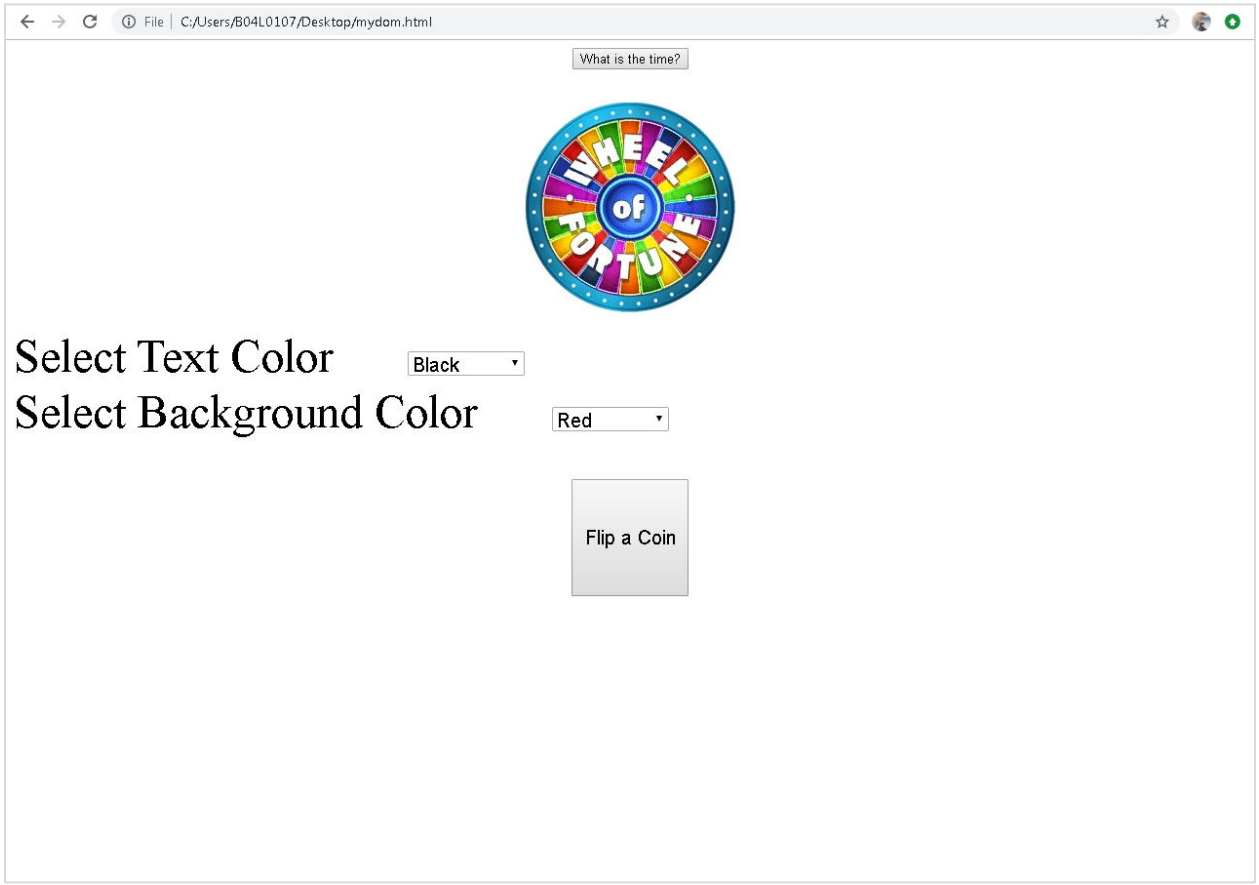
HTML Code Snippet:

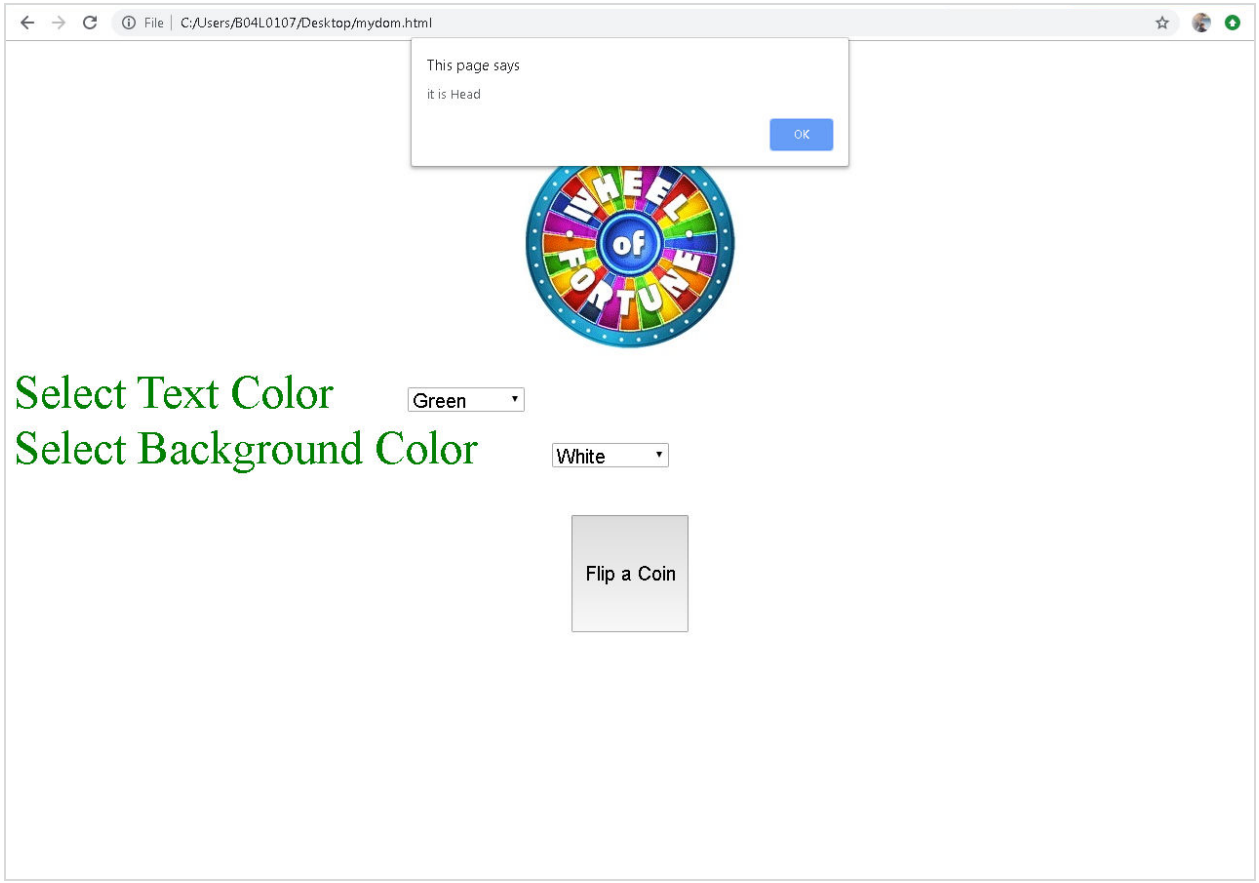
```
<html>
  <head>
    <title>DOM Tests</title>
    <script type="application/javascript">
 function changeImage() {
//alert(document.getElementById("im4").src);

        if (document.getElementById("im4").src ==
"file:///C:/Users/B04L0107/Desktop/img1.jpg")
        {
            document.getElementById("im4").src = "img2.gif";
        }
        else{  document.getElementById("im4").src = "img1.jpg";  }       }

function coinFlip()
{     (Math.floor(Math.random() * 2) == 0) ? alert('it is Head') : alert('it is
Tail') }



function setBodyAttr(attr, value)
{
      if (document.body)
       eval('document.body.'+attr+'="'+value+'"');
      else
       notSupported();
    }
</script>
</head>

<body>
<center>
```

```
<button onclick="getElementById('demo').innerHTML=Date()">What is the
time?</button>
<p id="demo"></p>
</center>
<center>
        <img src="img1.jpg" id="im4" onclick="changeImage()" />

</center>
      <font size=20>Select Text Color      </font>

        <select style="font-size:20px; width: 120px; height: 25px;"
onChange="setBodyAttr('text',this.options[this.selectedIndex].value);">

          <option value="black">Black

          <option value="red">Red
   <option value="Green">Green

          <option value="purple">Purple

          <option value="blue">Blue
   <option value="brown">Brown

        </select><br>
<font size=20>Select Background Color      </font>

        <select style="font-size:20px; width: 120px; height: 25px;"
onChange="setBodyAttr('bgColor',

        this.options[this.selectedIndex].value);">
      <option value="red">Red
   <option value="Green">Green

          <option value="purple">Purple

          <option value="blue">Blue
   <option value="brown">Brown

          <option value="white">White

          <option value="lightgrey">Gray

        </select>
<br>
       <br><br>
    <center>
```

```
        <input type="button" style="font-size:20px; width: 120px; height: 120px;"
value="Flip a Coin" onclick="coinFlip()" />
</center>
</form>
    </div>
</body>
</html>
```

Rendered View





Platform/ Languages used: Windows/Ubuntu, Web Browser(Google Chrome/Mozilla Firefox), HTML Code Editor(Notepad/Adobe Dreamweaver/Sublime Text)

Ref:
https://developer.mozilla.org/en-S/docs/Web/API/Document_Object_Model/

Conclusion:
Hence in this way we have implemented a program with HTML & JavaScript to access and manipulate Document Object Model (DOM) objects in an HTML web page.

Web Technologies Laboratory 05

Laboratory Continuous Assessment (LCA) [As per Rubrics]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim: Write server side script in PHP to perform form validation and create database application using PHP and MySQL to perform insert, update, delete and search operations.

Objectives:

1. To understand Server Side Script
2. To learn database connectivity in PHP
3. To perform insert, update, delete and search operations on database

Theory:

1. PHP Architecture
2. Steps for Database connectivity in PHP

FAQ:

1. What are Client Side Scripts and Server Side Scripts?
2. What are the advantages of Server Side Scripting?
3. Write some differences between Client Side Scripting and Server Side Scripting?
4. List some Server Side Scripting languages.
5. What is XAMPP and phpMyAdmin?
6. What are the two ways to connect to database in PHP?

Output: Screenshots of the output to be attached.

PHP Introduction

The term PHP is an acronym for **PHP: Hypertext Preprocessor**. PHP is a server-side scripting language designed specifically for web development. Websites like www.facebook.com, www.yahoo.com are also built on PHP. One of the main reasons behind this is that PHP can be easily embedded in HTML files and HTML codes can also be written in a PHP file.

The thing that differentiates PHP with client-side language like HTML is, PHP codes are executed on server whereas HTML codes are directly rendered on the browser. PHP codes are first executed on the server and then the result is returned to the browser. PHP files can contain text, HTML, CSS, JavaScript, and PHP code. PHP files have extension ".php"

The only information that the client or browser knows is the result returned after executing the PHP script on the server and not the actual PHP codes present in the PHP file. Also, PHP files can support other client-side scripting languages like CSS and JavaScript.

Before learning PHP, one should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

What can PHP do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is easy to learn and runs efficiently on the server side

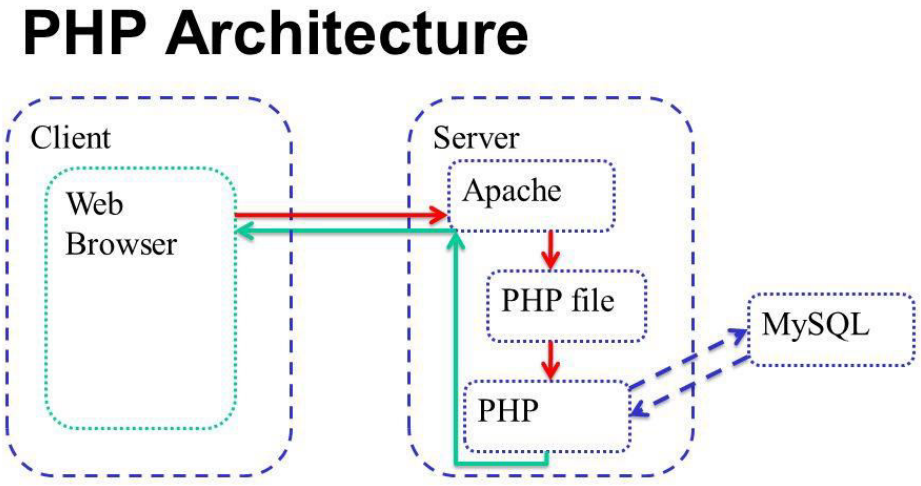
PHP Architecture: PHP is based on the Model-View-Controller is a concept involved in software development evolved in the late 1980s. It is a software architecture built on the idea that the logic of an application should be separated from its presentation. A system developed on the MVC architecture should allow a front-end developer and a back-end developer to work on the same system without interfering with each other.

- **Model**
  Model is the name given to the component that will communicate with the database to manipulate the data. It acts as a bridge between the View component and the Controller

component in the overall architecture. It doesn't matter to the Model component what happens to the data when it is passed to the View or Controller components.

- **View**
  The View requests for data from the Model component and then its final output is determined. View interacts with the user, and then transfers the user's reaction to the Controller component to respond accordingly. An example of this is a link generated by the View component, when a user clicks and an action gets triggered in the Controller.
- **Controller**
  The Controller's job is to handle data that the user inputs or submits through the forms, and then Model updates this accordingly in the database. The Controller is nothing without the user's interactions, which happens through the View component.

- A simple way to understand how MVC works is given below.
  1) A user interacts with View.
  2) The Controller handles the user input, and sends the information to the model.
  3) Then the Model receives the information and manipulates it (either saving it or updating it by communicating with the database).
  4) The View checks the state of the Model and responds accordingly (lists updated information).



PHP Syntax

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
A PHP script can be placed anywhere in the document.
A PHP script starts with <?php and ends with ?>:

```
<?php
// PHP code goes here
?>
```

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Output:

My first PHP page
Hello World!

Note: PHP statements end with a semicolon (;)

Steps to Run a PHP program in XAMPP Server

PHP program can be run under various like WAMP, XAMPP etc.

- **WAMP Server**: This server is a web development platform which helps in creating dynamic web applications.
- **XAMPP Server**: It is a free open source cross-platform web server package.

Download XAMPP from the following link:
https://www.apachefriends.org/download.html
After downloading, just follow the following step to start XAMPP server:

Step 1

Install XAMPP

Step 2

Assume you installed XAMPP in C Drive.
Go to: **C:\xampp\htdocs**

Create your own folder; name it for example as **tutorialspoint**.

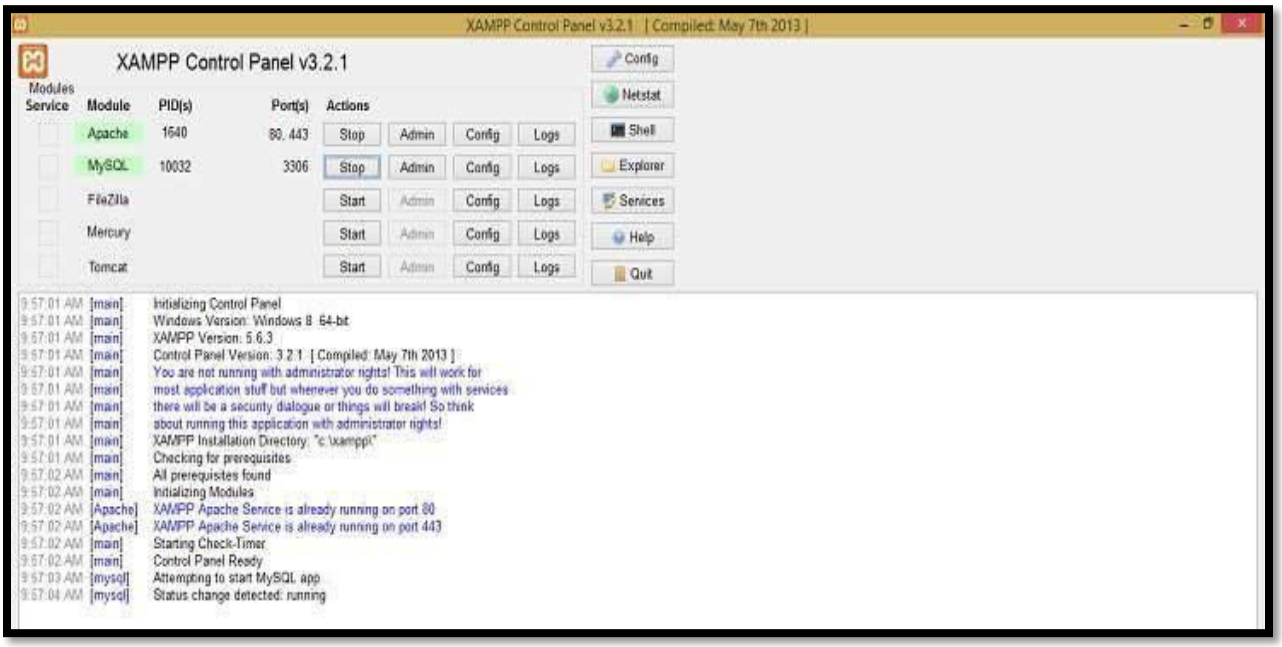
Step 3

Now create your first php program in XAMPP and name it as **"add.php"**:

```
<html>
<head><title>Addition php</title></head>
<body>

<?php
        # operator
        print "<h2>php program to add two numbers...</h2><br />";
              $val1 = 20;
              $val2 = 20;
              $sum = $val2 + $val2;   /* Assignment operator */
              echo "Result(SUM): $sum";
?>

</body>
</html>
```

Step 4

Now double click on "**XAAMP CONTROL PANEL**" on desktop and START "**Apache**" (icon also appears on the bottom)



Step 5

Type **localhost** on your browser and press enter:

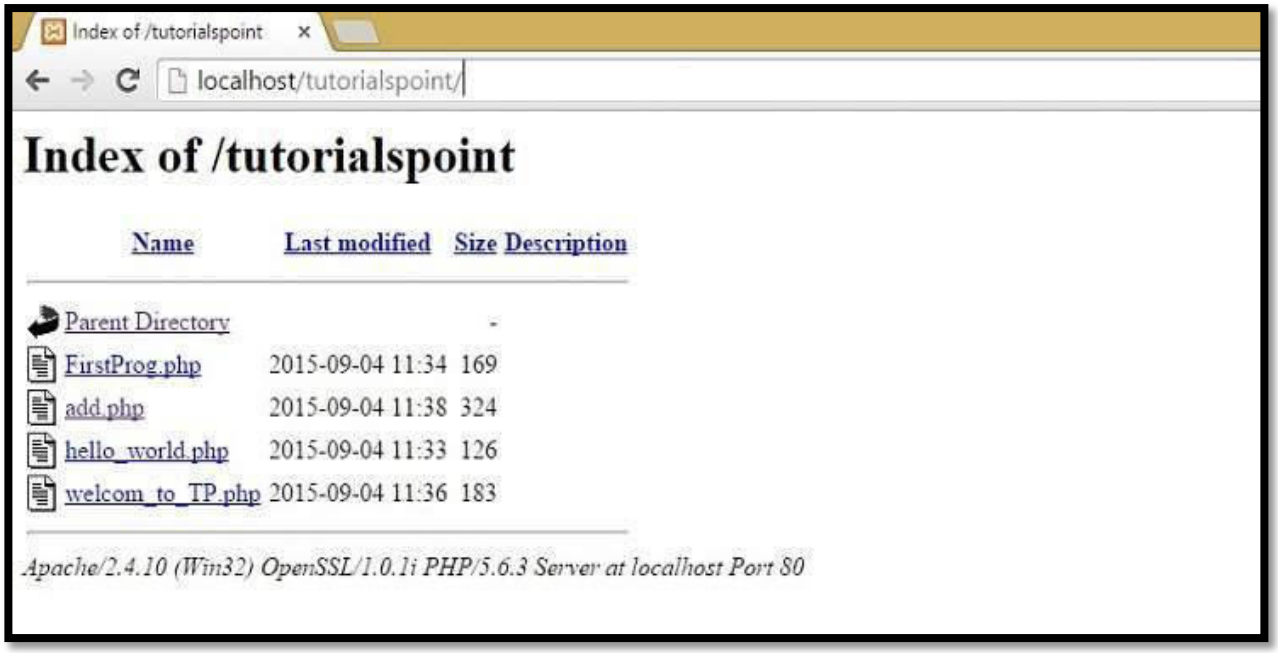
It will show the following:

Step 6

Now type the following in browser:

http://localhost/tutorialspoint/

Below screenshot shows php files created under folder "tutorialspoint"



Step 7

Click on "**add.php**" and it will show the following:

PHP Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.
The example below displays a simple HTML form with two input fields and a submit button:
Example

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?> <br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

The output could be something like this:

Welcome John
Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:
Example

```
<html>
<body>
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_GET["name"]; ?> <br>
Your email address is: <?php echo $_GET["email"]; ?>
</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

GET vs. POST

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. GET may be used for sending non-sensitive data.

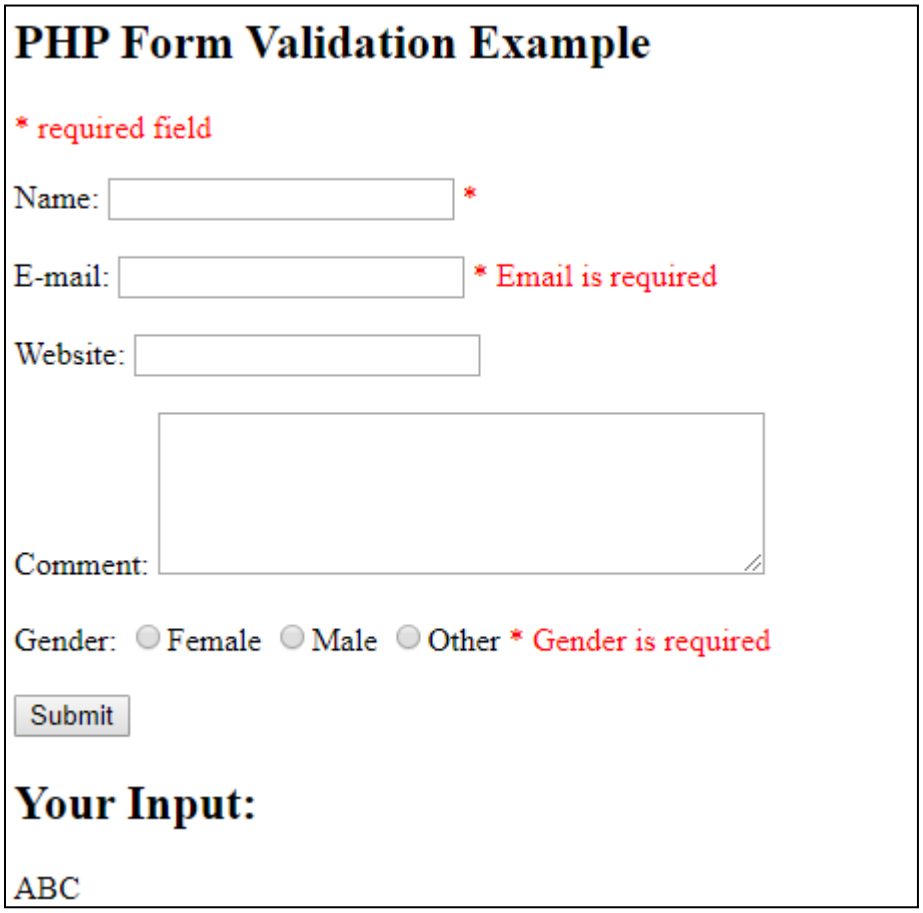
Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send. However, because the variables are not displayed in the URL, it is not possible to bookmark the page. Developers prefer POST for sending form data.

PHP Form Validation

The HTML form which we will be using; contains various input fields: required and optional text fields, radio buttons, and a submit button:



The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

The Form Element
The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"])
;?>">
```

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The **$_SERVER["PHP_SELF"]** is a super global variable that returns the filename of the currently executing script. So, the **$_SERVER["PHP_SELF"]** sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The **htmlspecialchars()** function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

In the following code, some new variables are added: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

PHP - Display the Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields).

PHP Forms - Validate Name, E-mail and URL

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name))
{
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var()** function.

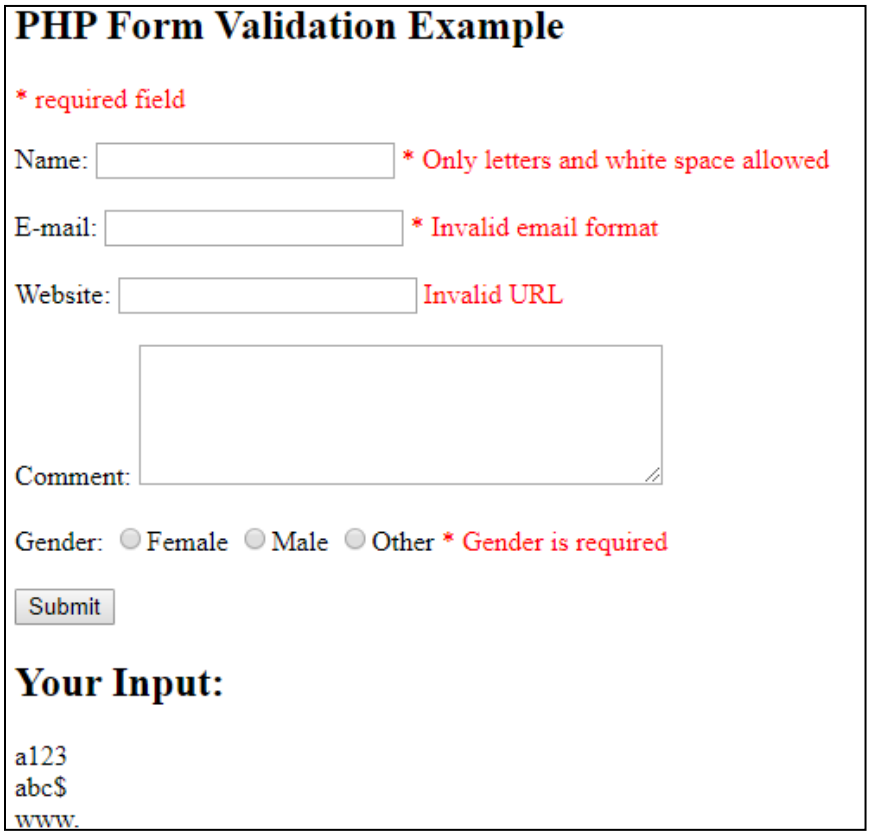
In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website))
{
  $websiteErr = "Invalid URL";
 }
```



The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

PHP - Keep the Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows dashes in
the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
```

```
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]
);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?
></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
```

```
 <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

PHP Form Validation Example

* required field

Name: ABC12 * Only letters and white space allowed

E-mail: ABC123@ * Invalid email format

Website:

Comment:

Gender: Female Male Other * Gender is required

Submit

Your Input:

ABC12
ABC123@

PHP MySQL Database

With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web and runs on a server
- MySQL is very fast, reliable, and easy to use
- MySQL compiles on a number of platforms, is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

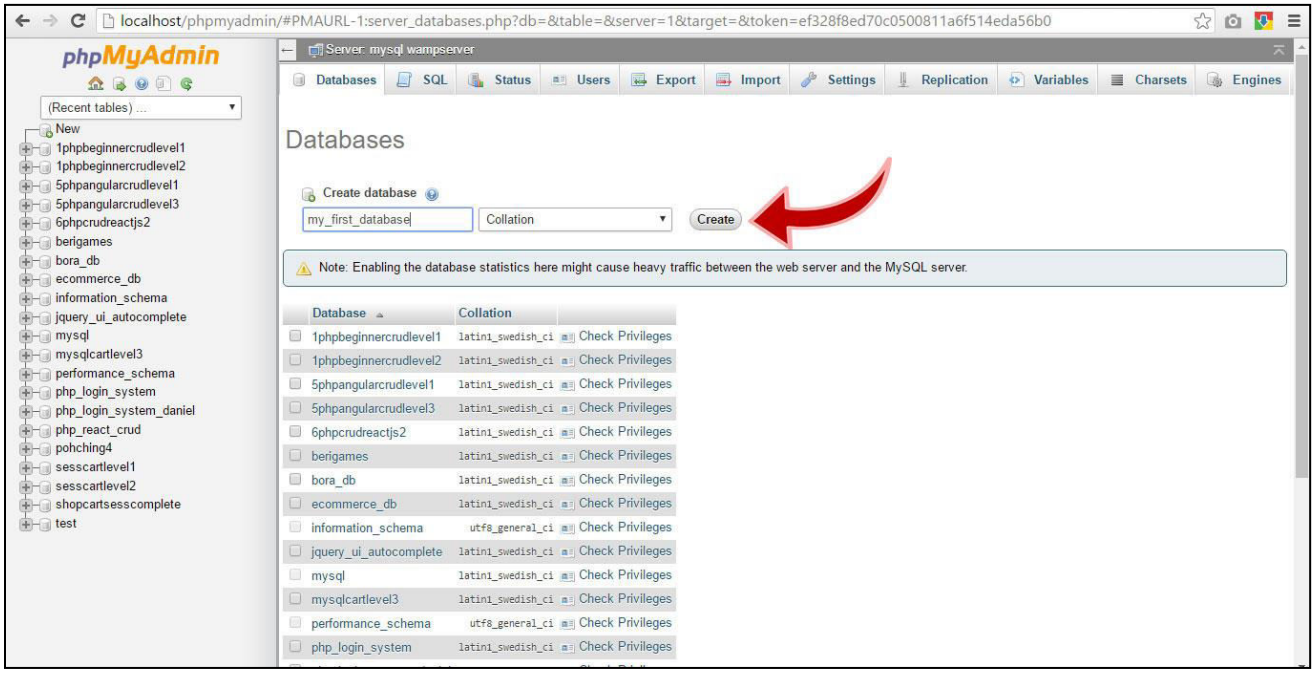
Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Steps to manage MYSQL with PHPMYADMIN

phpMyAdmin is a free and open source tool written in PHP intended to handle the administration of MySQL with the use of a web browser. In the following example, we will see how easy we can handle MySQL with phpMyAdmin.

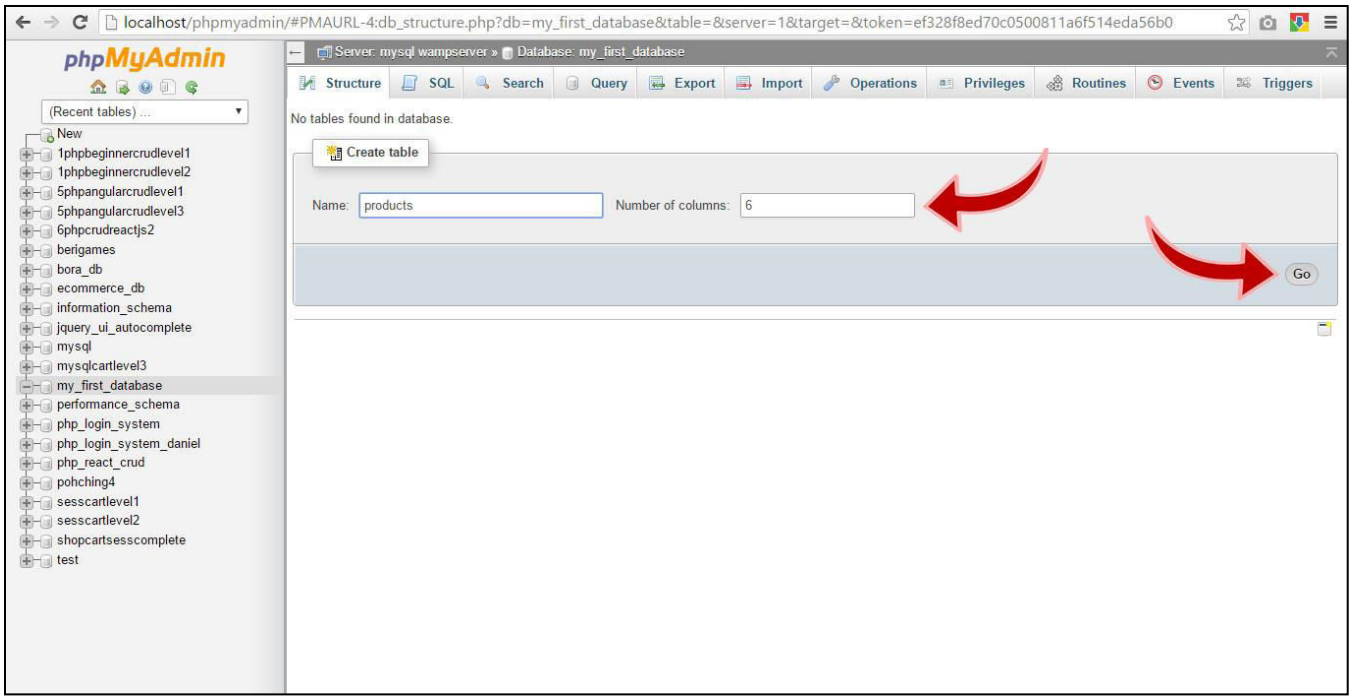
Create a Database

1. Go to **http://localhost/phpmyadmin/**
2. Click the "**New**" link on the upper left corner (under recent tables)
3. Fill out the "**Database Name**" field with "**my_first_database**".
4. Click the "**Create**" button

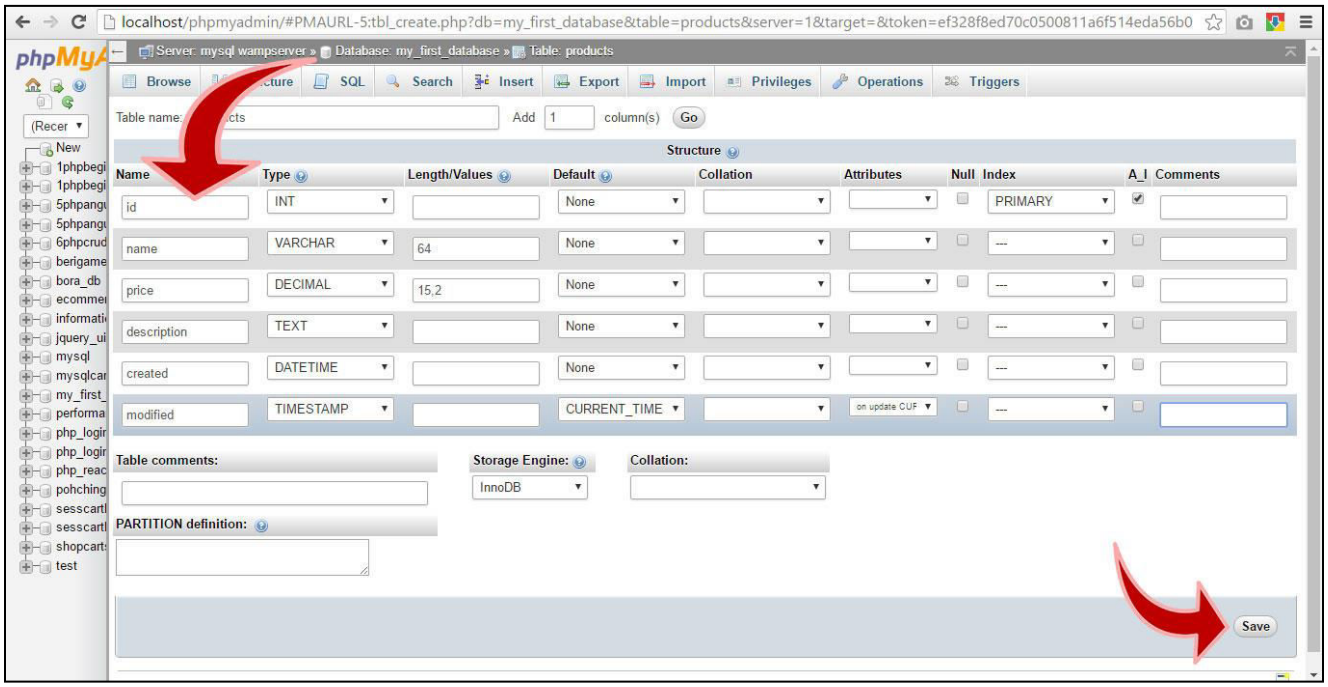


Create a Table

1. Click "**my_first_database**" on the left side of the screen
2. On the "Create Table" section, fill out the *Name* with "**products**" and *Number of Columns* with "**6**"
3. Click "**Go**" button

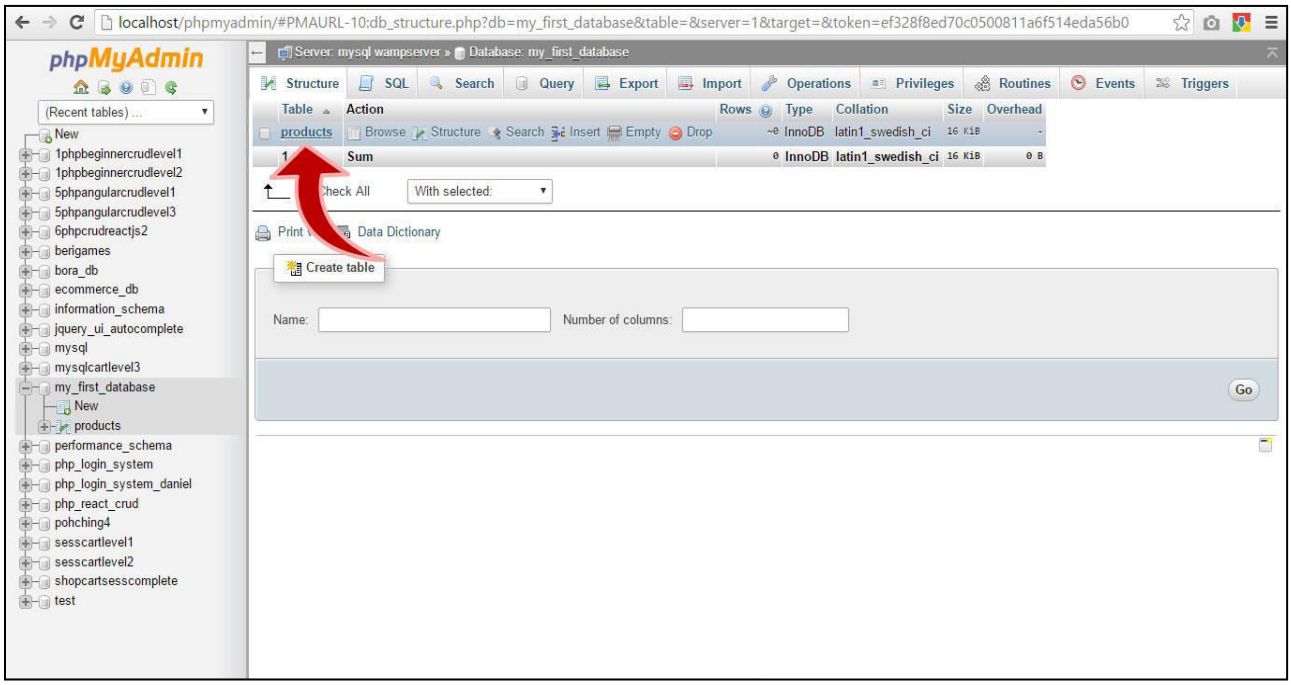


1. Fill out the fields with id, name, etc.
2. Mimic everything in the following image
3. Click the "**Save**" button

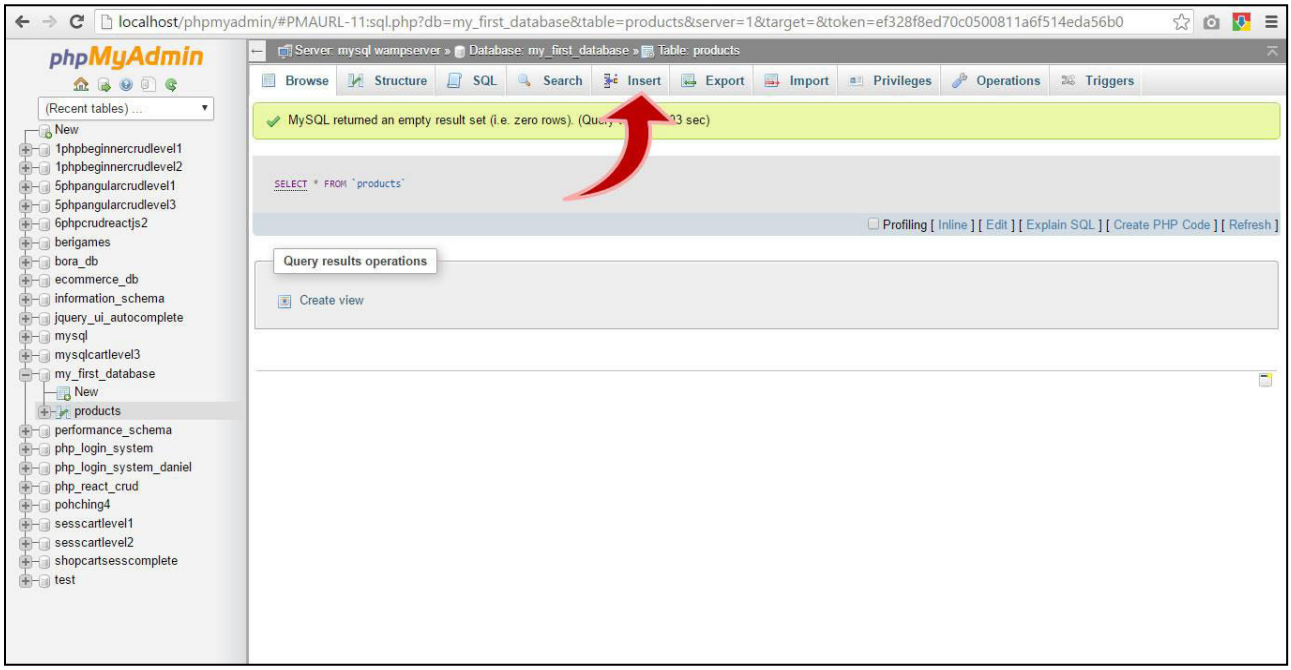


Insert Data

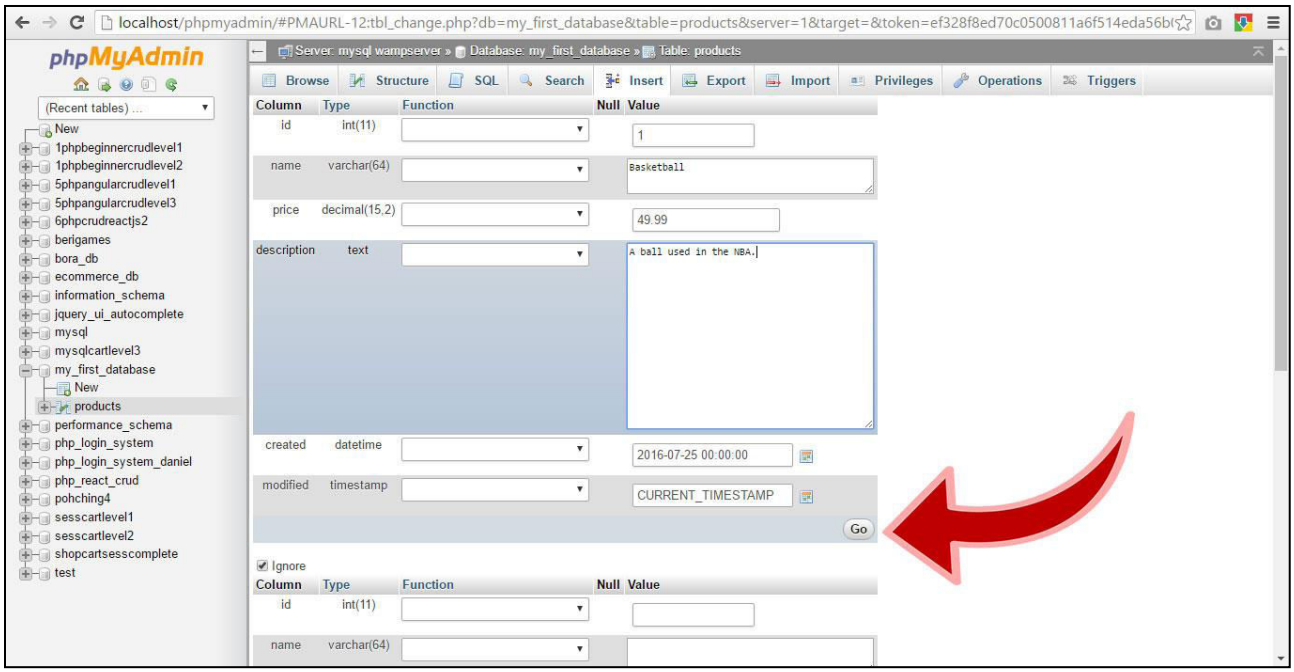
Click the "**products**" table.



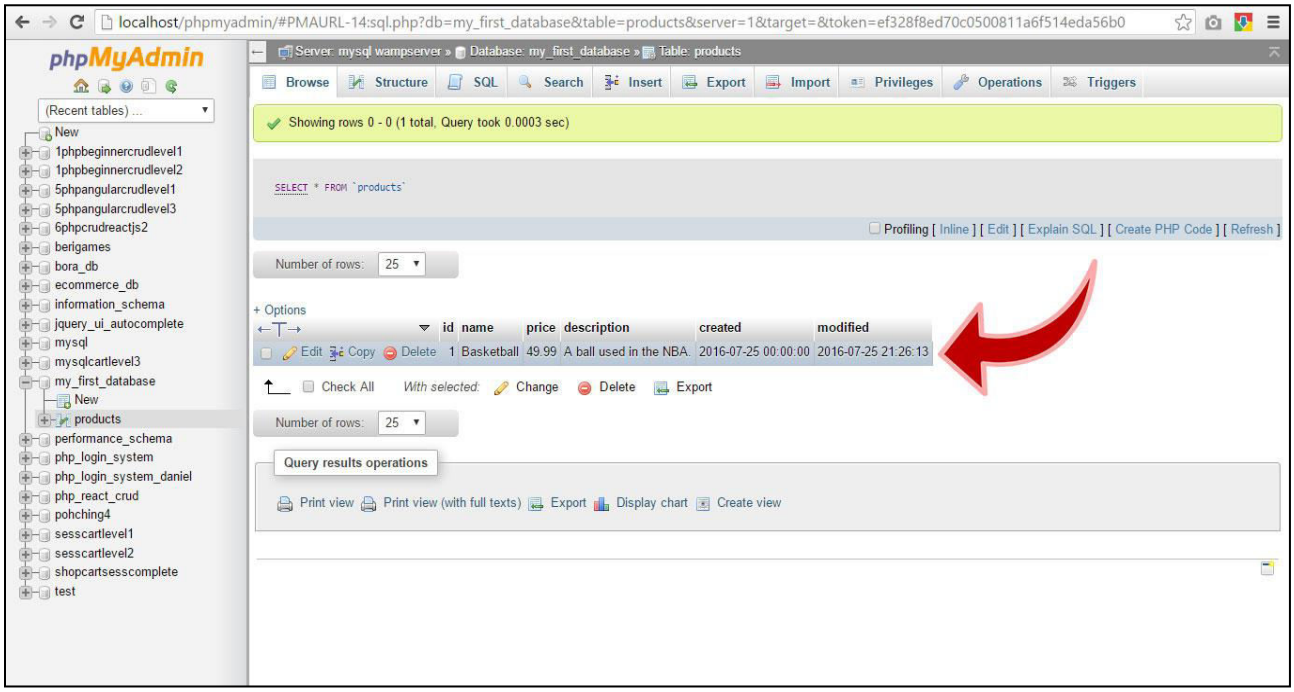
Click the "Insert" tab.



Fill out the form; mimic the data on the following image. Click the "**Go**" button.



We now have a database, a table inside the database and a record inside the table.



Similarly, make other entries in the table.

Steps to Run PHP script with database

1. Go to XAMPP server directory
2. Go to your "C:\xampp\htdocs\" directory
3. Create read_one.php, write code inside read_one.php and save
4. On your browser window, type http://localhost/read_one.php
5. See the output

PHP script that fetches one record from the MySQL using PDO:

```
<?php
// 1. database credentials
$host = "localhost";
$db_name = "my_first_database";
$username = "root";
$password = "";

// 2. connect to database
$con = new PDO("mysql:host={$host};dbname={$db_name}", $username, $password);

// 3. prepare select query
$query = "SELECT id, name, description, price FROM products WHERE id = ? LIMIT
0,1";
$stmt = $con->prepare( $query );

// 4. sample product ID
$product_id=1;

// 5. this is the first question mark in the query
$stmt->bindParam(1, $product_id);

// 6. execute our query
$stmt->execute();

// 7. store retrieved row to the 'row' variable
$row = $stmt->fetch(PDO::FETCH_ASSOC);

// 8. show data to user
echo "<div>Name: " . $row['name'] . "</div>";
echo "<div>Description: " . $row['description'] . "</div>";
echo "<div>Price: $" . $row['price'] . "</div>";
?>
```

Output

You should see the following output.

PHP script that fetches records from the MySQL using MySQLi Procedural:

(Note: First Create Database and Table)

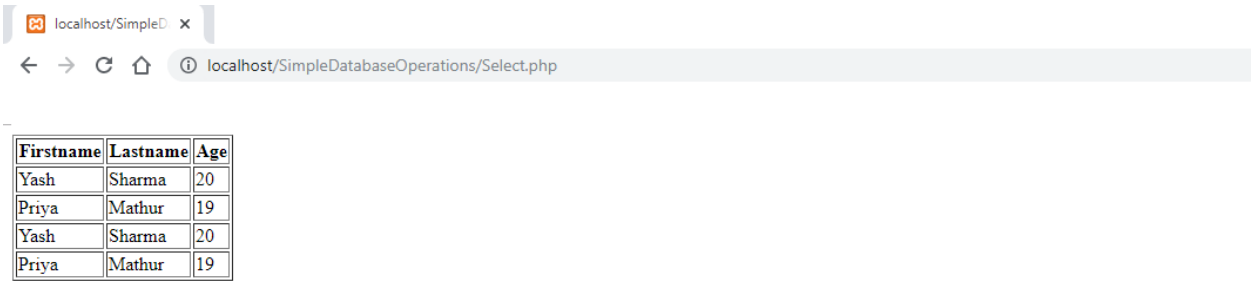
Database name is "Mydb" and Table name is "Data" with 3 columns as Firstname, Lastname and Age.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "Mydb";

// Create connection
$link = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$link) {
   die("Connection failed: " . mysqli_connect_error());
}
$sql = "SELECT * FROM Data";
if ($res = mysqli_query($link, $sql)) {
   if (mysqli_num_rows($res) > 0) {
      echo "<table border='1'>";
      echo "<tr>";
      echo "<th>Firstname</th>";
      echo "<th>Lastname</th>";
      echo "<th>Age</th>";
      echo "</tr>";
      while ($row = mysqli_fetch_array($res)) {
         echo "<tr>";
         echo "<td>".$row['Firstname']."</td>";
         echo "<td>".$row['Lastname']."</td>";
         echo "<td>".$row['Age']."</td>";
```

```
            echo "</tr>";
        }
        echo "</table>";
       mysqli_free_result($res);
    }
    else {
        echo "No matching records are found.";
    }
}
else {
    echo "ERROR: Could not able to execute $sql. ".mysqli_error($link);
}
mysqli_close($link);
?>
```

Output

You should see the following output.



Firstname	Lastname	Age
Yash	Sharma	20
Priya	Mathur	19
Yash	Sharma	20
Priya	Mathur	19

Conclusion:

Written and successfully executed Server Side Script in PHP to generate the web pages dynamically using the database connectivity.

WT Laboratory 7

Laboratory Continuous Assessment (LCA)[*Refer Rubric table in WTL manual]

Understanding of the Objective (5)	Performance (5)	Journal Submission and Ethics (Neatness, Handwriting, Timely submission) (5)	Orals (5)	Total (20)	Remarks	Instructor's Sign

Aim:

a) **Creating HTTP server in Node.js**
b) **Creating simple static file server**
c) **Using Express framework develop a website using Node JS and MySQL**

Objectives:

1. To understand difference between Node.js and PHP
2. To understanding how to choose correct java script framework for web development.
3. Create a simple web application using Node.js and MySQL.

Theory:

1. What is Node.js? Why it popular?
2. How to set up environment for Node.js?
3. Using Node as HTTP server and static File Server
4. Create simple CRUD application using Node.js and database

FAQ:

1. What is Full Stack Development? What are different technologies related to full stack development front end and back end? What are popular stacks?
2. How to choose Technology Stack for Web Application Development?
3. Compare Java script frameworks available for Web Development.

(Minimum 3 handwritten pages)

Note: Student is expected to attach this page as a title page of an assignment.

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36.

Definition

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

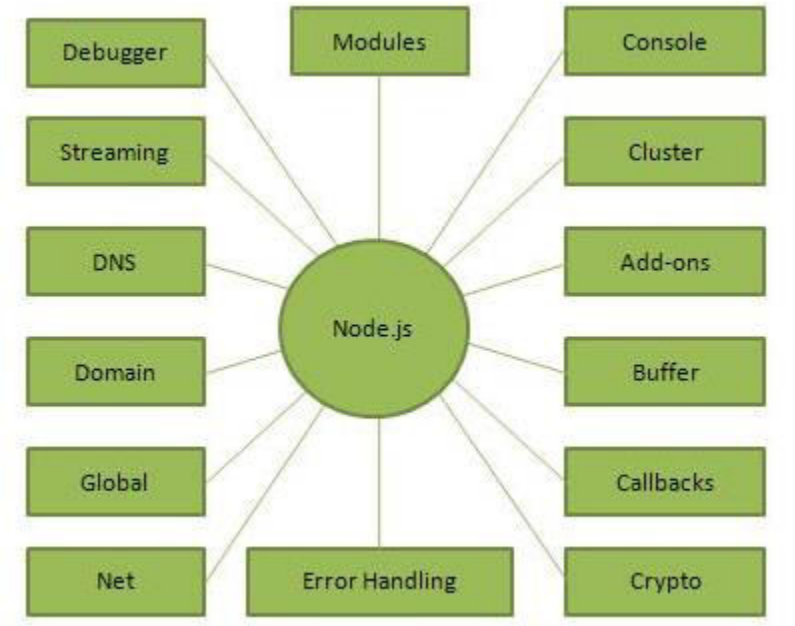
Following are some of the important features that make Node.js the first choice of software architects.

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
- **License** − Node.js is released under the MIT license

Who Uses Node.js?

Following is the link on github wiki containing an exhaustive list of projects, application and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, and Yammer to name a few.

Concepts



Why it is popular???

In a sequential language such as PHP, in order to get the HTML content of a page you would do the following:

```
$response = file_get_contents("http://example.com");
print_r($response);
```

In Node, you register some callbacks instead:

```
var http = require('http');

http.request({ hostname: 'example.com' }, function(res) {
  res.setEncoding('utf8');
  res.on('data', function(chunk) {
    console.log(chunk);
  });
}).end();
```

There are two big differences between the two implementations:

- Node allows you to perform other tasks while waiting to be notified when the response is available.
- The Node application is not buffering data into memory, but instead it's outputting it chunk-by-chunk.

Where to Use Node.js?

Following are the areas where Node.js is proving itself as a perfect technology partner.

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

For more information explore: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

Node.js application examples: https://magnetoitsolutions.com/blog/node-js-application-examples

Environment Setup:

- Node JS installed on your PC.
- Basic understanding of Node JS and Express JS.
- Knowledge of SQL, you should know and understand how to query a database.
- phpmyadmin installed on your PC. I recommend installing xampp as it already contains phpmyadmin in it.
- Understand how to use templating engines -- we are going to be using ejs in this tutorial).
- A text editor or IDE of your choice.

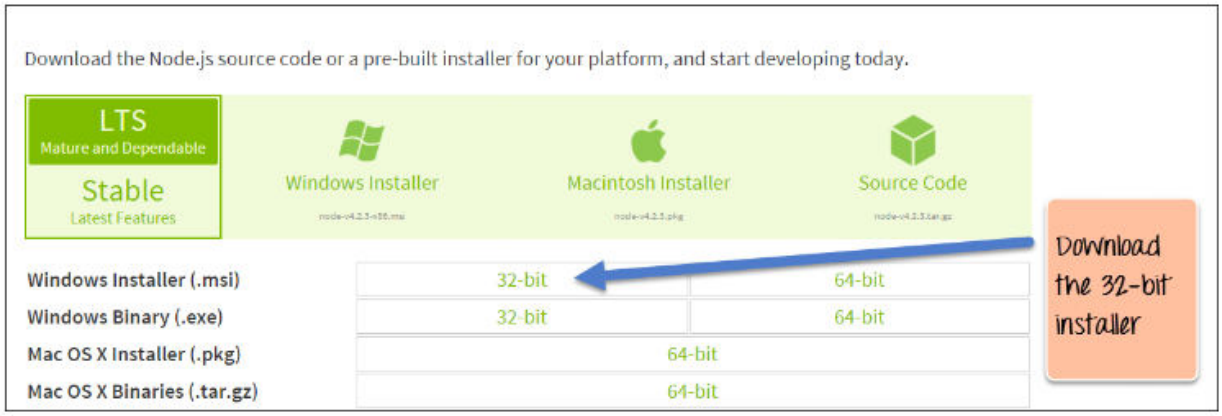
To start building your Node.js applications, the first step is the installation of the node.js framework. The Node.js framework is available for a variety of operating systems right from Windows to Ubuntu and OS X. Once the Node.js framework is installed, you can start building your first Node.js applications.

Node.js also has the ability to embedded external functionality or extended functionality by making use of custom modules. These modules have to be installed separately. An example of a module is the MongoDB module which allows you to work with MongoDB databases from your Node.js application.

How to install Node.js on Windows

The first steps in using Node.js is the installation of the Node.js libraries on the client system. To perform the installation of Node.js, perform the below steps;

Go to the site https://nodejs.org/en/download/ and download the necessary binary files. In our example, we are going to download the 32-bit / 64-bit setup files for Node.js.



Follow defaults steps and finish installation.

Installing NPM (Node Package Manager) on Windows

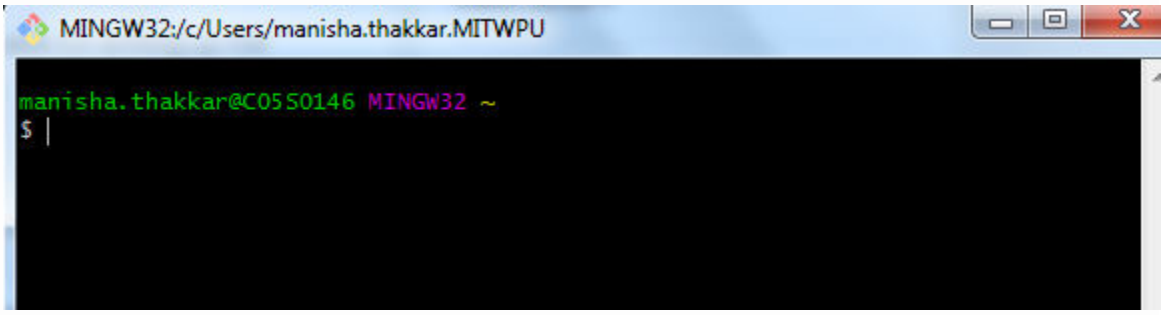
NPM is a "package manager" that makes installing Node "packages" fast and easy. A package is just a code library that extends Node by adding useful features. For example, the "request" package simplifies the process of making HTTP requests so you can easily get web resources from other sites.

NPM is installed when you install Node.js®

Prerequisites

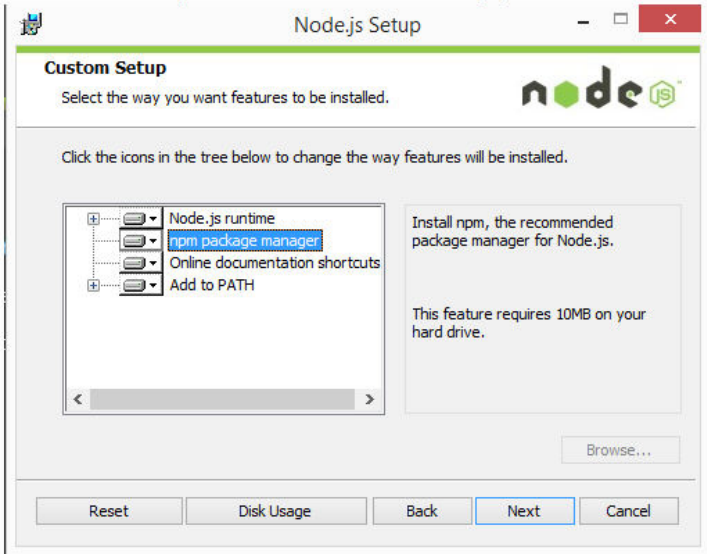
You should have some familiarity with an application that lets you issue command line instructions. For example, the Windows Command Prompt, PowerShell, Cygwin, or the Git shell (which you get when you install Github for Windows).

- Follow **the prompts in the installer**
    1. First of all download the Git Bash



Main difference between command prompt and the git bash is: In command prompt we use back slashes (\) but in GitBash we use forward slashes (/)

Follow the prompts in the installer (Accept the license agreement, click the NEXT button a bunch of times and accept the default installation settings).

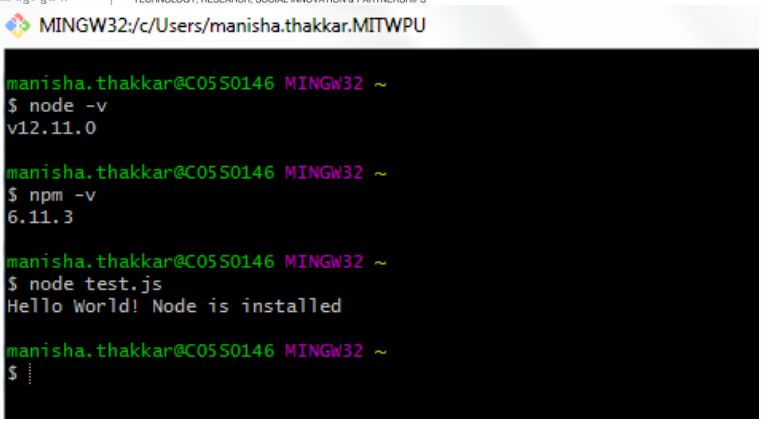


- Restart **your computer.** You won't be able to run Node.js until you restart your computer.

Test it!

Make sure you have Node and NPM installed by running simple commands to see what version of each is installed:

- **Test Node.** To see if Node is installed, open the Windows Command Prompt, Powershell or a similar command line tool, and type `node -v`. This should print the version number so you'll see something like this `v0.10.35`.
- **Test NPM.** To see if NPM is installed, type `npm -v` in Terminal. This should print the version number so you'll see something like this `1.4.28`
- **Create a test file and run it.** A simple way to test that node.js works is to create a simple JavaScript file: name it hello.js, and just add the code `console.log('Node is installed!');`. To run the code simply open your command line program, navigate to the folder where you save the file and type `node hello.js`. This will start Node.js and run the code in the `hello.js` file. You should see the output `Node is installed!`.


MINGW32:/c/Users/manisha.thakkar.MITWPU
manisha.thakkar@C05S0146 MINGW32 ~
$ node -v
v12.11.0
manisha.thakkar@C05S0146 MINGW32 ~
$ npm -v
6.11.3
manisha.thakkar@C05S0146 MINGW32 ~
$ node test.js
Hello World! Node is installed
manisha.thakkar@C05S0146 MINGW32 ~
$

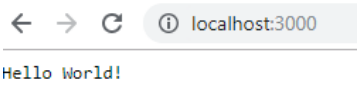
Creating HTTP server in Node.js

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the `createServer()` method to create an HTTP server:

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(3000); //the server object listens on port 8080
```

Output:

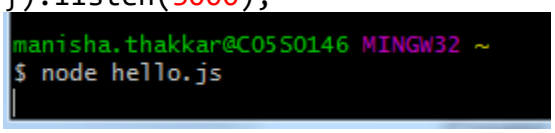


Creating simple static file server (hello.html)

```
<html>
<body>
<h1>Header: Hello World!</h1>
<p>Paragraph: Hi There!</p>
</body>
</html>
```

Create a Node.js file that reads the HTML file, and return the content: (hello.js)

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('hello.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(3000);
```

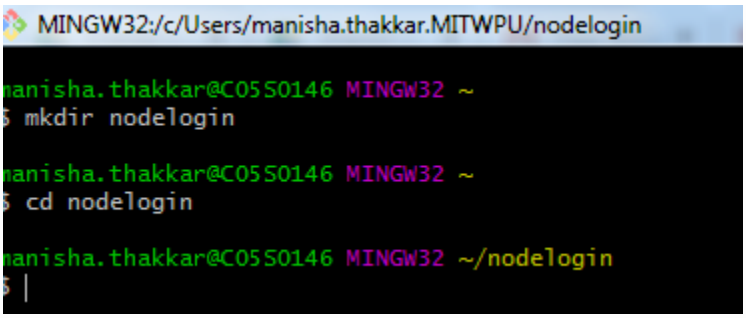


Using Express framework develop a website using Node JS and MySQL

Folder Structure: (*** Students are required to create other application using node.js and complete CRUD operations**)

```
\-- nodelogin
  |-- login.html
  |-- login.js
```

Creating the directory and change path to Project directory



Initialize the Project

Run the command: **npm init** from inside the directory, it will prompt us to enter a package name, enter: **login**.

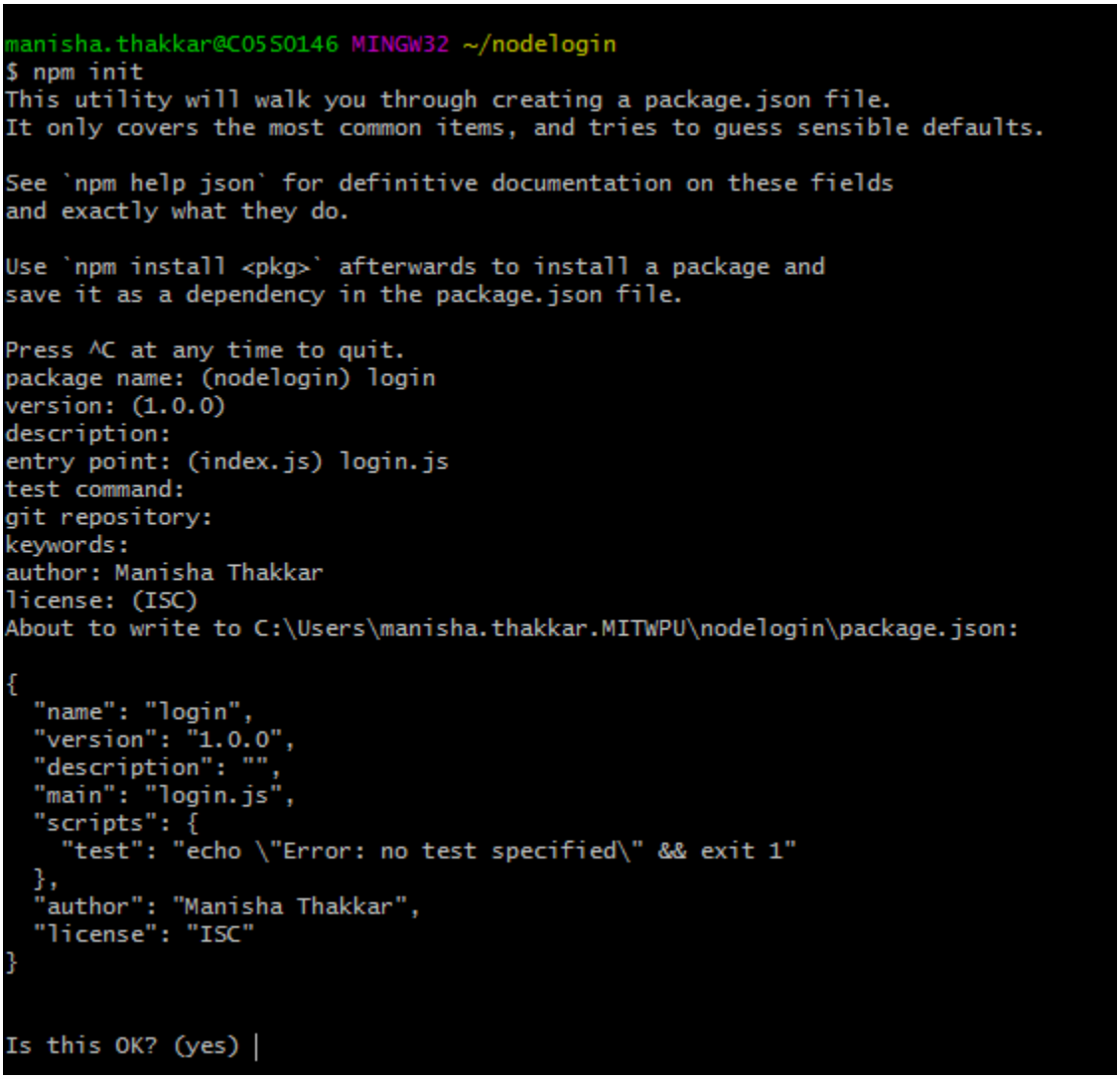
When it prompts to enter the entry point enter **login.js**.

Now we need to install the packages listed in the requirements, while still in the command line run the commands listed in the requirements above.

We should now have a new directory called: **node_modules** with all the modules installed

Run the command: npm init from inside the directory, it will prompt us to enter a package name, enter: login.

$ npm init



Install required modules

The following modules are going to be needed to successfully build the app.

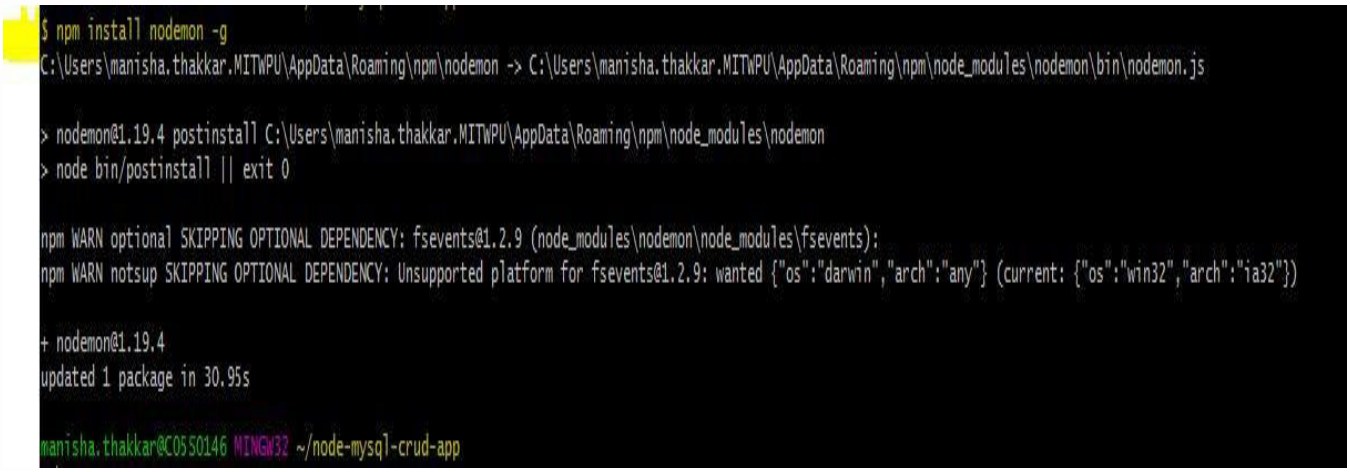
- MySQL Server
- Node.js
- Express - Install with command: **npm install express**.
- Express Sessions - Install with command: **npm install express-session**.
- MySQL for Node.js - Install with command: **npm install mysql**.

Then type the following command to install the last module globally on your PC.

What is Nodemon?

You started your server and it's finally up and running. You make a change or two, save the file, and open up your browser again. Something is wrong. It didn't reload and now you find yourself manually restarting the server every time. This is where Nodemon comes in.

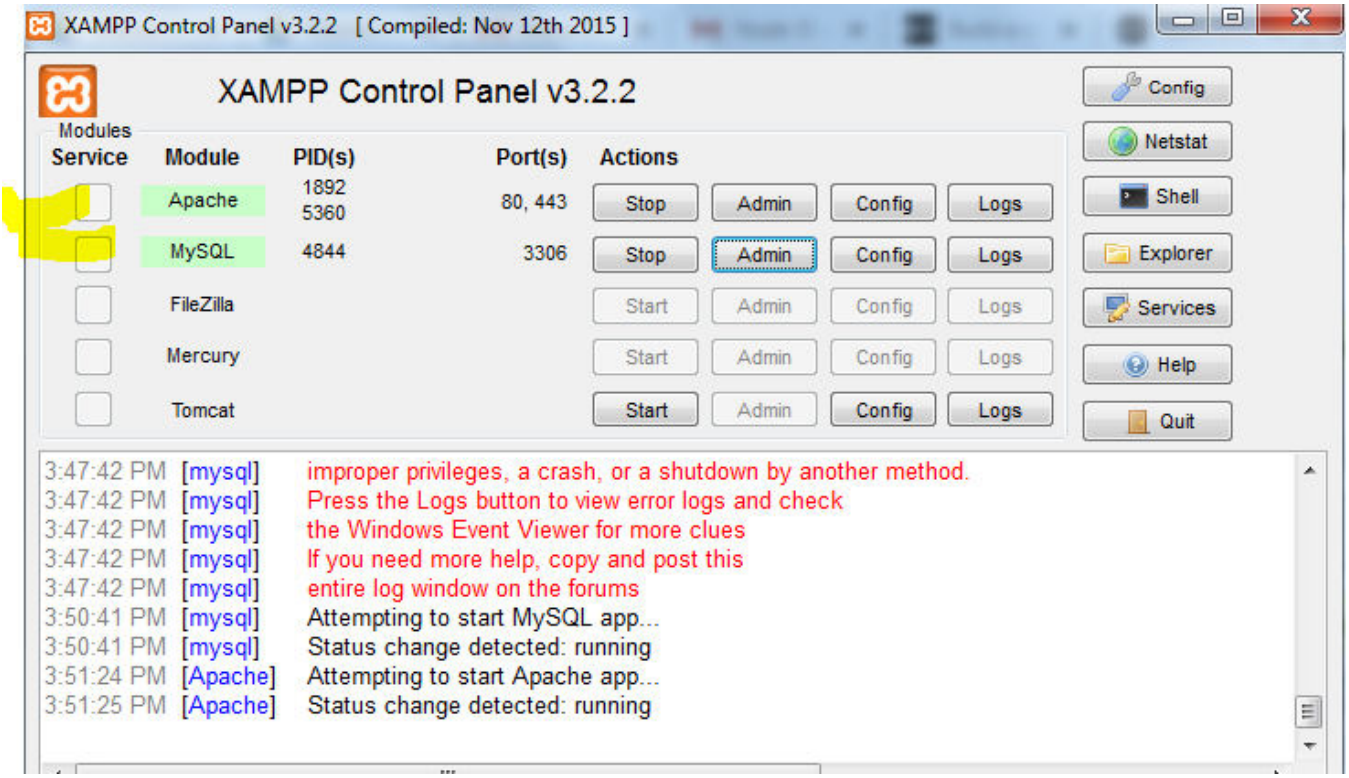
Nodemon is a development dependency that **monitors for any changes** in your Node.js application and **automatically restarts the server**, saving time and tedious work.



Tutorial is available here : https://codeshack.io/basic-login-system-nodejs-express-mysql/

Creating the database for the app

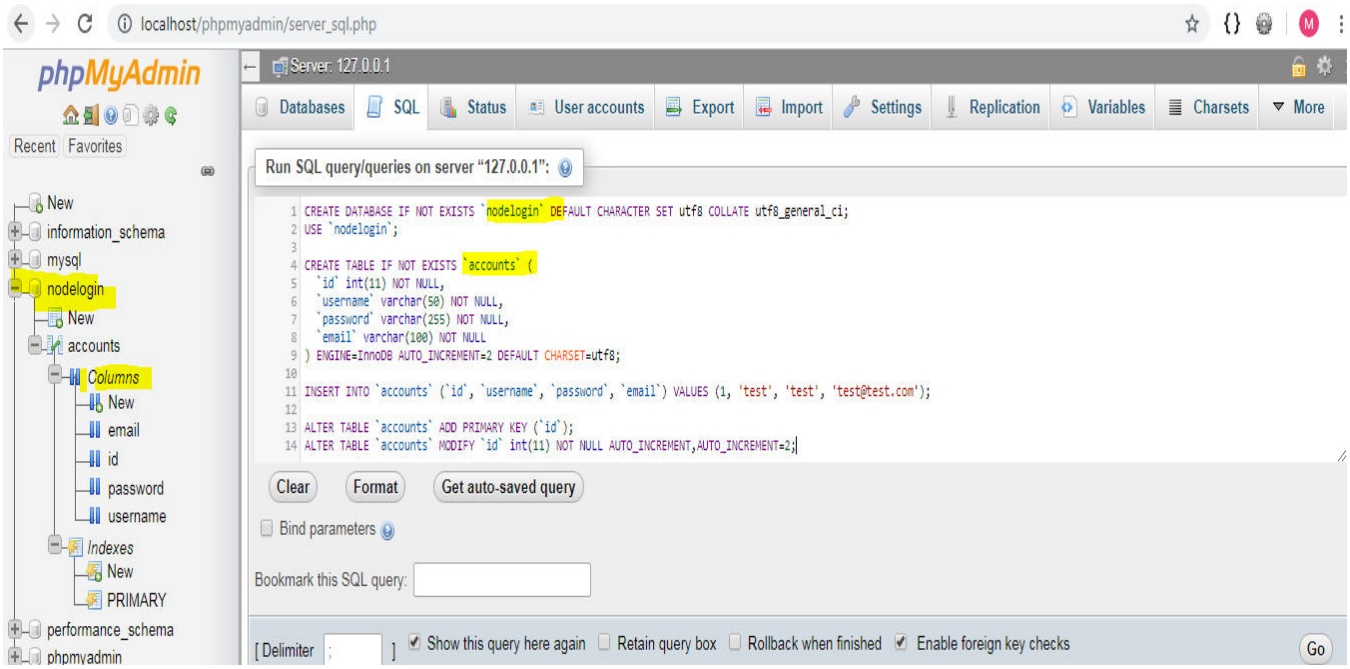
Copy the command below and navigate to your phpmyadmin dashboard and execute the following query in the console (usually found at the bottom of the page) in order to create database and table for the app.



Open MySQL admin tab

```
CREATE DATABASE IF NOT EXISTS `nodelogin` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `nodelogin`;

CREATE TABLE IF NOT EXISTS `accounts` (
 `id` int(11) NOT NULL,
 `username` varchar(50) NOT NULL,
 `password` varchar(255) NOT NULL,
```

```
 `email` varchar(100) NOT NULL
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

INSERT INTO `accounts` (`id`, `username`, `password`, `email`) VALUES (1, 'test', 'test', 'test@test.com');

ALTER TABLE `accounts` ADD PRIMARY KEY (`id`);
ALTER TABLE `accounts` MODIFY `id` int(11) NOT NULL AUTO_INCREMENT,AUTO_INCREMENT=2;
```



Adding the views

Creating the Login System

login.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Login Form Tutorial</title>
		<style>
		.login-form {
			width: 300px;
			margin: 0 auto;
			font-family: Tahoma, Geneva, sans-serif;
		}
		.login-form h1 {
			text-align: center;
			color: #4d4d4d;
			font-size: 24px;
			padding: 20px 0 20px 0;
		}
		.login-form input[type="password"],
		.login-form input[type="text"] {
			width: 100%;
			padding: 15px;
			border: 1px solid #dddddd;
			margin-bottom: 15px;
			box-sizing:border-box;
		}
		.login-form input[type="submit"] {
			width: 100%;
			padding: 15px;
			background-color: #535b63;
```

```
                    border: 0;
                    box-sizing: border-box;
                    cursor: pointer;
                    font-weight: bold;
                    color: #ffffff;
                }
                </style>
        </head>
        <body>
                <div class="login-form">
                    <h1>Login Form</h1>
                    <form action="auth" method="POST">
                        <input type="text" name="username" placeholder="Username" required>
                        <input type="password" name="password" placeholder="Password" required>
                        <input type="submit">
                    </form>
                </div>
        </body>
</html>
```

login.js Source

```
var mysql = require('mysql');
var express = require('express');
var session = require('express-session');
var bodyParser = require('body-parser');
var path = require('path');

var connection = mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : '',
```

```
        database : 'nodelogin'
});

var app = express();
app.use(session({
        secret: 'secret',
        resave: true,
        saveUninitialized: true
}));
app.use(bodyParser.urlencoded({extended : true}));
app.use(bodyParser.json());

app.get('/', function(request, response) {
        response.sendFile(path.join(__dirname + '/login.html'));
});

app.post('/auth', function(request, response) {
        var username = request.body.username;
        var password = request.body.password;
        if (username && password) {
                connection.query('SELECT * FROM accounts WHERE username = ? AND password = ?', [username, password], function(error, results, fields) {
                        if (results.length > 0) {
                                request.session.loggedin = true;
                                request.session.username = username;
                                response.redirect('/home');
                        } else {
                                response.send('Incorrect Username and/or Password!');
                        }
                        response.end();
                });
        } else {
                response.send('Please enter Username and Password!');
                response.end();
```

```
        }
});

app.get('/home', function(request, response) {
        if (request.session.loggedin) {
                response.send('Welcome back, ' + request.session.username + '!');
        } else {
                response.send('Please login to view this page!');
        }
        response.end();
});

app.listen(3000);
```

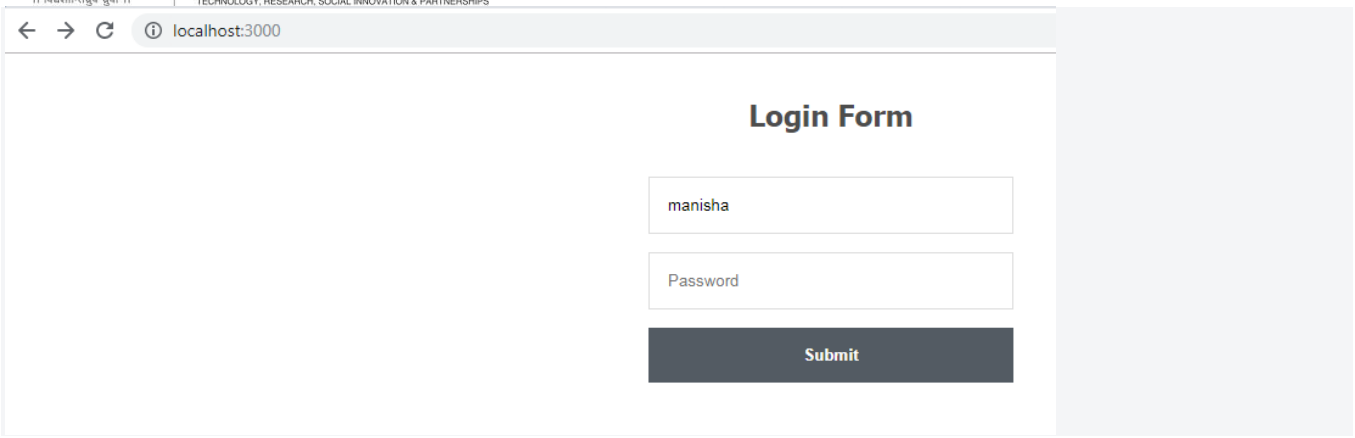
Our web application needs to listen on a port, for testing purposes we'll use port 3000:

```
app.listen(3000);
```

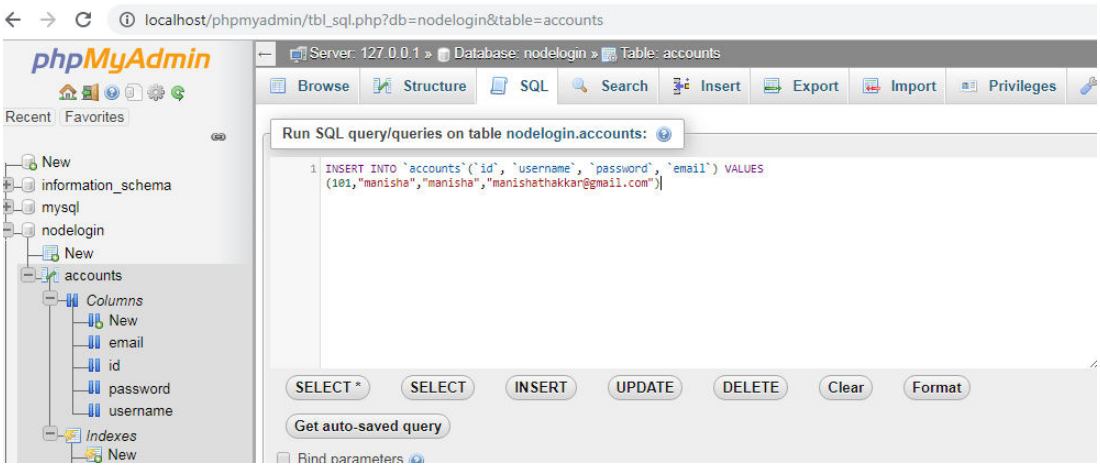
To run our new web application we can run the following command:

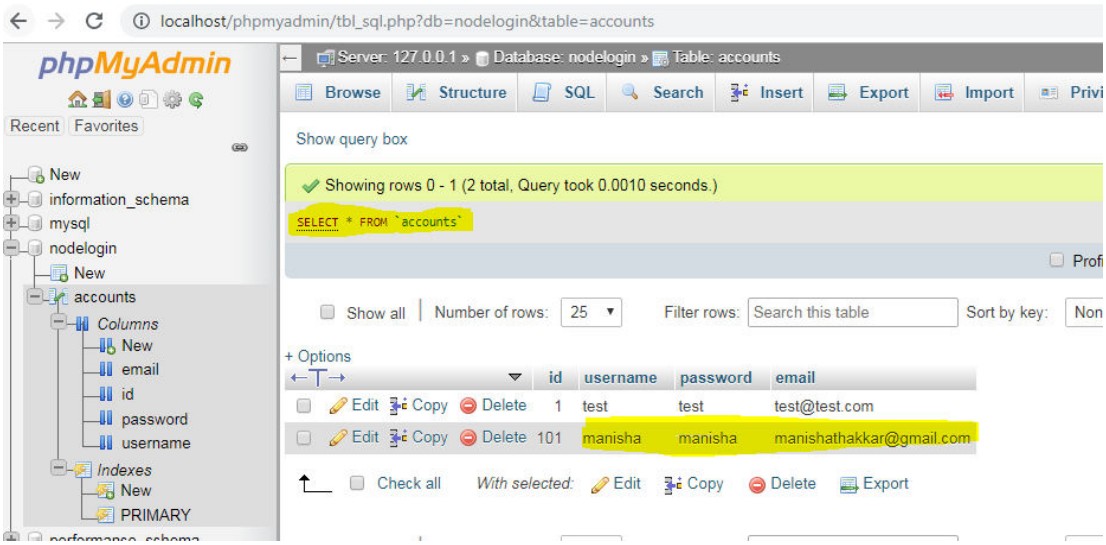
node login.js in command prompt/console, this will start the server, if we enter the address: **http://localhost:3000/** it should display our login form.

```
manisha.thakkar@C0550146 MINGW32 ~/nodelogin
$ node login.js
Error: ENOENT: no such file or directory, stat 'C:\Users\manisha.thakkar.MITWPU\
nodelogin\login.html'
```

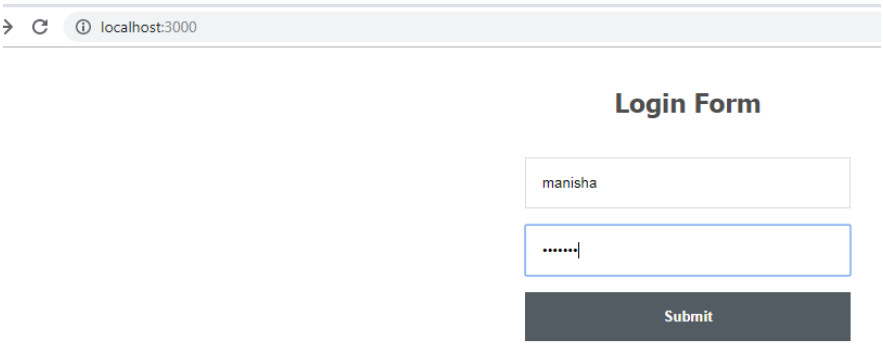

localhost:3000
Login Form
manisha
Password
Submit

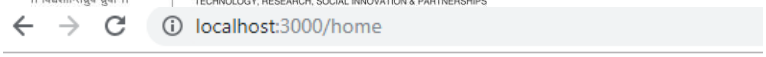
Insert row in table accounts





Login with new user name and password





Welcome back, manisha!