UNIT -I

1. **HTML:**
   - 1.1 Tables
   - 1.2 Basic Text Markup
   - 1.3 Images
   - 1.4 Lists
   - 1.5 Forms
   - 1.6 Frames
2. **XML:**
   - 2.1 The Introduction to XML
   - 2.2 The syntax of XML
   - 2.3 XML Document Structure
   - 2.4 Document type Definition
3. **JAVASCRIPT:**
   - 3.1 Introduction to JavaScript
   - 3.2 Objects in Javascript

1. HTML

- HTML stands for Hyper Text Markup Language.
- Markup is just something which we add to a document to give it special meaning .
  - Ex: High Lighter pen is used to Markup the document
- This Markup allows a web browser to display your document appropriately.
- HTML is a Documentation Language to create Web Document.
- HTML is a Universal Language to design a "static web pages".
- HTML is a Machine Independent Language, and that all browsers accept the HTML code.

1.1 Tables

- The Table is one of the most useful HTML constructs.
- Tables have 2 uses:
  - Structuring pieces of information .
  - Structuring the whole web page.
- **A Table is a grid of information like ledger or spreadsheet.**
- The <table> tag defines an html table.
- An html table consists of the <table> element and one or more <tr>, <th>, <td> elements.

  The <tr> element defines a table row.
  The <th> element defines a table header.
  The <td> element defines a table cell.

Syntax of <table> tag:

```
<table [align="center"/"right"/"left"]
       [border="n"] [cellpadding="n"]
       [width="nn%"] [cellspacing="n"]
       [bgcolor="colorname"]>
</table>
```

Attribute name	Description
align	Specifies alignment of a table
border	Specifies the table border
cellpadding	Specifies the space between the cell and the cell content
cellspacing	Specifies the space between cells

width	Specifies the width of the table

<tr> tag: <tr> used to create row in the html

Syntax:

```
<tr [align="center"/"right"/"left"]
    [valign="top"/"center"/"bottom"]
    [bgcolor="colorname"]>
</tr>
```

Attribute name	Description
Align	aligns the content in a table row.
valign	Vertical aligns the content in a table row.
Bgcolor	Specifies a background color for a table row

<th> tag: To create table header.

Syntax:

```
<th [align="center"/"right"/"left"]
    [valign="top"/"center"/"bottom"]
    [now rap][colspan="n"][rowspan="n"]
    [bgcolor="colorname"]>
</th>
```

Attribute name	Description
Align	aligns the content alignement in a header cell.
valign	Vertical aligns the content in a header cell.
Bgcolor	Specifies a background color for a table row
Nowrap	Specifies that the content inside a header cell should not wrap
Colspan	Specifies the no. Of columns a header cell should span
Rowspan	Specifies the no. Of rows a header cell should span

<td> tag: To create table cell

Syntax:

```
<td [align="center"/"right"/"left"]
    [colspan="n"][rowspan="n"]
    [bgcolor="colorname"]>
</td>
```

- The text in <th> elements are BOLD and centered by default.
- The text in <td> elements are Regular and left-aligned by default.

Example:

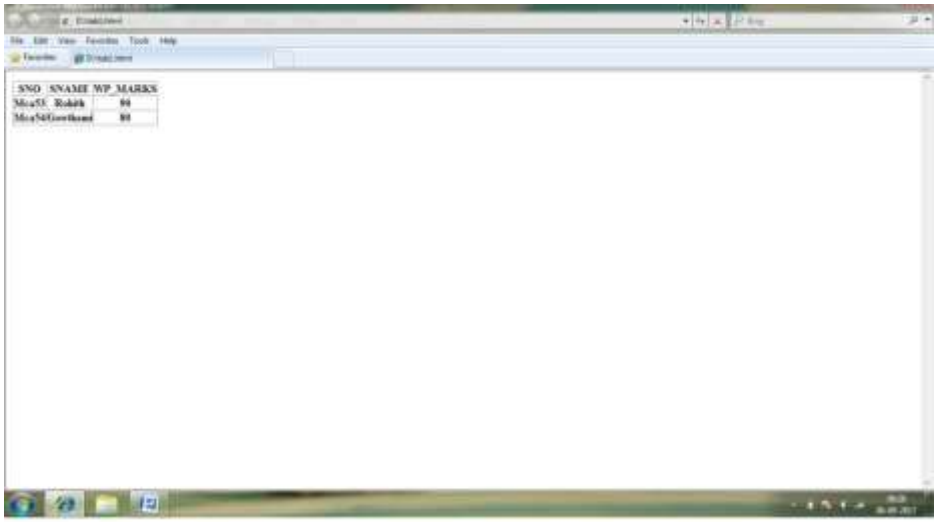
Design following format using Table tags.

SNO	SNAME	WP_MARKS
Mca53	Rohith	90
Mca54	Gowthami	80

```
<html>
    <head></head>
    <body>
        <table border="2"    width="50%" align="center"
          Cellpadding="2"    cellspacing="4" bgcolor="pink">
        <tr>
```

```
                <th>SNO</th>
                <th>SNAME</th>
                <th>WP_MARKS</th>
            </tr>
            <tr>
                <th>Mca53</th>
                <th>Rohith</th>
                <th>90</th>
            </tr>
            <tr>
                <th>Mca54</th>
                <th>Gowthami</th>
                <th>80</th>
            </tr>
            </table>
        </body>
</html>
```

OUTPUT:



1.2 Basic Text Markup

1.2.1) Paragraphs
1.2.2) Line Breaks
1.2.3) Headings
1.2.4) Block Quotations
1.2.5) Font Styles and Sizes
1.2.6) Character Entities
1.2.7) Horizontal Rules

1.2.1) Paragraphs

- Paragraph element offers a way to structure the text in the web page.
- Each paragraph of text should go in between an opening and closing tag.
- General form of paragraph tag
  **<p [align="left"/"right"/"center"]> ..... **
- when a browser displays a paragraph it usually inserts a new line before the next paragraph and adds a little bit of extra virtual space.

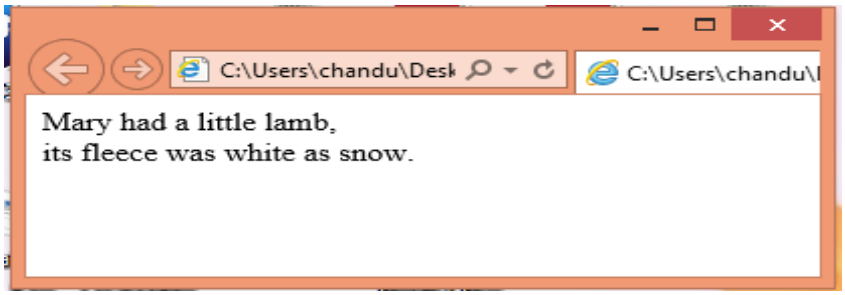
- Each paragraph can be aligned on the screen either to the left, the right or centered.
- Html processors ignore all white spaces in source documents except for spacing words.
- That is tabs, newlines, paragraphs are not formatted as we expect. In fact all these that are encountered in the source code get converted into a single space characters.
- Any space that are placed between the words will get converted into a single space in the displayed document.
- Escape sequences are used to display more than one space.

1.2.2) Line Breaks

The HTML br element produces a line break in text.The br tag differs from the p tag in that it can have no conte
nt and therefore has no closing tag. so br tag is empty tag. The break tag is specified as <br />. The slash indicates that the tag is both an opening and closing tag. The space before slash represent the absent content.

Eg :
```
<html>
        <head></head>
        <body>
            <p>Mary had a little lamb, </br> its fleece was white as snow.</p>
        </body>
</html>
```

OUTPUT:



1.2.3) Headings:

- Headings also helps to structure the document.
- Html offers six levels of headings which use the elements <h1>, <h2>, <h3>, <h4>, <h5>, <h6>.
- While browsers can display headings differently, they tend to display the <h1> element as the largest of the size and <h6> as the smallest
- General form:
  <h1 [align="left"/"right"/"center"]>..... </h1>

Eg:
```
<html>
        <head></head>
        <body>
            <h1>Aidan's Airplanes (h1)</h1>
            <h2>The best is used Airplanes (h2)</h2>
            <h3>we've got them by the hangar ful (h3)</h3>
            <h4>we're the guys to see for a good used airplane (h4)</h4>
            <h5>we offer great prices on great planes(h5)</h5>
            <h6> Noreturns, no guarantees, no refunds, all sales are final(h6)</h6>
        </body>
</html>
```

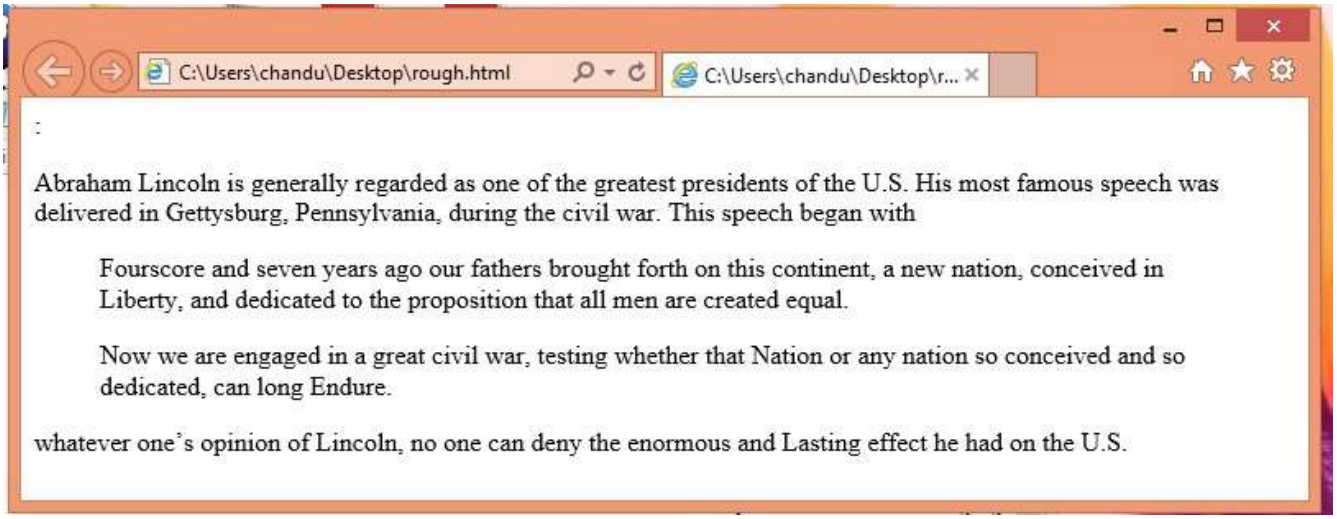
OUTPUT:

1.2.4) Block Quotations

- Sometimes we want a block of text to be set off from the normal flow of text in a document. In many cases, such a block is a long quotation.
- The <blockquote> tag is designed for this situation. Browser designers determine how the content of <blockquote> can be made to look different from the surrounding text.
- In many cases, the block of text is indented, either on the left or right side or both.
- Another possibility is that the block is set in italic.

Eg:
```
<html>
    <head></head>
    <body>
        <p>Abraham Lincoln is generally regarded as one of the greatest presidents of
           the U.S. His most famous speech was delivered in Gettysburg,
           Pennsylvania, during the civil war.  This speech began with
        </p>
        <blockquote>
            <p>Fourscore and seven years ago our fathers brought forth on this
               continent, a new nation, conceived in Liberty, and dedicated
               to the proposition that all men are created equal.
            </p>
            <p>Now we are engaged in a great civil war, testing whether that
               Nation or any nation so conceived and so dedicated, can long
               Endure.
            </p>
        </blockquote>
        <p>whatever one's opinion of Lincoln, no one can deny the enormous and
           Lasting effect he had on the U.S.
        </p>
    </body>
</html>
```

OUTPUT:



1.2.5) Font Styles and Sizes:

- Frequently, one needs to set off or emphasize words or phrases in text. This is usually done by using different font styles or different font sizes for the text that is to be emphasized.
- XHTML included a few tags for doing this. The simplest of these are <b> and <i>>, which change the font style of the text in their content to bold face and italic, respectively.
- The text on an HTML page can be altered in a number of ways.
- Ex: The actual font used can be changed to attempt to force the browser to use a specific font.
- <basefont size=n">
- Helps to specify minimum font size for basic text. The size argument takes an integer from 1 to 7.
- <font size= "[+ / -]" color=" #rrggbb" face ="font name">
- Sets the font size relative to either the default value or to any size set by <basefont>
- Absolute font sizes can be forced by using an integer from 1 to 7.
- Relative font sizes are set by using + / - 1 to 7.
- The color of the text is set with the help of color attribute. This takes a hexadecimal value which represents the amount of red, green, and blue in the chosen color.

Ex: Font variations

```
<html>
    <head></head>
    <body>
        <h1> Font Variations</h1>
        <p> we can use <b> simple </b> tags to <i> change </i> the appearance of
            <strong>text</strong> within <tt>webpages</tt>. Even super<sup>
            Script </sup> and sub <sub>scripts</sub> are <em>supported</em>
        </p>
    </body>
</html>
```

OUTPUT:

1.2.6) Character Entities:

- XHTML provides a collection of special characters that are something needed in a document but cannot be typed as themselves. In some cases, these characters are used in XHTML in some special way,
- For example >, <, and & . in other cases, the characters do not appear a keyboard, such as the small raised circle that represents “degrees” in a reference to temperature.
- These special characters are defined as entities, which are names for the characters by the browser.
- An entity in a document is replaced by its associated character by the browser.
- The following are some of the most used entities.

Entity	Meaning
&amp;	Ampersand
	for single space
&lt;	Lessthan
&gt;	Greaterthan
&quot;	Quotation marks

Ex:

```
<html>
        <head></head>
        <body>
                <h1>character entites</h1>
                <p> &quot; HTML &quot; tags are enclosed in &lt; and &gt; symbols
                    For example  &lt; html &gt;</p>
        </body>
</html>
```

OUTPUT:

1.2.7) Horizontal Rules:

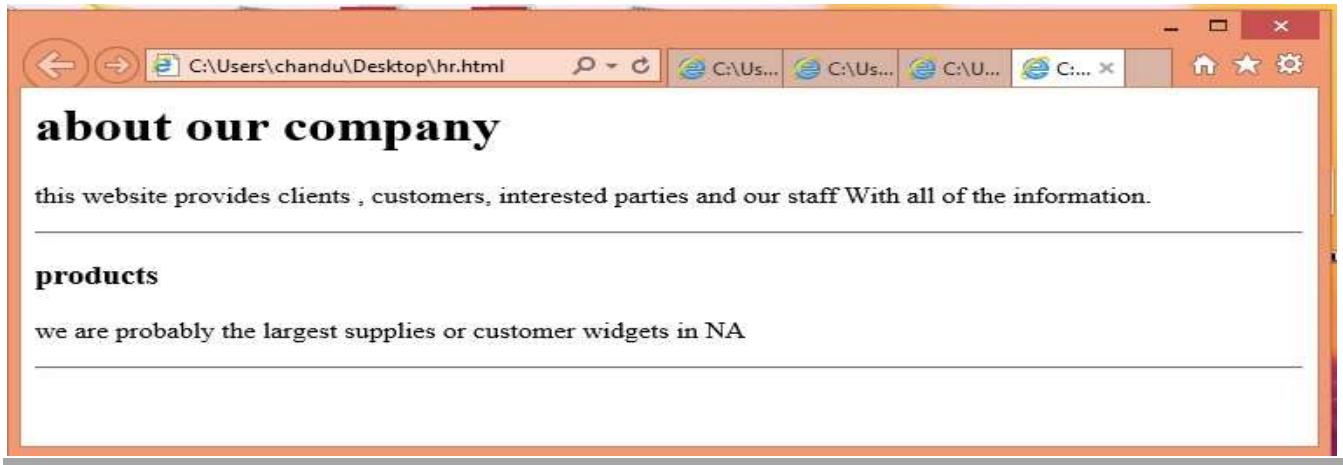
- A horizontal rule can be created using the <hr> tag.

- The <hr> element creates a horizontal rule across the page.
- This is frequently used to separate distinct sectors of a page where a new heading is not appropriate.
- General form of hr tag is:

    - <hr [align=”left”/”right”/”center”][size=”n”]
    - [width = “nn%”]>
- The size option specifies the thickness of the rule in pixels.
- The width of the line is best given as a percentage of the available screen size.

Ex:
```
<html>
    <head></head>
    <body>
        <h1>about our company</h1>
        <p>this website provides clients , customers, interested parties and our staff
            With all of the information.</p>
        <hr>
        <h3>products</h3>
        <p>we are probably the largest supplies or customer widgets in NA</p>
        <hr width=”50%”>
    </body>
</html>
```

OUTPUT:



1.3 Images

Images:

- The inclusion of images in a document can dramatically enhance is appearance although images slow the document-download process considerably for clients that do not have high-speed internet access.
- The image is stored in a file, which is specified by an XHTML request .
- The image in the file is inserted into the display of the document by the browser.

Image formats:

- The two most common methods of representing images are:
- Graphic Interchange Format(GIF-pronounced as jif-fy)
- Joint Photographic Experts Group(JPEG, pronounced as jay-peg)
- Most contemporary browsers can render images in either of these two formats.
- Files in both of these formats are compressed to reduce storage needs and provide faster transfer over the internet.

<img/>:

- The image tag, <img/> which is an inline tag, specifies an image that is to appear in a document. In its simplest form, the image tag includes two attributes:

- Two optional attributes of img, width and height , can be included to specify (in pixels) the size of the rectangle for the image.
- These can be used to scale the size of the image (that is, to make it larger or smaller).
- Care must be taken to ensure that the image is not distorted in the resizing.
- For example, if the images is square , the width and height attribute values must be equal.
- The following is an example of an image element:

<img src =”c210.jpg” alt=”(picture of a Cessna 210)” / >

- Src(source) which specifies the file containing the image;
- And alt which specifies text to be displayed when it is not possible to display the image.

Ex:
```
<html>
    <head></head>
    <body>
        <p>inserting image from my folder</p>
        <img src =”pic.jpg”>
    </body>
</html>
```
OUTPUT:

1.4 Lists

- One of the most effective way of structuring a website or its contents is to use as ***List***.
- In addition to the headings and paragraphs ***Lists*** help to organize and present information for easy reading.
- Itemized lists helps to highlight important points.

Html provides 3 types of lists:

1. Unordered or Bulleted list
2. Ordered or Numbered list
3. Definition List

The ordered and unordered lists are each made up of sets of list item **<li>**

1. **Unordered List:**
   - The basic unordered list has a bullet in front of each list item.
   - Every thing between the tags must be encapsulated within -----
tags.
   - Each item in the list must encoded between --

Syntax: <ul [type= “disc”/”square”/”circle”]
**[compact]>--------
**

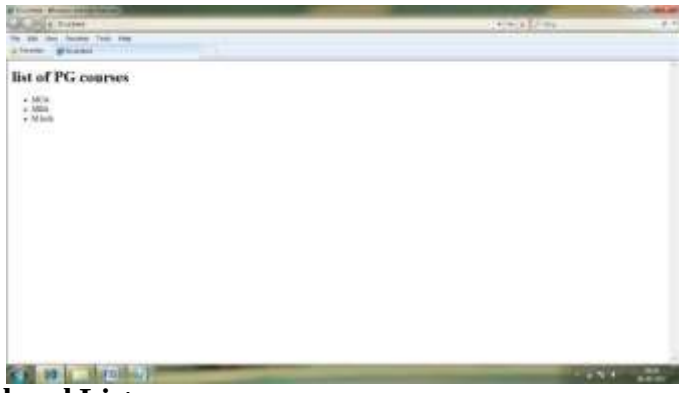
   - Type attribute sets the style of a list item.
   - The bullet can be disc or square or circle
     - Disc: causes each bullet to appear as a solid round disc.
     - Square: causes each bullet to appear as a solid square.
     - Circle: causes each bullet to appear as an empty circle.
   - Component attribute is used to minimize the amount of space that a list uses.
   - Disc is the default bullet on some browsers and circle on some browsers.
   - Type attribute will not work if it is written in upper case letters.
   - Unordered lists may be nested inside same or other type of lists.

- A line space automatically is inserted before and after an unordered list except for a list nested written another list.

Ex:

```
<html>
    <head></head>
    <body>
        <h1>list of PG courses</h1>
        <ul type = “disc”>
            <li>MCA</li>
            <li>MBA</li>
            <li>M.tech</li>
        </ul>
    </body>
</html>
```

OUTPUT:



2. Ordered List:

- An ordered list has a number instead of a bullet in front of each list item.
- Ordered lists are used to define a list whose items are in sequential, numerical order.
- General form:
    - Syntax: <ol [type= “1”/”i”/”a”/”A”]
        - [start=”n”][compact]>-------
- Type attribute sets the style of a list item ;
    - “I” causes the items to be numbered as I, II, III.....
    - “A” causes the items to be numbered as A, B, C.....
    - “a” causes the items to be numbered as a, b, c.....
    - “1” (default) causes the items to be numbered as 1, 2, 3.....
    - “i” causes the items to be numbered as i, ii, iii.....
- Compact attribute to minimize the amount of space that a list uses.
- Start attribute specifies the starting number of the first item in an ordered list.
- The default starting number for type “1” is 1

Ex:

```
<html>
    <head></head>
    <body>
        <h1>list of PG courses</h1>
        <ol type = “1”>
            <li>MCA</li>
            <li>MBA</li>
            <li>M.tech</li>
        </ol>
    </body>
</html>
```

OUTPUT:

3. **Definition List:**
   - <dl> , <dt>, <dd> are the tags used to list definitions .
   - DL is used to provide a list of items with associated definitions.
   - Each item should be placed in a DT and its definition goes into DD directly following it.

   <DL> tag:
   - Elements within a definition lists are either items being defined or their definitions.
   - General form

     <dl [compact]> ------ </dl>

   <DT> tag:
   - Definition terms mark items whose definition will be provided by the next data definition.
   - General form

     <dt> -------- </dt>

   <DD> tag:
   - Definitions of terms are enclosed within these tags .
   - The definition can include any text or block formatting text.
   - General form

     <dd> -----</dd>

Ex:

```
<html>
    <head></head>
    <body>
        <h1>Definition List</h1>
        <dl>
            <dt> HTML</dt>
            <dd>HTML is a Hyper Text Markup Language used to create a static web
                Pages</dd>
        </dl>
        <dl>
            <dt>XML</dt>
            <dd>XML is a Extensible Markup Language</dd>
        </dl>
    </body>
</html>
```

OUTPUT:

1.5 Forms

- Forms are used to add an element of interactivity to a website. They are usually used to let the reader send information back to the server.
- A form is made up of one or more fields that allow the user to enter information
- General form:
  <form action="url"
    - Method="get"/"post">

  </form>
- The form is used to create an html form for user input.
- An html form is used to pass data to a server. The URL in action attribute specifies where to send the form data when a forn is submitted.
- Two ways are there to send form data to server
  - Using get()
  - Using post()
- When get() is used, the data is included as part of the URL. In post() emports the data within the body of the http message.
- Post can be used to send for larger amount of data, is for more secure than get().
- Forms are used to select different kinds of user input.
- A Form contain input elements like:
  - Text fields
  - Checkboxes
  - Radio buttons
  - Submit buttons
  - Text area
  - Password field
  - Drop down list
- The most important form element is "input element"
- General form of input element is

<input type="text"/"password"/"checkbox"/radio"
"submit"/"reset"/"button"/name=string"
[Value="string"][checked][size="n"]
[max length="n"]
[src="url"][align="top"/"bottom"]/>

Text:

creates an input upto size characters long and is able to accept upto maximum length characters as input. If value is set, that string will be used as the default string.

TextField defines a one line input field that a user can enter the text information.

Password:

Password field, a special type of text field, which also gets and sends text input from user. But when the user types the input only astricts(*) are displayed on the screen.

Radio:

It creates a radio button. It lets the user to select only one of the limited number of choices. To achieve the typical characteristics of radio button that is mutually exclusive that buttons should be grouped. Buttons can be grouped by giving the same variable name for all the buttons. Gender selection is a best example for radio button.

Checkbox :

It provides a simple checkbox. It lets a user to select one or more options of a limited number of choices.The hobbies selection is the example for the checkbox

Submit:

It creates a button which displays the value attribute as its text. It is used to send data to the server.

Reset:

It also creates a button. But it is used to clear the form.

Drop down list:

It presents a list to the users. The user can select his/her choices from the list. The drop down list definition is begins with <select> and ends with </select>

General form:

```
<select name="text" size="n" multiple>
   ------
   ------
</select>
```

- Name attribute assigns a name for the variable, which will go to the selected choice.
- Size attribute is used to design the number of items in the list to be shown
- Multiple attribute is used to select, multiple items in the list.

Ex:
```
<select name="branch" size="2" multiple>
    <option>MCA</option>
    <option>MBA</option>
    <option>Msc</option>
</select>
```

Text area:

Defines a multi line text input control . the size of the text area is specified the columns and rows attributes. The text area input control can be created by using the <textarea> tag.

General form:

```
<textarea rows="n" cols="n" name="text">
   ------
   ------
</textarea>
```

Ex:<textarea rows=4 cols=30></textarea>

Ex: Design a form:

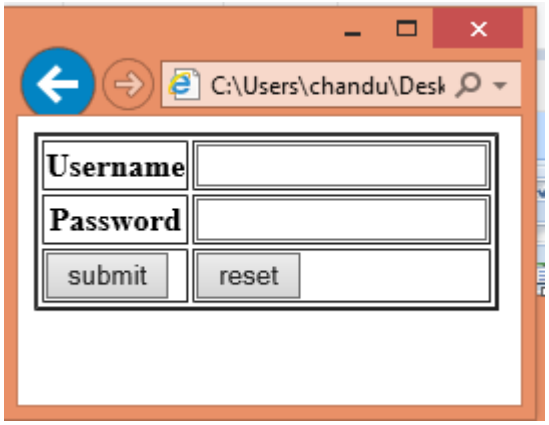
```
<html>
    <body>
        <form>
            <table border = "2" align="center">
            <tr>

                <th> Username</th>
                <th><input type="text"/>  </th>
            </tr>
            <tr>
                <th>Password</th>
                <th><input type="password"/></th>
            </tr>
            <tr>
```

```
                    <td><input type =submit value = "submit"/></td>
                    <td><input type = reset value = "reset"/></td>

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

OUTPUT:



1.6 Frames

- Frames provide a pleasing interface which makes website easily to navigate.
- Frames based sites display more than one page at the same time, they can be complex to setup.
- Once the layout is established, frame based sites can require less maintainance than alternative approaches.
- This is simply because the index is usually in one frame with page content displayed in another.
- If newpages are added to the site, their details are only added to the index.

<frameset> tag:

This tag determines how the screen will be divided between the various frames .

General form of frameset tag is:

```
<frameset [cols=" %/pixels/*"]
          [rows=" %/pixels/*"]>
</frameset>
```

- Cols attribute specifies no.of columns.
- Rows attribute specifies no.of rows.

Value	Description
Pixels	The size of pixels
%	The size in percent of the available space
*	The rest of the available space should be available to either cols or rows

<Frame> tag:

- The <frame> defines one particular window (frame) with in a framset.
- Frame in a frameset can have different attributes such as border, scrolling, the ability to resize etc.

NOTE:

- <frame> tag has no end tag.
- <frameset> and <frame> tags should be without <body> tag

General form:

```
<frame [name="name"] src= "filename"
       [scrolling="yes"/"no"/"auto"]  [frameborder="0"/"1"]/>
```

Attribute	Value	Description

Name	Text	Specifies name of the frame
Src	url	Specifies the url of document to show in a document.
Scrolling	Yes No Auto	Specifies whether or not to display scrollbars in frame
Frameborder	0 1	Specifies whether or not to display the border around a frame.

2. XML

Extensible Markup Language (XML) is a tool which is used to define, store and transport data across platforms/ applications.

- XML is used to store and transport data;
- XML is a case-sensitive language.
- XML tags are not pre-defined.
- XML defines your own tags.
- XML is platform independent and language independent.

XML is not a replacement for HTML. Both are designed for different purposes. HTML is used to display data, whereas XML is used to store and transport data

2.1 The Introduction of XML

Features of XML:

1. **XML simplifies Data Sharing:**
   XML data is stored in plain text format. This provides a software an hardware independent way of storing data. This makes it much easier to create data that can shared by different applications.
2. **XML simplifies Data Transport:**
   One of the most time consuming challenges for developers is to exchange data between incompatible systems.
   Exchanging data as XML greately reduces this complexity, since the data can be different incompatible applications.
3. **XML simplifies Platform Changes:**
   XML data is stored in text format, this makes it easier to expand or upgrade to new operating systems, to new applications or new browsers without using data.
4. **XML makes your data more Available:**
   With XML, data can be available to all kinds of "reading machines"(handheld computers, voice machines etc), and make it more available for blind people, or people with other disabilities.

Differences between HTML and XML:

XML document contains following things:

1) XML Elements:

- An XML document contains XML elements.
- An XML element is everything from the elements start tag to the elements end tag.
- An element can contain:

HTML	XML
HTML is used to display the structure that user designed.	XML is used to store, define, transport data.
HTML is not an case-sensitive language	XML is a case-sensitive
HTML tags are pre-defined tags	XML tags are user-defined tags.
We need not to close all the corresponding closing tags	Must have closing tag to every tag.

1. Other elements
2. Text

3. Attributes
4. Or a mix of the above.

2) XML Entities:

- Some characters have a special meaning in xml
- If character like " < " is placed inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
- Ex: <message> if salary < 1000 then </message> ....will generate an xml error.
- To avoid this error, replace the " < " characters with an "entity reference" i.e <message> if salary &lt ; 1000 then </message>
- There are 5 pre-defined entity references in XML.

Entity reference	Character	Meaning
<	<	Less than
>	>	Greater than
&	&	Ampersand
'	'	Apostrophe
"	"	Quotation mark

3) XML Attributes:

- XML elements can have attributes, just like HTML.
- Attributes provide additional information about an element.
- Attribute provides extra information about information.
- XML attributes must always be quoted. Either single or double quotes can be used.

```
<bookstore>
    <book category= "children">
        <title> Harry Potter</title>
        <author> JK Rowling </author>
        <price>20.35</price>
    </book>
</bookstore>
```

2.2 The Syntax of XML

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

1. **All XML Elements must have a closing tag:**
   In html, some elements do not have closing tag.
   Ex: <p> This is a paragraph
   In XML, it is illegal to omit the closing tag. All elements must have a closing tag.
2. **XML tags are case-sensitive:**
   XML tags are case-sensitive i.e the tag <letter> is different from the tag <Letter>
   Opening and closing tags must be written with the same case.
3. **XML elements must be properly nested:**
   In HTML, you might see improperly nested elements.
   Ex:This text is bold and italic
   In XMl, all elements must be properly nested within each other.that is shown in following example.
   Ex: This text is bold and italic
4. **XML documents must have a root element:**
   XML documents must contain one element that is the parent of all other elements. This element is called the root element.
   Ex: <root>
   <child>
   <subchild> ----------</subchild>
   </child>
   </root>
5. **XML Attribute values must be quoted:**

XML elements can have attribute in name/value pairs just like in HTML.
In XML, the attribute values must alwaysbe quoted.

Ex: <note date="2/9/2017">
<to> Tove</to>
<from>Jani</from>
</note>

Well Formed XML Document

- An XML document that follows correct XML syntax is called "well Formed" XML Document
- XML syntax rules are
  1. All XML Elements must have a closing tag
  2. XML tags are case-sensitive
  3. XML elements must be properly nested
  4. XML documents must have a root element
  5. XML Attribute values must be quoted

Ex: <bookstore>
<book >
<title> everyday Italian
<author> Geada Laurentus</author>
</book>
</bookstore>

The above example is not well formed XML because <title> tag has no end tag...

2.2 XML Document Structure

2.3 Document Type Definition

- DTD means Document Type Definition
- The purpose of a DTD is to define the structure of an XML document.
- If defines the structure or Grammar or schema of an XML document.
- A DTD defines the legal building blocks of an XML document.
- It defines the document structure with a list of legal elements and attributes.
- A DTD can be declared inline inside an XML document, or as an external reference i.e :
  1. Internal DTD Declaration.
  2. External DTD Declaration

1. Internal DTD Declaration:

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root_element [element-declaration] >

Ex:

```
<?XML version="1.0"?>
<!DOCTYPE bookstore [
<!ELEMENT bookstore(book+)>
<!ELEMENT book(title,author,price)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT author(#PCDATA)>
<!ELEMENT price(#PCDATA)>
]>
<bookstore>
    <book>
        <title> Programming with C++</title>
        <author>BalaGuruswamy</author>
        <price>Rs.500</price>
    </book>
```

```
</bookstore>
```

The above DTD is interpreted as:

1. !DOCTYPE bookstore - defines that the root element of this document is bookstore.
2. !ELEMENT bookstore - defines that the bookstore element can contain one or more book elements.
3. !ELEMENT book - defines that the book element contains 3 element i.e title,author,price.
4. !ELEMENT title- defines the title element to be of type "#PCDATA".
5. !ELEMENT author- defines the author element to be of type "#PCDATA".
6. !ELEMENT price – defines the price element to be of the type "#PCDATA".

2. External DTD Declaration:

- DTD declaration can be done internally (i.e within XML document) or Externally (i.e as an External file).
- If the DTD is declared in an external file it should be wrapped in a DOCTYPE definition with the following syntax.
  <!DOCTYPE root.element SYSTEM " filename">
- The same XML document with external DTD looks as

Ex:

```
<?XML version="1.0"?>
<!DOCTYPE note SYSTEM "books.dtd">
<bookstore>
        <book>
                <title> Programming with C++</title>
                <author>BalaGuruswamy</author>
                <price>Rs.500</price>
        </book>
</bookstore>
```

And the books.dtd which contains the DTD is:

```
<!ELEMENT bookstore(book+)>
<!ELEMENT book(title,author,price)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT author(#PCDATA)>
<!ELEMENT price(#PCDATA)>
```

PCDATA:

- PCDATA means Parsed Character data.
- PCDATA is text that will be parsed by a parser. The text will be examined by the parser for entities and markup.
- That is tags inside the text will be treated as MArkup and entities will be expanded.

CDATA:

- CDATA means Character Data.
- CDATA is text that will not be parsed by a parser.
- That is tags inside the text will not be treated as markup and entities will not be expanded.

- From DTD point of view, all XML documents are made up by the following building blocks.
  1. Elements
  2. Attributes
  3. Entities

ELEMENTS are the main building blocks of both XML and HTML documents
ATTRIBUTES provide extra information about elements.
ENTITIES are used instead of special characters like < , >, & etc....

1. **DTD-Elements:**
   - In a DTD, elements are declared with an ELEMENT declaration.
   - Various ways of declaring Elements in DTD are:
     1. Empty elements.
     2. Elements with parsed character data
     3. Elements with children.
     4. Declaring only one occurrence of an element.
     5. Declaring minimum one occurrence of an element.
     6. Declaring zero or more occurrences of an element .
     7. Declaring zero or one occurrence of an element.
     8. Declaring either/or content.

     1. **Empty elements:**
        Empty elements are declared with the category keyword EMPTY.
        <!ELEMENT element-name EMPTY>
        Ex: DTD- <!ELEMENT br EMPTY>
        XML-</br>
     2. **Elements with parsed character data:**
        Elements with only parsed character data are declared with #PCDATA inside paranthesis.
        DTD- syntax: <!ELEMENT > element _name (#PCDATA)
        DTD-ex: <!ELEMENT> from (#PCDATA)
        XML-ex:<from> Jani </from>
     3. **Elements with children:**
        Elements declared with the one or more children are declared with the name of the children elements inside paranthesis.
        DTD syntax: <!ELEMENT element_name (child1, child2,. ..>
        DTD Ex: <!ELEMENT note (to, from, heading)>
        XML ex: <note>
        <to>.....</to>
        <from>. .....</from>
        <heading> .....</heading>
        </note>
     4. **Declaring only one occurrence of an element**:
        If the element should occur only once, then:
        DTD syntax:<!ELEMENT element_name(child-name)>
        DTD ex:<!ELEMENT note (message)>
        The child element "message " must occur once, and only once inside the "note: element.
     5. **Declaring minimum one occurrence of an element:**
        " + " meta character indicates the occurrence of the child element (i.e one or more occurrence).
        DTD syntax: <!ELEMENT element_name (child_name +)>
        DTD ex: <!ELEMENT note (message +)>
        That is the message element can occur one or more times in the note element.
     6. **Declaring zero or more occurrences of an element :**
        Meta character is used to indicate zero or more occurrences of the child element.
        DTD syntax: <!ELEMENT element_name (child.names)>
        DTD ex: <!ELEMENT note message*)
        That is the message element can occur zero or more times in the note elements.
     7. **Declaring zero or one occurrence of an element:**
        ? meta character is used to indicate zero or one occurrence of child element.
        DTD syntax: <!ELEMENT element_name (child.name?)>
        DTD ex: <!ELEMENT note (message?)>
        That is the message element can occur zero or one time in the note element.
     8. **Declaring either/or content:**

" | " meta character idnidcate Either-Or
DTD syntax: <!Element element_name (child1 / child2)>
DTD ex: <!ELEMENT note (to, from, header,...(message/ body))>
The above example declares that the "note" element must contain a "to", "from", header" and either "message" or "body" element.

2. DTD Attributes:

Syntax for declaring the attributes in DTD are
<!ATTLIST element_name attribute Attribute _type default_value>

DTD ex:
<!ATTLIST payment type CDATA "check">
XML ex:
<payment type ="check"/>
The attribute_type can be one of the following

Type	**Description**
CDATA	The value is character data
(en1/en2/. ...)	The value must be one from an enumerated list
ID	The value is a unique ID
ENTITY	The value is an Entity
ENTITIES	The value is a list of entities

The default_ value can be one of the following :

Value	The default value of the attribute
#Required	The attribute is required
#Implied	The attribute is not required
#Fixed value	The attribute value is fixed
Enumerated	The attribute value to be one of a fixed set of legal values

1. A Default Attribute Value:

Default value is used to set default value to an attribute
Ex: DTD
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
Valid XML
<Square width="100"/>
The "square " element is defied to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

2. #REQUIRED:

#REQUIRED keyword should be used you don't have an option for a default value, but still want to force the attribute to be present.
DTD syntax:
<!ATTLIST element_name attribute_name attribute_type #REQUIRED>
DTD example:

```
<!ELEMENT ingredient (qty, item)
 <!ELEMENT qty(#PCDATA)>
        <!ATTLIST qty
                    Amount CDATA #REQUIRED
                    Unit CDATA "G"  >
</ELEMENT item (#PCDATA)>
```

XML example:

```
<ingredient>
        <qty amount ="20" units ="g"/>
        <item >Dried Yeast</item>
```

</ingredient>

3. **#IMPLIED:**
   Use the #implied if you don't want to force the author to include an attribute, and you don't have an option for a default value.
   DTD ex:
   <!ATTLIST contact fax CDATA #IMPLIED>
   Valid XML:
   <contact fax =”555.667788”/>
4. **#FIXED:**
   Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it.
   If an author includes another value, the xml parser will return an error.
   DTD syntax:
   <!ATTLIST element.name att _name att _type #FIXED”value”>
   DTD ex:
   <!ATTLIST sender company PCDATA #FIXED “Microsoft”>
   Valid XML - <sender company= “Microsoft”>
   Invalid XML- <sender company =”w3schools”>
5. **Enumerated Attribute values:**
   Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.
   DTD syntax:
   <!ATTLIST element_name att_name (en1/en2…)default_value>
   DTD ex:
   <!ATTLIST payment type (check/cash)”cash”>
   Valid XML:
   <payment type =”check/> OR
   <payment type=”cash”/>

3. **DTD ENTITIES:**

- Entities are variable used to define shortcuts to standard text or special character or repeated text.
- Entity references are references to entities.
- Entities can be declared internal or external

1. **An internal entity declaration:**
   Internal entities are used to create small pieces of data which you want use repeatedly throughout your schema.
   DTD syntax:
   <!ENTITY entity_name “Entity_value”>
   DTD ex:
   <!ENTITY writer “Donald Duck”>
   <!ENTITY copyright “copyright w3schools”>
   XML ex:
   <author> &writer; &copyright;</author>
   DTD ex2:
   <!ENTITY pos “pinch of salt”>
   XML ex:
   <item> Finally add the &pod;</item>
   An entity has three parts:
   1. An ampersand(&)
   2. An entity name
   3. An a semicolon(;)

2. **An External Entity Declaration:**

External entities are used to create external files which are required to use repeatedly through out your schema.

DTD syntax:

<!ENTITY entity_name SYSTEM "URL/URL">

DTD ex1:

<!ENTITY writer SYSTEM
http://www.w3schools.com/entities.dtd>

XML ex1:

<author> &writer;</author>

3. JAVASCRIPT

3.1) Introduction to JavaScript

3.1.1) what is Dynamic html
3.1.2) JavaScript
3.1.3) Variables
3.1.4) String Manipulation
3.1.5) Mathematical Functions
3.1.6) Statements
3.1.7) Operators
3.1.8) Arrays
3.1.9) Functions

3.1.1) What is Dynamic HTML

Dynamic HTML is a combination of content formatted using

- ✓ HTML
- ✓ CSS
- ✓ Scripting Language
- ✓ DOM(Document Object Model)

DHTML = HTML + CSS + Scripting language + DOM

HTML DOM defines a standard way for accessing and manipulating HTML documents.

DOM presents an HTML document as a tree structure.

3.1.2) JavaScript

- Introduction.
- How easy is programming in JavaScript
- Benefits of JavaScript.
- problems with JavaScript.

Introduction:

- JavaScript is a scripting language designed primarily for adding interactivity to webpage.
- JavaScript developed by Net Scape Navigator in 1995 as a method for validating forms and provide interactive content to website.
- JavaScript is a scripting language used to make web pages interactive.
- JavaScript code can be imbedded in HTML pages and interpreted by the web browser (or client)

Scripting languages:

- A Scripting language is a form of programming language that is usually interpreted rather than compiled.
- A conventional programs are converted permanently into executable files before they are run.
- In contrast, programs in scripting language are interpreted one command at a time.
- Scripting language are often written to facilitate enhanced features of websites.

How easy is programming in JavaScript:

- Programs tend to be large piece of code where as script are small pieces of code.
- Of course there are difference between full-scale programs and small pieces of JavaScript.

- Compiled programs are hardware and operating system specifies and have to be compiled separately for every platform on which they will execute where as scripting language which will be run by the interpreter, any script can be run on any system that contains a suitable interpreter.
- Users on any system, whether linux, windows 2000, apple or anything else can use that script if their browser contains a suitable JavaScript interpreter
- So JavaScript is nicely platform independent and can be run every where.
- JavaScript has been designed to run through browsers and can actually do very little.

Benefits of JavaScript:

JavaScript has a number of big benefits to anyone who wants to make their website dynamic.

- It is widely supported in web browsers
- It gives easy access to the document objects and can manipulate most of them.
- JavaScript can give interesting animations without the long download times associated with many multimedia data types
- Web browsers don't need a special plug-in to use your scripts.

problems with JavaScript:

- Most scripts rely upon manipulating the elements of the DOM and access to objects differs from browser to browser.
- If your script doesn't work then your page is useless.
- Scripts can runs slowly and complex scripts can take a long time to start up.

3.1.3) Variable

- Variable names
- Data types
- Creating variables

Variable names:

- A variable is a named location in memory that is used to hold a value that can be modified by the program.
- There are strict rules governing how to name variables in javascript
    1. Names must begin with a letter, digit or the underscore.
    2. You cannot use spaces in names.
    3. Names are case-sensitive.
    4. You cannot use a reserved word as a variable name.

Data types:

- Programming languages usually have several different types of data.
- Commonly programmers may use characters, integers, Booleans, strings, real and many others besided.
- Java script has only 4 types of data
    - ✓ Numeric
    - ✓ Strings
    - ✓ Boolean
    - ✓ Null

Creating variables:

" var " is the keyword used to create variables in java script.

Ex: var first= 23;
Var second = "some words";

- Java script keyword var is often optional, since we want our scripts to be consistent and to work as we intended when we wrote them, its best to always use var before variable declarations

3.1.4) String Manipulation Functions

charAt(index):

This function returns the character which is at position index in the string.

Ex: var str = "JavaScript";

```
var res = str.charAt(4);
document.writeln(res);      // prints " s "
```

concat("string" [, "string"[.."string"]])

This function is used to add two strings

```
Ex: var name = prompt("enter your name");
    var msg = "welcome";
    var res = msg.concat(name);
    document.writeln(res);          //prints welcome smith
```

indeOf("search"[,offset]):

- The string is searched for the string in the first parameter.
- If the search is successful, the index of the start of the target string is returned.
- If the search is unsuccessful the operation returns -1
- By default the indexOf ( ) functions starts index at 0 (zero).
- An optional offset may be specified so that the search starts part way along the string.

```
Ex:     var str = "JavaScript";
        var res = str.indexOf("S");
        document.writeln(res);      // prints " 4 "
```

lastIndexOf("search" [,offset]):

- This function does exactly the same thing as indexOf () but works its way backwards along the string.
- The offset works in exactly the same way as for indexOf ().

```
Ex:     var str = "JavaScript";
        var res = str.lastIndexOf("a");
        document.writeln(res);      // prints " 3 "
```

length:

Returns number of characters in the string.

```
Ex:     var str = "JavaScript";
        var res = str.length;
        document.writeln(res);      // prints " 10 "
```

split(separator [,limit]):

- This function breaks the string a part whenever it encounters the character passed in as the first parameter.
- The pieces of the string are stored in an array.
- Split() has an optional second parameter which is an integer value indicates how many of the pieces are to be stored in the array.

```
Ex:  var str="JavaScript is a scripting language";
     var res= str.split( "  ");
     for(var i=0;i<res.length;i++)
          document.writeln(res[i] + "< /br>");  // prints   "  JavaScript
                                                              is
                                                              scripting
                                                              language"
```

subStr(index[,length]):

- Returns a substring which starts at the position specified by index.
- The substring continues either to end of the string or for the no.of characters indicated by the length parameter.

```
Ex:     var str = "JavaScript";
        var res = str.substr(4,3);
        document.writeln(res);      // prints " scr"
```

substring(index1,[index2]):

Returns the set of characters which starts at index1 and continues up to, but does not include, the character at index2

```
Ex:     var str = "JavaScript";
```

```
var res = str.subString(4,6);
document.writeln(res);      // prints " sc  "
```

toLowerCase( ):

Converts all characters in the string to lowercase.

Ex:
```
var str = "JavaScript";
var res = str.toLowerCase();
document.writeln(res);      // prints " javascript "
```

toUpperCase( ) :

Converts all characters in the string to uppercase.

Ex:
```
var str = "javascript";
var res = str.toUpperCase();
document.writeln(res);      // prints " JAVASCRIPT  "
```

3.1.5) Mathematical Functions

- Mathematical functions and values are part of a built in javascript object called Math.
- JavaScript provides following mathematical functions.

abs (value) :

Returns the absolute value of the number passed into it.

Ex: abs(-3); //prints 3

ceil(value):

Returns the smallest integer which is greater than, or equal to, the value passed in.

Ex: ceil (22.8) ,ceil(22.4); //prints 23,23

floor (value):

Returns the largest integer which is smaller than, or equal to, the number passed in.

Ex: floor (22.8),floor(22.8) ;//prints 22,22

min(value1, value2):

Returns the smaller of its argument.

Ex: min(3,4); // prints 3

max(value1, value2):

Returns the largest of its argument.

Ex: max(3,4); // prints 4

pow(value1, value 2):

Returns the result of raising value to power.

Ex: pow(2,4); // prints 16

round(value):

Returns the result of rounding its argument to the nearest integer.

Ex: round (5.3);// prints 5

sqrt(value):

Returns the square root of the value

Ex: sqrt(144);//Prints 12(sqrt of 144 is 12)

3.1.6) Statements

3.1.7) Operators

Following are some of the operators:

Operator	Meaning
+	If the argument are numbers then they are added together. If the arguments are strings then they are concatenated.
-	If supplied with two operands this subtracts one from the other. If supplied with a single operand it reverse its sign.
*	Multiplies two number together.
/	Divides the first number by the second.
%	Modulus division returns the integer remainder from a division.
!	Logic Not returns false if the operand evaluates to true. Otherwise it returns true.
>	Greater than returns true if the left operand is greater than the right.
<	Returns true if the left operand is less than the right
>=	Returns true if the left operand is greater than or equal to the right one.
<=	Returns true if the left operand is less than or equal to the right one.
==	Returns true if the two operands are equal
!=	Returns true if the two operands are not equal.
=	Assigns a value to a variable.
//	Logical Or returns true if one or both operands are true, otherwise returns false.
&&	Logical And returns true if both operands are true otherwise returns false
+=	Adds two no's then assigns the result to the one or the left of the exp.
-=	Subtracts the term on the right from the term on the left, then assigns the result to the one on the left exp
*=	Multiplies two values then assigns the result to the one of the left of the exp.
/ =	Divides the term on the left by the term on the right and then assigns the result to the one on the left of the exp.
++	Auto-increment, increases the value of its argument by one
--	Auto-decrement , decreases the value of an integer by one
% =	Performs modulus division then assigns the result to one on the left of the exp.

3.1.8) Arrays

- An array is an ordered set of date elements which can be accessed through a single variable name.
- In many programming language arrays are contiguous areas of memory which mean that the first array element is physically located next to the second, and so on.
- In JavaScript, an array is slightly different because it is a special type of object and has functionality which is not normally available in other languages.

Basic Array Functions:

- In javascript, arrays are special type of objects.
- The basic operations that can be performed on traditional arrays are
  1. Creation
  2. Addition of elements
  3. Accessing individual elements
  4. Removing element
  5. Searching element

1. Creating arrays :

Three ways of creating javascript arrays are

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday"];
var days =new Array (" Monday", "Tuesday", "Wednesday", "Thursday");
var days =new Array(4);
```

2. Addition elements to an array:

- Array elements are added by their index. The index denotes the position of the element in the array and these start form 0(zero).
- Adding an element uses the square bracket syntax
  Ex: var days[3]="Monday";

What happens if we want to add an item to an array which is already full, the javascript interpreter simply extends the array and inserts the new line.

3. Accessing Array:

The elements in the array are accessed through their index.

```
Ex: var days=["mon", "tue", "wed"]
    for(var  i=0;i<days.length;i++)
        document.writeln(days[i]);
```

4. Searching an array:

To search an array, simply read each element and compare it with search element.

```
Ex: for(var  i=0;i<length;i++)
    {
      if (data[i]== "Tuesday") {
        document.writeln(data[i]+" , ");
        break;      }
      }
```

5. Removing array members:

JavaScript does not provide a built in function for removing an element from an array. So the following procedure is used to remove an element.

- Read each element in the array
- If the element is not the one you want to delete, copy it into a temporary array
- If you want to delete the element then do nothing .
- Increment the loop counter
- Repeat the process.

Ex:

```
<script>
    var days = ["Monday", "Tuesday","Wednesday",Thursday"];
    for(var i=0; i< days.length;i++)
        document.writeln(days[i]);
    var rem = prompt ("enter element to remove");
    var temp=new Array(days.length -1);
    var count=0;
    for(var i=0;i<len;i++)
    {
        if(days[i]==rem)
        {
        }
        else
        {
            temp[i]=days[i];
            count++;
        }
    }
</script>
```

Object-based Array function:

Since array is an object in Java script the following Array functions helps to manipulate Array elements.

1. concat(array2[, array3 [,array n]]):

A list of arrays is concatenated onto the end of the array and a new array returned.The original arrays are all unaltered by this process.

```
Ex: var first=[1,2,3];
    var second=[4,5];
    var third=[6,7,8];
    var res=first.concat(second, third);
```

```
document.writeln(res);          // prints 1,2,3,4,5,6,7,8
```

2. join(string):

- This function joins all the elements of array as a string.
- In the resulting string, the elements are separated using the optional string parameter.
- If this is omitted the elements will be separated using a comma(,)

```
Ex:   var a =[1,2,3,4];
      var res= a.join(“  ; “ );
      document.writeln(res);          //prints  1;2;3;4
```

3. pop( ):

Removes the last element from the array.

```
Ex:   var a=[4,5,6,7];
      a.pop( );
      document.writeln(res);          //prints    4,5,6
```

4. push (element1 [, element 2 [, element n]]):

Adds a list of items into the end of the array.

```
Ex:   var a=[1,2,3];
      a.push(4,5,6);
      var res1=a.join(“  ; “);
      document.writeln(res);          //prints  1;2;3;4;5;6
```

5. reverse( ):

Swaps all the elements in the array so that which was first is last, and viceversa.

```
Ex:   var a=[1,2,3];
      a.reverse();
      var res1=a.join( “ ; “);
      document.writeln(res1);         //prints 3;2;1
```

6. shift( ):

removes the first element of the array

```
Ex:   var a=[1,2,3];
      a.shift();
      var res1=a.join( “ ; “);
      document.writeln(res1);         //prints 2;3
```

7. slice(start [,finished]):

To extract a range of elements from an array

```
Ex:   var a=[1,2,3,4,5,6,7];
      a.slice(1,3);
      var res1=a.join( “ ; “);
      document.writeln(res1);         //prints 2;3
```

8. sort():

Helps to sort array elements into lexicographic, dictionaryorder.

```
Ex:   var a=[9,8,6];
      a.sort( );
      var res1=a.join( “ ; “);
      Rdocument.writeln(res1);        //prints  6;8;9
```

9. splice(number [,element1][, element2[element 3]]]):

- splice is used to alter an array by removing some elements and at the same elements and at the sametime adding in new ones.
- The first parameter indicates the position in the array at which the new element will start.
- The second parameter indicates how many elements will be deleted from the original array.
- If you don’t want to delete element then set this to o
- Finally there is a list of elements which are to be inserted into the array.

```
Ex:   var a=[1,2,3,6,7];
      var res=a.splice(3,0,4,5);
```

```
var res1=a.join( " ; ");
document.writeln(res1);                    //prints 1; 2;3;4;5;6;7
```

3.1.9) Functions

- A function is a piece of code that performs a specific task.
- Functions are defined using the function keyword.

```
function Name(parameter)
{

}
```

- The function name can be any combination of digits, letters and underscores but cannot contain a space.
- That's the same rule as for variable meaning.
- To find factorial of a number using function is:

```
<html>
    <head>
        <script>
            function factorial ( k )
            {
                var fact=1;
                for(var i=1;i<=k;i++)
                    fact=fact*i;
                    return fact;
             }
        </script>
    </head>
    <body>
        <script>
            var n=prompt("enter n value:");
            var res=factorial ( n );
            document.writeln(res);
        </script>
    </body>        </html>
```

- **Program to find n_{cr} using recursion.**

```
<html>
    <head>
        <script>
            function factorial ( k )
              {
               if(k==0)
                    return 1;
               else
                    return * factorial(k-1);
            }
         </script>
    </head>
    <body>
        <script>
            var n=prompt("get n value");
            var r=prompt("get r value");
            var nr=n-r;
            var fact-n = factorial(n);
            var fact-r = factorial(r);
            var fact-nr = factorial (nr);
            var res=(fact-n)/(fact-r *fact-nr);
```

```
                document.writeln(res);
        </script>
    </body></html>
```

3.2) OBJECTS IN JAVASCRIPT

3.2.1) Data and objects in JavaScript.
3.2.2) Regualr Expressions
3.3.3) Exception handling
3.3.4) Built-in Objects

3.2.1) Data and Objects in JavaScript

- JavaScript has several built-in objects like string, date, array etc..
- In addition to these built-in objects, we can also create your own object.
- An object is just a special kind of data, with collection of properties and methods.
- For example: A person is an object , properties are the values associated with the object.
  The persons properties include name, height, weight etc..All persons have these properties, but the values of those properties will differ from person to person.Objects will have methods. Methods are actions that can be performed on objects.The persons methods could be eat( ), sleep( ), work( ), paly ( ) etc..

Steps to create user defined object:

1. Declare the object by using an object function.
2. Add properties to the new object.
3. Add methods to the new object.
4. Instantiate the newly created object by using the "new" keyword.

1. Declare the object by using an object function:

- First step forwards creating an object requires to define an object function. An object function is virtually identical in syntax as a regular function.

```
funtion userobjecct(parameter)
{
}
```

- The new object called user object does nothing at this stage, and will continue to do until we add in properties and methods.

2. Add properties to the new object:

To add properties to a user defined object, directly embed the properties into the object funtion, with each property proceeded by the keyword " this " plus dot (.)

```
funtion userobjecct(parameter)
{
        this.firstProperty=parameter;
        this.secondProperty="this is the second property";
}
```

3. Add methods to the new object:

- Adding methods to a user defined object is a bit more complicated.
- We need to first declare and define a function for each method, then associate this function with the object function

4. Instantiate the newly created object by using the "new" keyword:

- Once were defined an object function, we have to instantiate it to actually use it.
- Instantiating an object function means using the keyword "new" in front of the object, name .
- Creating an instance of the object by assigning it to a variable.

```
<html>
    <head>
        <script>
            function circle(r)
            {
                this.radius=r;
                this.area=computeArea;
```

```
                this.diameter=computeDiameter;
            }
            funtion computeArea( )
            {
                var a=this.radius*this.radius*3.14;
                return a;
            }
            function computeDiameter( )
            {
                var d=this.radius*2;
                return d;
            }
        </script>
    </head>
    <body>
            <script>
                var c= new Circle(5);
                document.writeln("Radius of circle is" + c.radius( ));
                document.writeln("Area of circle is" + c.area( ));
                document.writeln("Delimeter of circle is" + c.diameter( ));
            </script>
    </body>
</html>
```

3.2.2) Regular Expressions

- A regular expression is an object that describes a pattern of characters.
- A regular expression is a sequence or a pattern of characters that is matched against a string a text when performing searches and replacements.
- Regular expressions are used to perform pattern matching and "search – and – replace" functions on text.
- There are two ways to create a Regular Expression in JavaScript.
  1. Using literal syntax:
     var reExample = / pattern / modifiers
  2. Using the Regular Expression ( ) constructor:
     var reExample = new RegExp("pattern, modifiers");

  patterns specifies the pattern of an expression.

  Modifiers specify if a search should global, case-sensitive etc..
- If Regular Expression pattern is known is advance, then there is no real difference between the two however, if you don't use the pattern ahead of time, it can be easier to use the RegExp( ) constructor.
- For ex:

```
<html>
    <head>  </head>
        < body>
            <script>
                var pattern = /blue/;
                var string=" the sky is blue ";
                var result = string.match(pattern);
                if(result)
                    document.writeln("string found:"+result[0]);
                else
                    document.writeln("string not found");
            </script>
        </body>
</html>
```

- The match( ) method searches for a match between a regular expression and a string, and returns the methods.
- This method returns an array matches or null if no match is found.
- The following program is about using Regualr Expression object.

```
<html>
        <head>  </head>
        < body>
            <script>
                        var ss =prompt ("enter the pattern");
                        var pattern = new RegExp(ss);
                        var gs=prompt ("enter the original string");
                        var res=pattern .exec(gs);
                        if(res)
                            document.writeln("string found:"+res[0]);
                        else
                            document.writeln("string not found");
            </script>
        </body>
</html>
```

- ***Regular expression are manipulated using functions which belong to either the RegExp or String class.***

String Class functions:

1. **match(pattern):**
   searches for a matching pattern. Returns an array holding the results, or null if no match is found
2. **replace (pattern1, pattern2):**
   searches for pattern1 if the search is successful pattern1 is replaced with pattern2..
3. **search(pattern):**
   searches for a pattern in the string. If the match is successful, the index, offset, of the start of the match is returned fails, the function returns -1
4. **split(pattern):**
   splits the string into parts based upon the pattern, or reg exp, which is supplied as a parameter.

RegExp class functions:

1. **exec(string):**
   executes a search for a matching pattern in its parameter string. Returns an array holding the results of the operations.
2. **test (string):**
   searches for a match in its parameter string. Returns true if a match is found, otherwise returns false.

> **NOTE:**
> When literal syntax is used for creating RegExp, then string functions are used for pattern matching.
> Whereas if RegExp constructor is used for creating RegExp, then RegExp functions are used for pattern matching.

Pattern matching using string class functions:

1. **match function:**

```
<script>
        var pattern = /Fred/fred/;
        var gs=" have you met fred recently";
```

```
var res=gs.match(pattern);
if(res)
        document.writeln("string found:"+res[0]);
else
        document.writeln("string not found");
</script>
```

Output: string found fred

2. Repalce function:

```
<script>
        var pattern = "smith";
        var gs=" have you met fred recently";
        var res=gs.repalce("fred",pattern);
        document.writeln(res);
 </script>
```

Output: have you met smith recently

3. search(pattern):

```
<script>
        var pattern = "smith";
        var gs=" have you met fred recently";
        var res=gs.search(pattern);
        document.writeln(res);
</script>
```

Output: -1 (because smith is not there in GS)

4. split(pattern)

```
<script>
        var pattern = "  ";
        var gs=" have you met fred recently";
        var res=gs.split(pattern);
        for(var i=0;i<res.length;i++)
                document.writeln(res[i] +"</br>");
 </script>
```

Output:
Have
You
Met
Smith
Recently

Using RegExp object function:

1. **exec(string):**

```
<script>
        var pattern = "smith";
        var gs=" have you met fred recently";
        var res=gs.repalce("fred",pattern);
        document.writeln(res);
 </script>
```

Output: have you met smith recently

```
        var pattern = prompt("get the pattern");
        var re=new RegExp(pattern);
        var gs=prompt("get the givens stirng");
        var res=re.test(gs);
        document.writeln("res");
```

Flags or Modifiers:

i	Ignore the case of characters	/The/ i matches "The" and "The" and tHe"

g	Global search for all occurances of a pattern	/ain/ g matches both “ain” in “no pain no gain”, instead of list the first.
gi	Global search, ignore case	/it/gi matches all “it” and in “It is our IT department”.
m	Allows searching of data which spans several input times	/it/m matches all it i.e even in multiple lines

Meta characters:

- The following table lists the meta character that can be used in javascript regular expressions.
- Meta-characters provide for flexible patterns and are expanded to match ordinary characters.

Character	Matches	Example
^	Match at the start of the input string	/^Hi / matches “Hi” in “Hi Mark”
$	Match at the end of the input string	/Mark$/ matches “Mark” in “and Mark”
+	Matches one or more preceding form	/ahtz/matches “ahx” or “ahhhhx”
*	Zero or more preceding term	/ah*x/ matches ax, ahx, ahhx,.....
a/b	Matches a or b	/John /&ara/ matches “John” in “call John and Bob” or “sar” in “Also get sar”
?	Zero or one preceding term	/ah?x/ matches “ax” and “ahx” but not ahhx
\b	Word boundary	/ \ bto/ matches “to” in “today”
\B	No word boundary	/ \Bto / matches “to”in store.
\d	Digits o thro’g	/ H\ d/ matches “H3”
\D	Not a digit	/ H \ D/ matches “HG”
\s	Single white space	/cross\s browser/ matches “cross browser”
\S	Non white space	/cross\sBrowser/matches “cross-Browser”
\w	Letters, numbers	/ \ / w/ matches “/b” or “/-“
\W	Not letters, numbers and underscore	/|\W/ matches “/ %”
[“]	One of set	/th [eo] sc/ matches “those” or “these”
{~..}	Not one of set	/ th [^ eo]s/ matches “this”
{n}	Exactly n preceding term	/ \ d {3} / matches “123”
{n,}	N or more preceding term	/ \ d{2,}/ matches “456” but not “4”
{n,m}	Atleast n atmost m	/ \d{2,4}/ matches”123”
$	Match at the end of the input string	/Mark$/ matches “Mark” in “ and Marks”.....

Example programs:

1. **Example**

```
<script>
        var pattern = /ah?x/g;
        var gs=” ax ahx ahhx ahhhx agx”;
        var res=gs.match(pattern);
        var str=res.join(“ : “);
        if(res)
                document.writeln(“string found:”+str);
        else
                document.writeln(“string not found”);
 </script>
```

2. **Example** (a/b)

```
<script>
```

```
var pattern = /Fred/fred/;
var gs=" have you met fred recently";
var res=gs.match(pattern);
if(res)
        document.writeln("string found:"+res);
else
        document.writeln("string not found");
</script>
```

3. **Example (+ one or more preceding term)**

```
<script>
var pattern =/ah + x/g;
var gs=" ax ahx ahhx ahhhx agx";
var res=gs.match(pattern);
var str=res.join(" : ");
if(res)
        document.writeln("string found:"+str);
else
        document.writeln("string not found");
</script>
```

3.2.3) Exception Handling

- Errors can be classified into 3 types.
  1. Compile-time errors.
  2. Run-time errors
  3. Logical errors
     - ✓ Syntactical errors found at compile-time are called compile-time errors.
     - ✓ Semantic errors found at run-time are called run-time errors.
- An Exception is run time error (or) An exception is abnormal condition that arises in a code sequence at the run time.
- Errors found in logic of the program are called logical errors.
- Javascript exception handling is managed via five keywords
  1) Try
  2) Catch
  3) Throw
  4) Throws
  5) Finally
- Statements that need to be monitored for exceptions should be placed within a **try** block
- If an exception occurs within the try block, it is thrown.
- Your code can catch this exception using **catch** block and handles it in some rational manner.
- To manually throw an exception, use the keyword **throw**.
- Any exception that is thrown out of method must be specified as such by a **throws** clause.
- Any code that obsolutely must be executed after a try block completes is put in a **finally** block.

General Form Of An Exception Handling Block

```
try
{
        Block of code to moniter for errors
}
catch(Exception type1 exod)
{
        Exception handler for Exception type1
}
catch(Exception type2 exob)
```

```
{
        Exception handler for Exception tpye2
}
finally
{
        Block of code to be executed after
        Try block ends
}
```

Example:

```
<html>
        <head>
        <script>
                function message ( )    {
                        try       {
                                add alert("welcome guest");
                        }
                        Catch(e)
                        {
                                var text = "there was an error on this page \n\n";
                                text t= "error description:" + e.message + \n\n";
                                text t="click ok to continue \n\n";
                        }
                }
                alert(text);
        </script>
        </head>
        <body>
                <input type ="button" value ="view message" onclick= "message( )">
        </body>
</html>
```

Finally block example:

```
<html>
        <head>
                <script>
                        Function myfunc( )
                        {
                                var c=100'
                                try
                                {
                                        alert("value of variable a is "+ a);
                                }
                                catch(e)
                                {
                                        alert("error"+e.description( ));
                                }
                                finally( )
                                {
                                        alert("finally block will always execute");
                                }}
                        </script>
                        </head>
                        <body>
```

```
                    <p> click the following to see the result</p>
                    <form>
                        <input type="button" value = "click me"
                        onclick="myfunc( )">
                    </form>
                </body>
</html>
```

Throw example:

Throw clause is to throw exceptions manually.

```
<script>
    function fun1( )
    {
        var a=100;
        var b=10;
        try
        {
            if(b==0)
                throw ("divide by zero errors");
            else
                var c=a/b;
        }
        catch(e)
        {
            alert("Error:"+e);
        }
    }
</script>
```

3.2.4) Built-in-objects

- A javascript object is a complex variable with properties and methods.
- In javascript, almost everything is an object. Even primitive datatypes can be treated as objects.
- Most of the objects are prebuilt i.e it comes with the browser.
- Some of the built-in objects of javascript are:
  1. Document object
  2. Window object
  3. Form object
  4. Date object
  5. Browser object

1. Document object:

- A document is a webpage that is being either displayed or created.
- The document has a no.of properties and methods that can be accessed by java script programs and used to manipulate the content of the page.
- Some of the document object properties and methods are:
  1. bgcolor
  2. fgcolor
  3. anchors
  4. links
  5. forms
  6. layers
  7. close
  8. write,writeln

- ✓ bgcolor & fgcolor properties to set back ground and fore ground colors to webpage.
- ✓ Anchors property is an array which contains anchor names in the order in which they appear in the html document.
- ✓ Links also an array which contains linknames in the order in which they appear in the html document.
- ✓ Forms is an array in the order of the document.
- ✓ Layers is an array in the order of the document.
- ✓ Close( ) the document isn't completely written until the close( ) method has been called. If you don't use this method then the browser will keep writing for more data even if there is none.
- ✓ Write( ), writeln( ) methods to display content in webpage.

Ex:

```
<html>
    <head></head>
    <body>
        <a name= "first " href="sample.html">anchor1 </a>
        <script>
            document.bgcolor="blue";
            document.fgcolor="yellow";
            document.writeln(document.anchors[0].name);
            document.writeln(document.links[0].href);
        </script>
    </body>
</html>
```

2. Window object:

- ➢ Window object to create and manipulate a window on webpage.
- ➢ Two important methods of window object are
    1. open( )
    2. close( )

open: method to open a new window. General form of open( ) is:

open("url", "name", "attribute");

- ➢ url is the path of the file to be opened in new window.
- ➢ name is the name of the new window.
- ➢ attribute to control the look of the opened window with.

Window attribute	Description
Full screen	Specifies if window should be opened in full screen mode.
Height	Specifies the height of the window in pixels
Left	Specifies the x co-ordinate of the window in pixels
Menu bar[1/0]	Specifies if the menu bar of the window should be included.
Resizable [1/0]	Specifies if window should be resizable
Scrollbars[1/0]	Specifies if window should contain scrollbars
Status[1/0]	Specifies if the status bar of the window should be included
Toolbar[1/0]	Specifies if the toolbar of the window should be included.
Width	Specifies the width of the window.

3.Form object:

Forms on the page are represented in javascript via the form object, and can be accessed in one of the 3 standard ways.

1. document.your formname;
2. document.form.your formname;
3. document.forms["your formname"];

some of the properties of form object are:

Property	Description
Action	Read/ write string that reflects the action attribute of the form
Element[ ]	An array containing all of the elements of the form
Length	The number if elements in the form
Method	Read/write string that specifies how the method the form is submitted.
Name	The name of the form s

Ex:

```
<html>
    <head>
        <script>
            Function validate( )
            {
                var method=document.forms[0].method;
                var action=document.forms[0].action;
                var var1=document.forms[0].elements[0].value;
                if(value ! = “mary”)
                    document.forms[0].reset( );
                else
                    alert(“Hi mary);
            }
            </script>
            </head>
            <body>
                <form method=”post” >
                    <input type=”text” name = “user”/>
                    <input type=”submit” value =”press me”
                        onclick=validate( )”/>
                </form>
            </body>
            </html>
```

4.The Browser object:

- Browser is a java script object which can be used to find out which browser is being used to view our webpages.
- For historical reasons the browser object is actually called the navigator object.
- some of the properties of navigator object are :

1. **navigator.appCodeName**
   To get Inetrnal name of the browser. For both major products this is Mozilla.
2. **navigator.appName**
   To get the public name of the browser.
3. **navigator.appVersion**
   To display version of the browser.
4. **navigator.userAgent**
   The strings appCodeName and appVerrsion concatenated together.

```
<html>
    <head>
        <script>
            function BrowserDetails
            {
                var IntName=navigator.appCodeName;
                var Pubname=navigator.appName;
                var ver=navigator.appVersion;
```

```
                    var UAgent=navigator.userAgent;
                    document.writeln("internal name of the browser" + IntName
                                                    +"</br>");
                    document.writeln("public name of the browser" + pubName
                                                    +"</br>");
                    document.writeln("version of  the browser" +ver+
                                                    +"</br>");
                    document.writeln("UserAgent of the browser" + UAgent
                                                    +"</br>");
            <body>
                    <form action=" post">
                            <input type="button" value ="Browserinformation"
                                    On click="browserDetails( )">
                    </form>
            </body>
    </html>
```

5) Date object:

JavaScript includes a well developed date class which provides functions to perform many different date manipulations

Function	Description
getDate	**Return the day of the month**
getDay	Return an integer representing the dayof the week
getHours	Return the hour field of the date object
getMonth	Return the month field of the date object
setDate(day)	Set the day value of the object (range from 1 to 31)
setMonth(month[, day])	Set the month value to an integer in the range o through 11

```
<html>
    <head></head>
    <body onLoad="Dater ( )">
    <script>
        function Dater( )
        {
            var today =new Date( );
            var yesterday =new Date( );
            var diff=today.getDate(diff);
            document.writeln("the date is"+today);
            document.writeln("the date yesterday was "+ yesterday);
        }
    </script>
    </body>
    </html>
```

UNIT - II

Servlet

1) **Life cycle of a Servlet**
2) **A Simple Servlet**

3) **The Servlet API**
4) **The Javax Servlet Packages**
5) **Reading Servlet Parameters**
6) **The javax Servlet HTTP Package**
7) **Handling HTTP Requests and Response**
8) **Using Cookies**
9) **Session Tracking**

Servlet are small programs that execute on the server side of a web connection.

- Normally client side programming takes care of user interface, but the server side programming deals with actual handling of data. This data is typically stored in database.
- Hence if we develop an application which consists of
  Client Program (it may be applet or scripting page)
  Server Program (it may be servlet or Jsp) and **Database**, then it becomes an useful web application.
  i.e., servlet can process and store the data submitted to the server.

How Servlet works

When a client makes a Request to some servlet, he/she actually use web browser in which request is written as URL.

- The web browser then sends this Requests to web server. The web server first finds the Requested servlet.
- The obtained servlet gather the relevant information in order to satisfy the clients request and builds a web page accordingly.
- This web page is then displayed to the client. Thus the request made by the client gets satisfied by the servlet.

- We can configure our computer as a web server by downloading the freely available web server like Apache or the Microsoft IIS.

After downloading this software we have configure yours pc for the corresponding web server.
Then our pc can act as a web server and we can't host some web sites from your own web server.

<u>The famous web servers are</u>:

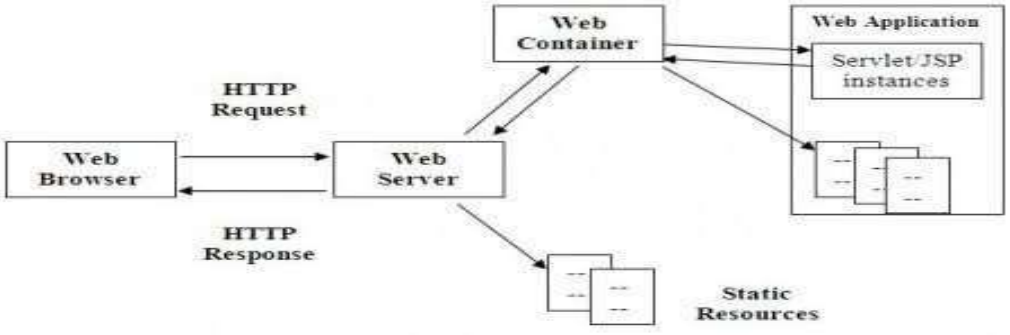
- **IIS**(Internet Information Server)
- Personal Web Server(PWS)

- Java Web Server
- Apache Tomcat Server
- Web Logic
- Google Web Server(GWS)

  PWS or GWS are commonly used web server for development purposes.

Servlet Container

- ✓ If we want to go for serious server side programming then writing the servlet is the best choice.
- ✓ These servlet need a special environment which they can be executed. For that matter, one commonly recommended servlet container is Tomcat.
- ✓ The Tomcat is an open-source product. It is maintained by Jakarta project of Apache Software Foundation.
- ✓ The Tomcat is basically a servlet container that contains the class libraries, documentation and runtime support, which is useful for executing and testing the servlets.
- ✓ Servlet containers are also referred to as web containers or web engines.
- ✓ A servlet containers is a runtime environment to provide life cycle management(i.e., for loading and unloading)
- ✓ Beside Life Cycle Management a container provides other facilities such as
  - Communication Support
  - Multithreading Support
  - Declarative Support
  - Session Management etc.
- ✓ A list of some of the most popular web containers is given below:
  - Apache Tomcat
  - JBoss
  - Java System Application Server
  - Web objects
  - Macromedia JRun etc.



Web container plugged into the webserver

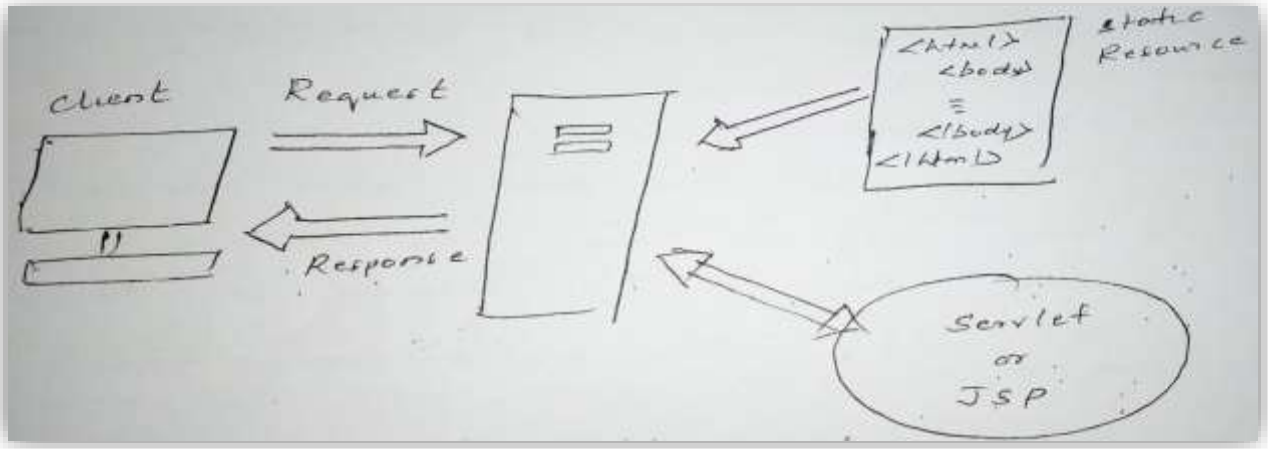
- ✓ The purpose of web server is to intercept all the request, delegate the request to the appropriate resource and send back the response.
- ✓ If the request is for static resource, then the web server can directly access the static resource. Otherwise, request for the dynamic resource (servlet, jsp) is delegated to the web container, plugged into webserver.

Web container as an internal part of web server

The HTTP Request /Response Model

- ✓ HTTP defines how clients and servers communicate.
- ✓ HTTP is a stateless protocol. This means that the server doesn't keep any information about the client after it sends its response, and therefore it can't recognize that multiple requests from the same client may be related.

HTTP Request/Response with two Resources



Let us see, how it works:

- ✓ A Client typically a web browser, sends a request for a resource to a server and the server sends back a response corresponding to the resource.
- ✓ A resource can be a number of things, such as a simple HTML file or a program that generates the response dynamically.
- ✓ Let us see HTTP Request and HTTP Response in detail.

HTTP Request

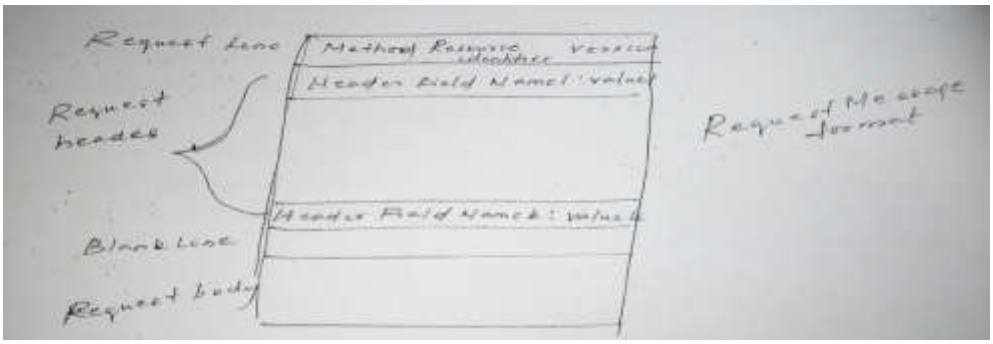
- A user sends a request to the server by clicking a link on a web page, submitting a form, or typing a web page address is the browsers address field.
- To send a request, the browser needs to know which server to talk to and which resource to ask for.
- This information is specified by an HTTP URL **http://www.university.com/index.html**
- The first part of the URL shown specifies that request is made using the HTTP protocol.
- This is followed by the name of the server in this case www.university.com
- The request for the desired web page can be mentioned in third part
- The web server waits for requests to come in on a specific TCP/IP port.

Port number 80 is the standard port for HTTP Request. If the web server uses another port, the URL must specify the port number in addition to the server name Eg: http://www.university.com:8080/index.html

This request is sent to a server that uses port 8080 instead of 80.

- The browser uses the URL information to create the request message it sends to the specified server using the specified protocol.
- An HTTP request message consists of three things.
  - ❖ A Request Line
  - ❖ Request Headers
  - ❖ Request Body
- The Request Line starts with the request method name, followed by a resource identifier and the protocol version used by the browser **GET/index.html HTTP/1.1**
- The most commonly used request method is named GET. As the name implies a GET request is used to retrieve a resource from the server, it's the default request method, so if you type a URL in the browser's address field the request is sent as a GET request to the server.
- The Request headers provide additional information the server may use to process the request
- Some of the Request header field names are host ,user-agent, accept, accept language etc.
- Eg: of a valid HTTP Request Message

  GET/index.html HTTP/1.1
  Host: www.university.com
  User-Agent: Mozilla/5.0
  Accept : image/gif, image/jpeg
  Accept-Language : English
- The Request Line specifies the GET Method and asks for the resources named index.html to be returned using the HTTP/1.1 protocol version.
- The Host header tells the server the hostname used in the URL
- The User-Agent header contains information about the type of browser making the request.
- The Accept headers provide information about the languages and file formats the browser accepts



- The majority of HTTP Request messages use the GET method. When you request an URL in a web browser , the Get method is used , a GET request does not have a body.

  When you send an HTML form, either GET or POST can be used, with a GET Request the parameter are encoded in the URL, with a POST request they are transmitted in the body.

The Main HTTP Request Methods are:

1) **GET:** Retrieve the resource the user has requested
2) **POST:** Transfers data to the specified resource

3) **HEAD:** Similar to GET but forces the server to return only on HTTP header instead of response data.
4) **PUT:** Uploads the resource to the server
5) **DELETE:** Deletes the resource form the server

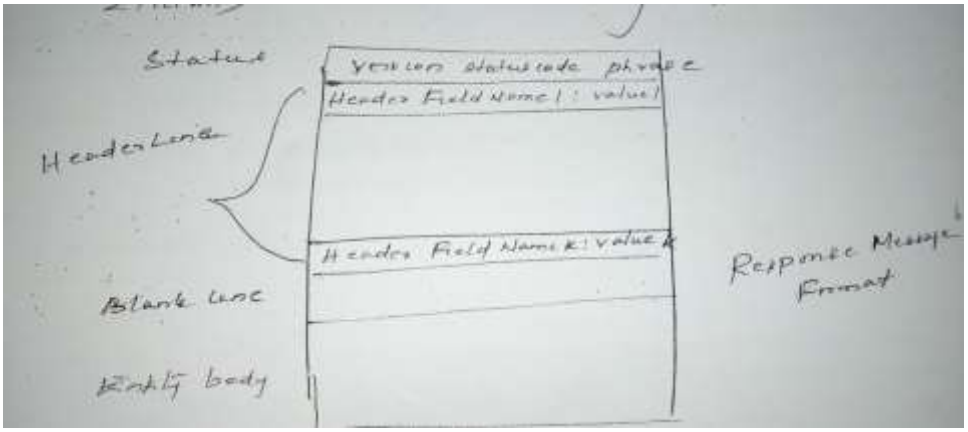
HTTP RESPONSE

- ✓ When the web server receives the request, if looks at the URL and decides ,based on configuration information, how to handle if.
- ✓ It may handle it internally by simple reading an HTML file from the file system, or it can forward the request to some component that is responsible for the resource corresponding to the URL.
- ✓ The response message looks similarly to the request message. It consists of three things
- ✓ A Status Line
- ✓ Response Headers and
- ✓ An Optional Response Body
- ✓ The status line starts with the name of the protocol, followed by a status code and a short description of the status code. Here the status code is 200, meaning the request was executed successfully.
- ✓ Various Response Headers are
- ✓ **Last-Modified** header gives the data and time for when the resource was last modified
- ✓ **Content-Type** header tells the browser what type of response data the body contains and
- ✓ **Content-Length** header tells how large if is e.g.

HTTP:/1.1 200 ok
Last-Modified: Mon, 20 Dec 2002 23.2.42 GMT
Date: Tue, 11 Jan 2003 20:52:40 GMT
Status: 200
Content-Type: text/html
Servlet-Engine: Tomcat Web server/5.0
A blank line separates the headers from the message body

```
<html>
        <body>
                <h1>Hello World</h1>
        </body>

</html>
```



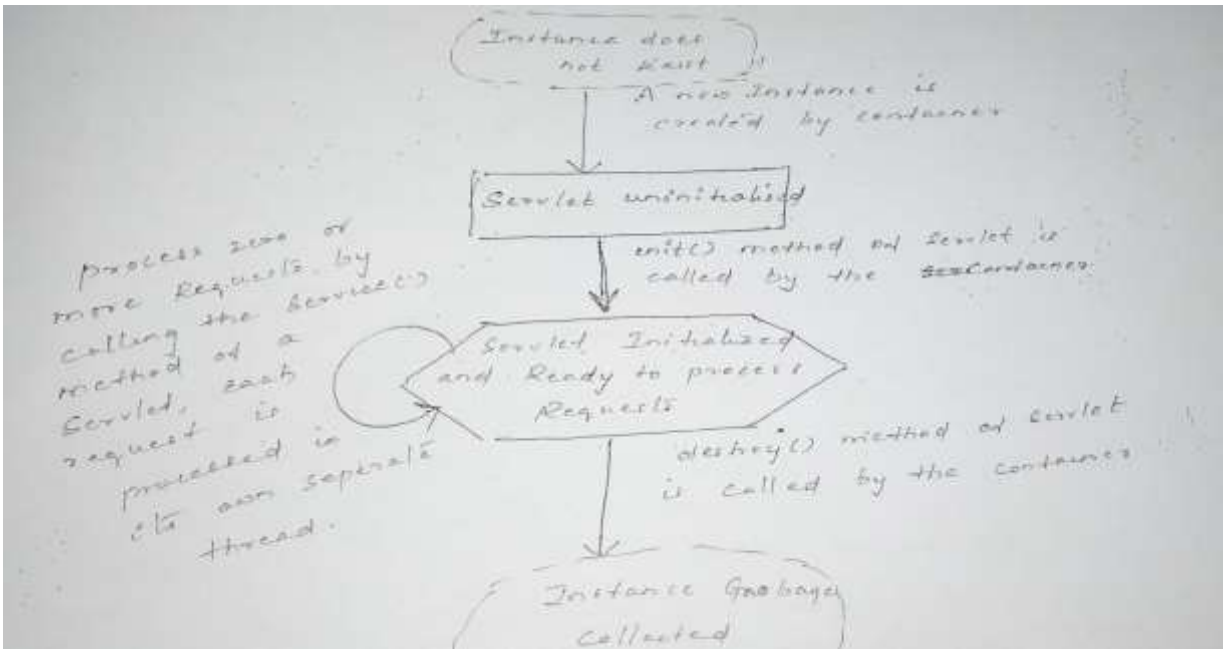
- ✓ Date header indicates the time and date when the HTTP response was created and sent by the server.
- ✓ Servlet-Engine header indicates that the message was generated by Tomcat.

LIFE CYCLE OF A SERVLET

Three methods are central to the life cycle of a servlet. These are:

1. **Init()**
2. **Service()**
3. **Destroy()**

✓ Life cycle of a servlet describes how and when a servlet is loaded, initialized, handles request and unloaded.

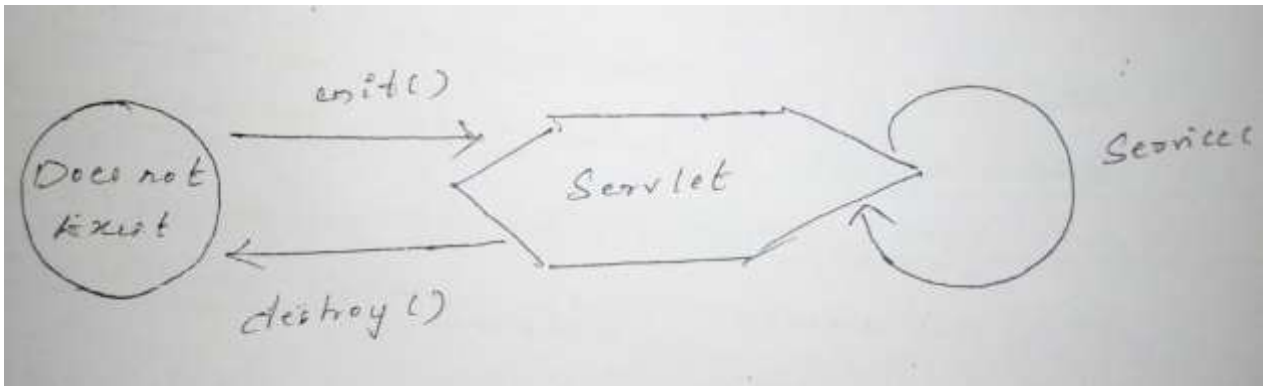


✓ The servlet life cycle consists of the following steps.

1. The client enter the URL in the web browser and makes a request
2. The browser then generates the HTTP request and sends it to the web server.
3. Whenever the web server receives a HTTP request for a resource, web server determines if the request should be handled by a servlet

4. If the Request is to be handled by a servlet, the web container checks to see if the instance of the servlet is already available. If an instance is available the container delegates the request to that instance. If no instance of the servlet is available, container creates a new instance of the class of servlet and delegates the request to that newly created.
5. The container calls the **init ()** method this method initializes the servlet and must be called before the servlet can service any request. In the entire life of a servlet, the **init ()** method is called only once.
6. After initialization the servlet can service client Request. Each Request is serviced in its own separate threads. The container calls the **service()** method of the servlet for every Request.
7. And finally if the servlet is not needed any more, the container calls the **destroy()** method which takes the servlet out of service. The **destroy()** method is also called only once in the life cycle of a servlet.

Life Cycle of a Servlet (simplified)



A SIMPLE SERVLET PROGRAM

A Simple servlet program

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet
{
        public void service(ServletRequest Req, ServletResponse Res)throws ServletException, IOException
        {
                Res.setContentType("text/html");
                PrinterWriter pw=res.getWriter();
                pw.println("<b>Hello Servlet World");
                pw.close();
        }
}
```

- ✓ First, note that it imports the javax.servlet package. This package contains the classes and interfaces required to build servlets.

- ✓ All servlets must implement the servlet interface. It declares the init(), service(), and destroy() methods that are called by the server during the Life Cycle of a Servlet. The GenericServlet class provides implementations of the basic life cycle methods [init(),service(),destroy()] for a servlet.
- ✓ So the program HelloServlet defined as subclass of Generic Servlet .Inside HelloServlet, the service() method is overridden. This method handles client request.

  The first argument of service() method is ServletRequest object. This enables the servlet to read data that is provided via the client Request.

  The Second argument of service() method is ServletResponse object. This enables the servlet to formulate a response for the client.

  The setContentType() indicates that the browser should interpret the content as HTML source code.
- ✓ getWriter () method of ServletResponse interface returns a PrintWriter object that can be used to write character data to the response i.e. getWriter() method obtains a PrintWriter. Anything written to this stream is sent to the client as part of the HTTP response. Then println() is used to write some simple HTML source code as the HTTP response.

JAVA SERVLET DEVELOPMENT KIT (JSDK)

- ✓ JSDK is a kit for developing servlets. It contains the servlet API classes and tools to execute and test servlets. Tomcat Replaces the old JSDK
- ✓ It contains the class libraries, documentation, and runtime support that you will need to create and test servlets.

THE SERVLET API

- ✓ Two packages contain the classes and interfaces that are required to build servlets. These are:
  1. javax.servlet
  2. javax.servlet. Http

They constitute the servlet API

- ✓ These packages are not part of the Java core packages. Instead, they are standard extensions provided by Tomcat. Therefore they are not include with Java SE6.
- ✓ Protocol independent classes and interfaces execute in the package javax.servlet whereas HTTP specific classes and interfaces exist in javax.servlet. Http
- ✓ The servlet API contains -20 interfaces (13 of javax.servlet and 7 of javax.servlet. Http)

  -15 classes (9 of javax.servlet and 6 of javax.servlet. Http)

THE JAVAX.SERVLET PACKAGE

- ✓ Protocol independent classes and interfaces exists in the package javax.servlet
- ✓ The core interfaces of this packages are
  1) Servlet
  2) ServletConfig
  3) ServletRequest

4) ServletResponse

Servlet Interface

- ✓ Contains the declaration of servlet life cycle methods. All servlet must implement this interfaces.
- ✓ It declares the init(), service() and destroy() methods that are called by the server during the life cycle of a servlet. The methods defined by servlet

Method	Description
1. Void init(ServletConfig sc)throws ServletException	Called when the servlet is initialized. Initialization parameter for the servlet can be obtained from sc An unavailable Exception should be thrown if the servlet cannot be initialized
2. ServletConfig get ServletConfig()	Returns a ServletConfig object that contains any initialization parameters
3. Void service(ServletRequest req, ServletResponse res)throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from req. The response to the client can be written to res. An exception is generated if a servlet or IO problem occurs
4. Void destroy()	Called when the servlet is unloaded
5. String getServletInfo()	Returns a string describing the servlet

ServletConfig Interface

- ✓ The ServletConfig interface allows a servlet to obtain configuration data when it is loaded
- ✓ The methods declared by this interface are:
  1) **String getInitParameter(string param):** This method returns the value of the parameter named "param"
  2) **Enumeration getInitParameterNames():**Returns an enumeration of all initialization parameter names.
  3) **String getServletName():**Returns the name of the invoking servlet

The ServletRequest Interface

- ✓ The ServletRequest interface enables a servlet to obtain information about a client request
- ✓ Several of it methods are:
  1) **String getParameter(string pname):**Returns the value of the Requested parameter named "pname"
  2) **Enumeration getParameterNames():** Returns an enumeration of the requested parameter names for this request.
  3) **String getSchema() :**Return the transmission scheme of the URL used for the request(for e.g : "http","ptp")
  4) **String getServerName():**Returns the name of the server.
  5) **String getProtocol():**Returns a description of the protocol.

The ServletResponse Interface

- ✓ The ServletResponse interface enables a servlet to formulate a response for a client.
- ✓ Several of its methods:
    1) **PrintWriter getWriter() throws IOException:** Return a PrintWriter that can be used to write character data to the response. An illegal StateException is thrown if getOutputStream() has already been involved for this Request.
    2) **void setContentLength(int size):** Sets the content length for the response of size.
    3) **void setContentType(string type):** Sets the content type for the response of type.

The core classes in this packages are

- GenericServlet
- ServletInputSteam
- ServletOutputStream
- ServletException
- UnavailableException

1) **GenericServlet :** GenericServlet class provides implementation of the basic lige cycle methods for a servlet.
   -GenericServlet implements the servlet and ServletConfig interfaces
   -In addition, a method to append a string to the server log file is available
   -The signatures of this method are shown here
   Void log(string s)
   Void log(string s , throwable e)
   -Here, s is the string to be appended to the log and e is an exception that occurred.
2) **The ServletInputStream class:** The ServletInputStream class extends InputStream
   -It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request
3) **The ServletOutputStream class:**The servletOutputStream class extends OutputStream
   -It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response.
   -It also defines the print() and println() methods, which output data to the stream.
4) **The ServletException class:** javax.servlet defines two exceptions.
   -The first is ServletException which indicates that a servlet problem has occurred
   -The second is UnavailableException which extends ServletException. If indicates that a servlet is unavailable.

The javax.servlet.http package

- ✓ Protocol independent classes and interfaces exists in javax.servlet whereas Http specifies classes and interfaces exists in javax.servlet.http package
- ✓ The core interfaces that are provided in this package are
    A. HttpServletRequest

B. HttpServletResponse

C. HttpSession

HttpServletRequest interface

- ✓ This interface enables a servlet to obtain information about a client request.
- ✓ Some of the methods in this interface are
  - **a) String getHeader(string field):** Returns the value of the header field named field
  - **b) Enumeration getHeaderNames():** Returns an enumeration of the header names.
  - **c) String getMethod():**Returns the HTTP method for this Request.
  - **d) String getQueryString():** Returns any query string in the URL.
  - **e) String getRemoteUser():** Returns the name of the user who used this request
  - **f) String getRequestSessionId():** Returns the ID of the session
  - **g) String Buffer getRequestURL():** Returns the URL
  - **h) HttpSession getSession():**Returns the session for this request, if a session does not exist, one is created and then returned.
  - **i) Boolean is RequestedSessionIdFromCookie():** Return true if a cookie contains the session ID,otherwise returns false.

HttpServletResponse Interface

- ✓ This interface enables a servlet to formulate an HTTP response to a client
- ✓ Several constants are defined. These correspond to the different status
  -For eg: sc-ok indicates that the HTTP request succeeded
- ✓ Some of the methods in this interface are
  - **a) boolean containsHeader(String field):**Return true if the HTTP response header contains a field named field
  - **b) void sendError(int c, string s) throws IOException:** Sends the error code c and message s to the client
  - **c) void semdRedirect(string url):** Redirect the client to url
  - **d) void setDateHeader(string field,long):** Adds field to the header with date value equals to m
  - **e) void setHeader(String field, string val):** Adds field to the header with value equal to val
  - **f) void setStatus(int code):** set the Status code for this response to code

The HttpSession interface

- ✓ The HttpSession interface enables a servlet to read and write the state information that is associated with an Http session
- ✓ Some of the methods in this interface are:
  - **a) Enumeration getAttributeNames():** Returns an enumeration of the attribute names associated with the session
  - **b) Long getCreationTime():** Return the time when this session was created
  - **c) String getId():** Return the session ID
  - **d) Long getLastAccessedTime():**Returns the time when the client last made a request for this session

e) **Void removeAttribute(String attr):** Removes the attribute specified by attr from the session

f) **Void setAttribute(String attr,object val):**Associates the value passed in val with the attribute name passed in attr.

The core classes that are provided in this package are:

A. Cookie

B. HttpServlet

C. HttpServletEvent

Cookie class

- ✓ The cookie class encapsulates a cookie. A cookie is stored on a client and contains state informations.
- ✓ Cookie are valuable for tracking user activities
- ✓ Important methods in cookie class are:

a) **String getComment():**Returns the comment

b) **String getDomain():** Returns the domain

c) **int getMaxAge():** Returns the maximum age(in seconds)

d) **string getName():** Returns the name

e) **string getValue():**Returns the value

f) **void setMaxAge(int secs):**sets the maximum age of the cookie to sees. This is the no.of seconds after which the cookie is deleted.

The HTTPServlet class

- ✓ The Httpservlet class extends GenericServlet
- ✓ It is commonly used when developing servlets that receive and process HTTP requests
- ✓ Core methods of this class are:

a) void doDelete(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException: Handles an HTTP Delete Request

b) void doGet(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException: Handles an HTTP Get Request

c) **void doPost(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException:** Handles an HTTP Post Request

HttpSessionEvent

- ✓ HttpSessionEvent encapsulates session events
- ✓ It extends EventObject and is generated when a change occurs to the session
- ✓ This class defines only one methods

a) **HttpSession getSession():** Returns the session in which the event occurred

Session Tracking

- HTTP is a stateless, request-response protocol.

This means when browser sends a request to the server, the server processes it and sends the response to the browser and does not remember anything about the request.

So when browser sends the same request to the server, server take it as a new request process it once again and then sends it as response to browser. If there are all static web pages then this doesn't matter much. But if there is a interactive web application (such as online shopping) then it is required that server should keep track of the user or the request made by the user.

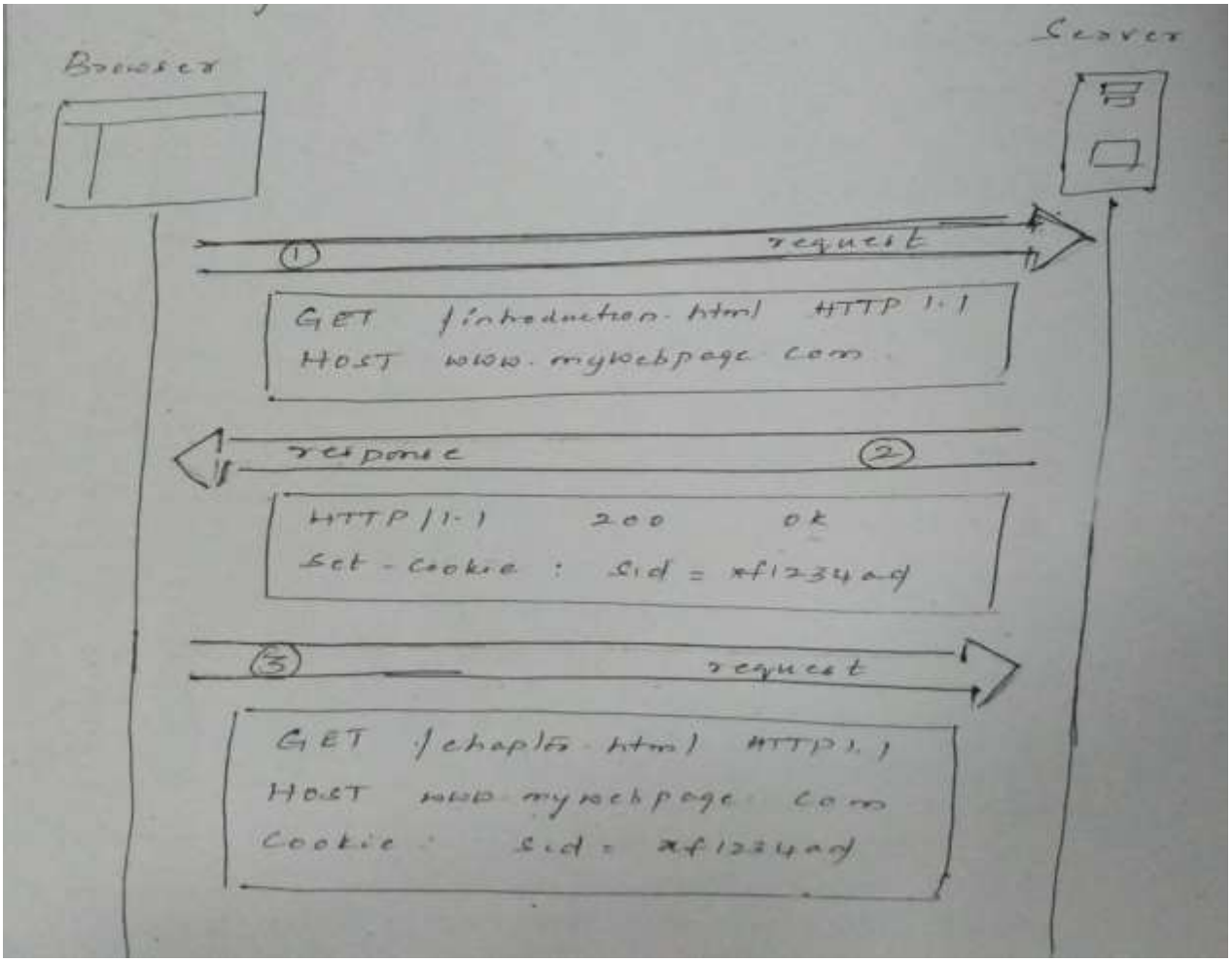
- To solve the problem there are three methods that are normally used:
    1. Use of cookies
    2. Embedding Hidden fields in a HTML form
    3. Sending URL string in response body
- For sending all state information to and fro between browser and server, usually an ID is used. This ID is basically a Session-ID.

  Thus session-ID is passed between the browser and server while processing the information.

  This method of keeping track of all the information between server and browser using session-ID is called **Session Tracking**.

Use of Cookies

- A Cookie is a name-value pair information. This information is passed from server to browser in response header
- The browser s then returns these cookies unchanged to the server by including the state.
- By returning a cookie to a web server, the browser provides the server a means of connection the current page view with previous page view.
- Use of session-ID in session tracking using the cookies can be illustrated as.



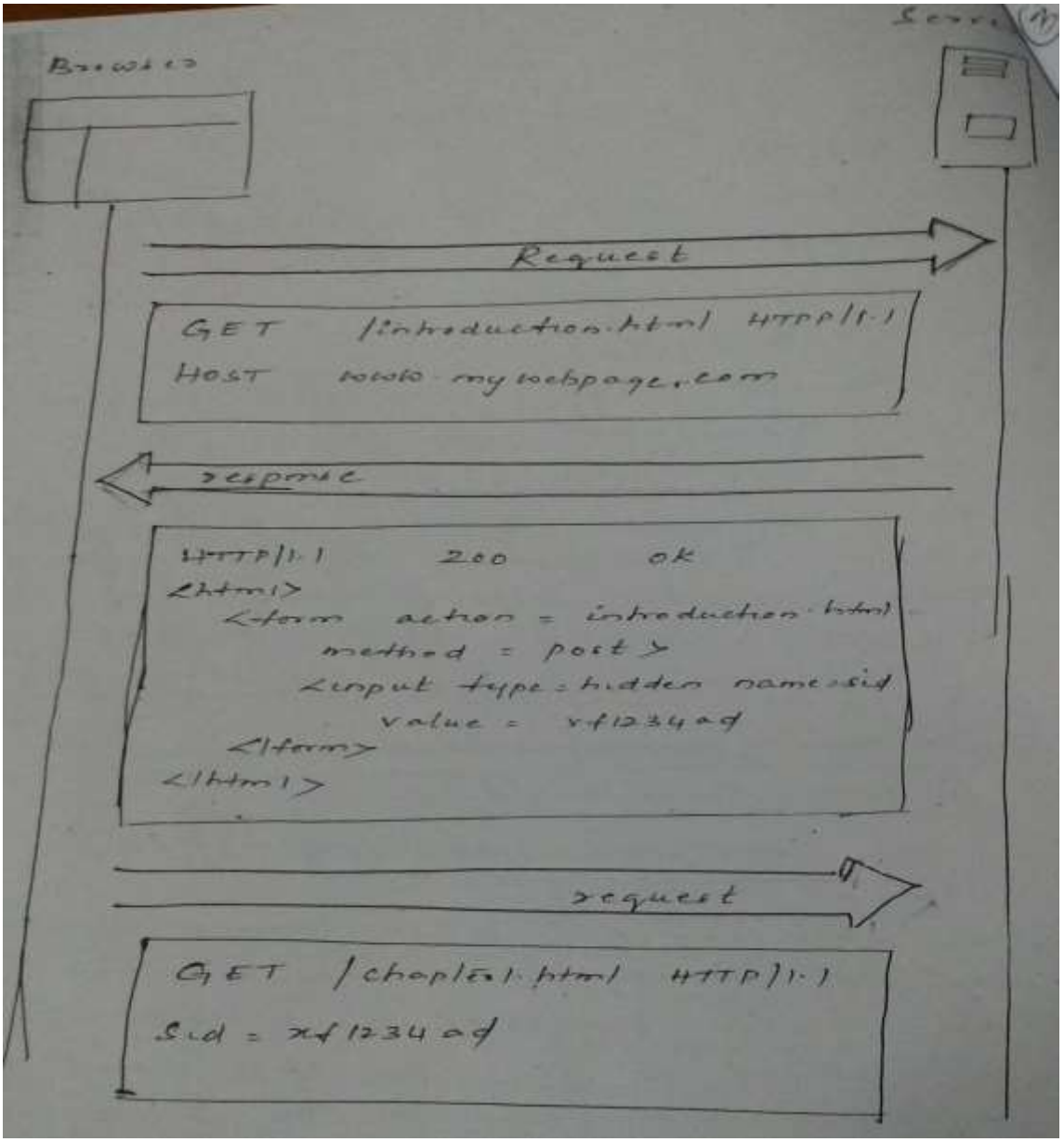
1. At first the browser connects to the server www.mywebpage.com by making a request
2. The server then replies in the form of HTTP response. Using set-cookies statement server is requesting browser to store the sid=x+1234ad.

   So if browser supports cookies then every subsequent page request to the same server will contain the cookies.

3. This is another request to the same server. By including cookies which contains sid=x+1234as, server knows that this request is related to the previous one. Thus server- browser can keep track of current session.

Hidden Form Field

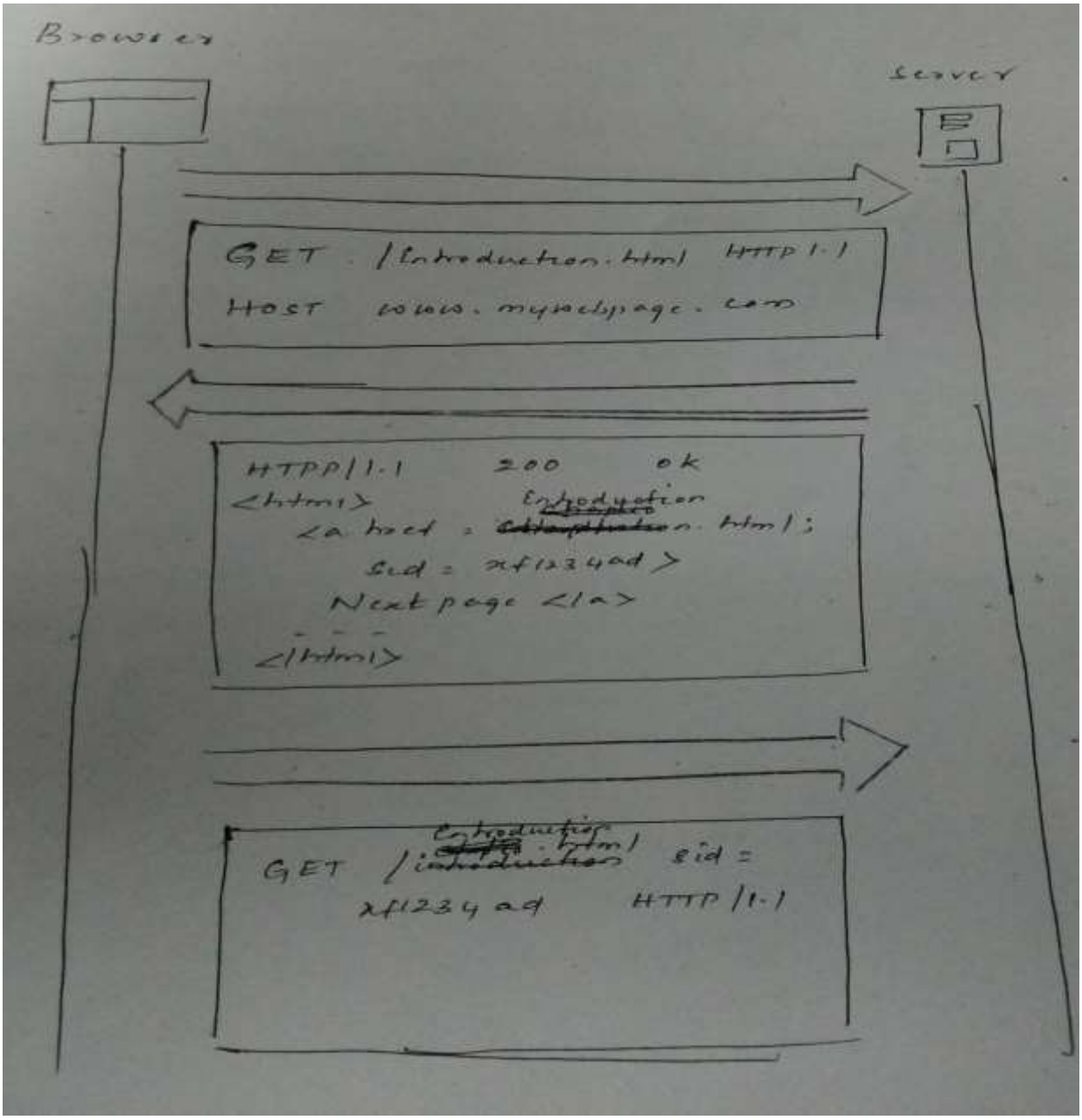
- The ASP.NET uses this method of session tracking.
- In this method, when browser makes a request to the server for some web page then server responds by inserting hidden form field in the html page, and then sending that page to the browser.



- The hidden contents will not be displayed on web browser. But those fields will be stored with browser and when browser makes another request to the same server.
- By hidden field sid=x+1234ad, server will come to know that this request is related to the previous one.

URL rewriting

- This is more precise technique in which information is embedded into URL.
- In URL rewriting, we append an identifier to the URL of the next resource.
- It will always work even the browser does not support cookies or when cookies are disabled.



- When browser makes request to the server, server responds by putting sid=x+1234ad in response body by rewriting the url as introduction html; sid=x+1234ad.
- When browser makes request for another page same URL is embedded. There by server will come to know that the request is related to previous page and the work in same session will get correlated.
- A session starts when the browser makes the first request for a JSP page in a particular application.
- The application can explicitly end the session or the JSP container can end it after a period of user activity (default value is typically 30 minutes after the last request) or closing a browser usually means losing the session.
- When the user opens a browser again, the server can't associate the new request with the previous session, and therefore cicadas a new session.

COOKIES

- ✓ A Cookies are the most commonly used means of tracking client session

Definition: A Cookie is a small piece of textual information sent by the server to the client and returned by the client for all requests to the server.

Or

Definition: A cookie is a piece of text that a web server can store on a user's hard disk.
Cookies allow a website to store information on a user's machine and later retrieve it. The pieces of information are stored as name-value pairs.

- ✓ A cookie contains one or more name-value pairs with certain additional attributes, which are exchanged in the response and request headers
- ✓ Web servers send a cookie by sending the set-cookie response header in the following format:

Set-cookie: Name=VALUE; comment=COMMENT;

Domain: DOMAINNAME; Max-Age; SECONDS;

Path :PATH; SECURE; version:1*DIGIT

where, **Name** specifies the name of the cookies

value specifies the sessionID

Max-Age specifies maximum life of the cookie in seconds

The domain and path of the cookie determine when it is included in the header of an HTTP request. If the user enter a URL whose domain and path match these values the cookie is then supplied to the web server otherwise it is not.

EG: set-cookie: uid=joe; Max-Age=3600; Domain="mysever.com"; path="/"

The above response header sends a cookie with name uid and value joe. The lifetime of this cookie is 3600 seconds and is valid for the myserver.com domain for the url path.

- ✓ Cookies are also used for tracking state information. For eg: assume a user visits an online store. A cookie can save the user's name, address and other information. The user doesn't need to enter this data each time he or she visits the store.

UNIT -III

1. **History**
2. **General Features**
3. **PHP Basics**
   - **3.1 Code embedding web pages**
   - **3.2 Commenting the Code**
   - **3.3 Output Data to Browser**
   - **3.4 Data Types**
   - **3.5 Identifiers**
   - **3.6 Variables**
   - **3.7 Constants**
   - **3.8 Expressions**
   - **3.9 String Interpolation**
   - **3.10 Arrays**

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use.
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"
- PHP can generate dynamic page content.
- 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 is easy to learn and runs efficiently on the server side

---

1. The History of PHP

- PHP is an "HTML-embedded scripting language" primarily used for dynamic Web applications. The first part of this definition means that PHP code can be interspersed with HTML, making it simple to generate dynamic pieces of Web pages on the fly.
- PHP takes most of its syntax from C, Java, and Perl. It is an open source technology and runs on most operating systems and with most Web servers.

- PHP was written in the C programming language by Rasmus Lerdorf in 1994 for use in monitoring his online resume and related personal information.
- For this reason, PHP originally stood for "Personal Home Page". Lerdorf combined PHP with his own Form Interpreter, releasing the combination publicly as PHP/FI (generally referred to as PHP 2.0) on June 8, 1995.
- Two programmers, Zeev Suraski and Andi Gutmans, rebuilt PHP's core, releasing the updated result as PHP/FI 2 in 1997. The acronym was formally changed to PHP: HyperText Preprocessor, at this time. (This is an example of a recursive acronym: where the acronym itself is in its own definition.)
- In 1998, PHP 3 was released, which was the first widely used version. PHP 4 was released in May 2000, with a new core, known as the Zend Engine 1.0. PHP 4 featured improved speed and reliability over PHP 3. In terms of features, PHP 4 added references, the Boolean type, COM support on Windows, output buffering, many new array functions, expanded object-oriented programming, inclusion of the PCRE library, and more. Maintenance releases of PHP 4 are still available, primarily for security updates.
- PHP 5,Although previous major releases had enormous numbers of new library additions, version 5 contained improvements over existing functionality and added several features commonly associated with mature programming language architectures like vastly improved object-oriented capabilities, Try/Catch exception handling, Improved XML and web services support etc.. The enhanced object-oriented capabilities introduced in PHP 5 resulted in an additional boost for the language.
- PHP 5.3, Although officially a point release, PHP 5.3 is actually the most significant upgrade to the language since the release of 5.0. Heralding a powerful array of new features including namespaces, late static binding, lambda functions and closures, a new MySQL driver, version 5.3 represents a serious step forward in PHP's evolution.
- PHP 6, A new major version of PHP known as PHP 6 has been concurrently developed alongside PHP 5.X for several years, with the primary goal of adding Unicode support to the language. However, in March, 2010 the development team decided to primarily focus on the 5.X series of releases. In fact, several features originally slated for PHP 6 have been integrated into 5.X releases. Although PHP 6 beta releases had previously been made available at **http://snaps.php.net**, at the time of this writing it appears as if those releases have been removed from the PHP website.

2. General Language Features:

Features of PHP

2.1 Practicality
2.2 Power
2.3 Possibility
2.4 Price

2.1 Practicality

From the very start, the PHP language was created with ***practicality*** in mind. The result is a language that allows the user to build powerful applications even with a minimum of knowledge.

For instance, a useful PHP script can consist of as little as one line; unlike C, there is no need for the mandatory inclusion of libraries. For example, the following represents a complete PHP script, the purpose of which is to output the current date, in this case one formatted like **September 23, 2007**:

<?php echo date("F j, Y"); ?>.

Another example of the language's penchant for compactness is its ability to nest functions. For instance, you can effect numerous changes

to a value on the same line by stacking functions in a particular order. The following example produces a string of five alphanumeric characters such as **a3jh8**:

$randomString = substr(md5(microtime()), 0, 5);

PHP is a *loosely typed* language, meaning there is no need to explicitly create, typecast, or destroy a variable, although you are not prevented from doing so. PHP handles such matters internally, creating variables on the fly as they are called in a script, and employing a best-guess formula for automatically typecasting variables. For instance, PHP considers the following set of statements to be perfectly valid:

<?php

```
$number = "5"; // $number is a string
$sum = 15 + $number; // Add an integer and string to produce integer
$sum = "twenty"; // Overwrite $sum with a string.
?>
```

2.2 Power

PHP's has ability to interface with databases, manipulate form information, and create pages dynamically, besides these PHP can also do the following

- Create and manipulate Adobe Flash and Portable Document Format (PDF) files.
- Evaluate a password for guess ability by comparing it to language dictionaries and easily broken patterns.
- Parse even the most complex of strings using the POSIX and Perl-based regular expression libraries.
- Authenticate users against login credentials stored in flat files, databases, and even Microsoft's Active Directory.
-

2.3 Possibility

PHP developers are rarely bound to any single implementation solution. On the contrary, a user is typically fraught with choices offered by the language.

For example, consider PHP's array of database support options. Native support is offered for more than 25 database products, including Adabas D, dBase, Empress, FilePro, FrontBase, Hyperwave, IBM DB2, Informix, Ingres, InterBase, mSQL, Microsoft SQL Server, MySQL, Oracle, Ovrimos, PostgreSQL, Solid, Sybase, Unix dbm, and Velocis

PHP's flexible string-parsing capabilities offer users of differing skill sets the opportunity to not only immediately begin performing complex string operations but also to quickly port programs of similar functionality (such as Perl and Python) over to PHP. In addition to almost 100 string-manipulation functions, Perl-based regular expression formats are supported.

PHP offers support for both procedural and Object oriented Programming

2.4 Price

PHP is available free of charge. i.e. PHP is a Open source software. In recent years, software meeting such open licensing qualifications has been referred to as *open source software*. Open source software and the Internet go together like bread and butter.

3. PHP BASICS

3.1 Code embedding web pages
3.2 Commenting the Code
3.3 Output Data to Browser
3.4 Data Types
3.5 Identifiers
3.6 Variables
3.7 Constants
3.8 Expressions
3.9 String Interpolation
3.10 Arrays

3.1 Code Embedding Web Pages

- One of PHP's advantages is that you can embed PHP code directly alongside HTML.
- The web server will send only those pages that have .php extension to the PHP Engine .
- It is highly inefficient for the engine to consider every line as a potential PHP command.
- Therefore, the engine needs some means to immediately determine which areas of the page are PHP-enabled.
- This is logically accomplished by delimiting the PHP code.
- There are four delimitation variants.
  - 3.1.1 Default Syntax
  - 3.1.2 Short –Tags
  - 3.1.3 Script

3.1.4 ASP Style

3.1.1 Default syntax

The default delimiter syntax opens with <?php and concludes with ?>, like this:

3.1.2 Short-Tags

- For less motivated typists, an even shorter delimiter syntax is available. Known as *short-tags*.
- However, to use this feature, you need to enable PHP's short_open_tag directive.
- An example follows:

```
<?
print "This is another PHP example.";
?>
```

- When short-tags syntax is enabled and you want to quickly escape to and from PHP to output a bit of dynamic text, you can omit these statements using an output variation known as ***short-circuit syntax***

```
<?="This is another PHP example.";?>
```

This is functionally equivalent to both of the following variations:

```
<? echo "This is another PHP example."; ?>
<?php echo "This is another PHP example.";?>
```

3.1.3 Script

Certain Editors have problems dealing with PHP's more commonly used escape syntax variants. To Solve such issues, another delimiter variant, <script>, is offered.

```
<script language="php">
    print "This is another PHP example.";
</script>
```

3.1.4 ASP Style

- Microsoft ASP pages employ a delimiting strategy similar to that used by PHP.
- ASP Style includes opening dynamic syntax with <%, and concluding with %>.
- If you're coming from an ASP background and prefer to continue using this escape syntax, PHP supports it. Here's an example:

```
<%
    print "This is another PHP example.";
%>
```

- The ASP Style and Script delimiting variants are rarely used and should be avoided unless you have ample reason for doing so. **Caution** ASP Style syntax is no longer available as of PHP 5.3.

Embedding Multiple Code Blocks

You can escape to and from PHP as many times as required within a given page. For instance, the following example is perfectly acceptable:

```
<html>
    <head>
        <title><?php echo "Welcome to my web site!";?></title>
    </head>
```

```
<body>
    <?php
        $date = "July 26, 2010";
    ?>
    <p>Today's date is <?=$date;?></p>
</body>
</html>
```

3.2 Commenting Your Code

- The main purpose of comments is to serve as a note to the developer and the maintenance people
- A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
- Comments can be used to
  - Let others understand what you are doing
  - Remind yourself of what you did.
- While there is only one type of comment in HTML, PHP has three types.
  1. Single-Line C++ Syntax
  2. Shell syntax
  3. Multiple-Line C Syntax

3.2.1 Single-Line C++ Syntax

The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. PHP supports C++ single-line comment syntax, which is prefaced with a double slash (//), like this:

```
<?php
// Title: My first PHP script
// Author: Jason Gilmore
echo "This is a PHP program.";
?>
```

3.2.2 Shell syntax

PHP also supports an alternative to the C++-style single-line syntax, known as *shell syntax*, which is prefaced with a hash mark (#).

```
<?php
# Title: My first PHP script
# Author: Jason Gilmore
echo "This is a PHP program.";
?>
```

3.2.3 Multiple-Line C Syntax

The multi-line PHP comment can be used to comment out large blocks of code or writing multiple line comments. The multiple line PHP comment begins with " /* " and ends with " */ ".

```
<?php
/* This Echo statement will print out my message to the
the place in which I reside on. In other words, the World. */
echo "Hello World!";
/* ec
ho "My name is Humperdinkle!";
echo "No way! My name is Uber PHP Programmer!";
*/
?>
```

3.3 Output Data to Browser

PHP offers several methods for dynamic web sites to output data to the browser

- 3.3.1 The print() Statement
- 3.3.2 The echo() Statement

3.3.3 The printf() Statement
3.3.4 The sprintf() Statement

3.3.1 The print() Statement

- The print() statement outputs data passed to it . Its prototype looks like this:

int print(argument)

```
<?php
print("<p>I love the summertime.</p>");
?>
<?php
$season = "summertime";
print "<p>I love the $season.</p>";
?>
<?php
print "<p>I love the
summertime.</p>";
?>
```

- All these statements produce identical output:

I love the summertime

- The print() statement's return value is misleading because it will always return 1 regardless of Outcome

3.3.2 The echo() Statement

- Alternatively, use the echo() statement for the same purposes as print().
- **void echo(string argument1 [, ...string argumentN])**
- To use echo(), just provide it with an argument just as was done with print():
- **echo "I love the summertime.";**
- echo() is capable of outputting multiple strings

```
<? php
        $heavyweight = "Lennox Lewis";
        $lightweight = "Floyd May weather";
        echo $heavyweight, " and ", $lightweight, " are great fighters.";
?>
```

- This code produces the following:
  Lennox Lewis and Floyd May weather are great fighters.
- Executing the following (in my mind, more concise) variation of the above syntax produces the same output:
  echo "$heavyweight and $lightweight are great fighters.";

Note: Which is faster, echo() or print()? The answer is that the echo() function is a tad faster because it returns nothing, whereas print() will return 1 if the statement is successfully output.

3.3.3 The printf() Statement

- The printf() statement is ideal when you want to output a blend of static text and dynamic information stored within one or several variables.
- It's ideal for two reasons.
  - First, it neatly separates the static and dynamic data into two distinct sections, allowing for easy maintenance.
- Second, printf() allows you to wield considerable control over how the dynamic information is rendered to the screen in terms of its type, precision, alignment, and position.
- Its prototype looks like this:
  integer printf(string format [, mixed args])
- For example, suppose you wanted to insert a single dynamic integer value into an otherwise static string:
  printf ("Bar inventory: %d bottles of tonic water.", 100);
- Executing this command produces the following:

Bar inventory: 100 bottles of tonic water

- In this example, *%d* is a placeholder known as a *type specifier*, and the *d* indicates an integer value will be placed in that position.

Type Description

Type	Description
%b	Argument considered an integer; presented as a binary number
%c	Argument considered an integer; presented as a character corresponding to that ASCII value
%d	Argument considered an integer; presented as a signed decimal number
%f	Argument considered a floating-point number; presented as a floating-point number
%o	Argument considered an integer; presented as an octal number
%s	Argument considered a string; presented as a string
%u	Argument considered an integer; presented as an unsigned decimal number
%x	Argument considered an integer; presented as a lowercase hexadecimal number
%X	Argument considered an integer; presented as an uppercase hexadecimal number

Another Example: printf ("%d bottles of tonic water cost $%f", 100, 43.20);

Executing this command produces the following:

100 bottles of tonic water cost $43.200000

3.3.4 The sprintf() Statement

- The sprintf() statement is functionally identical to printf() except that the output is assigned to a string rather than rendered to the browser. The prototype follows:

  string sprintf(string format [, mixed arguments])

- An example follows:

```
$cost = sprintf("$%.2f", 43.2); // $cost = $43.20
```

3.4 PHP's Supported Data Types

3.4.1 Scalar Data Type
3.4.2 Compound Data Types
3.4.3 Converting Between Data Types Using Type Casting
3.4.4 Type-Related Functions
3.4.5 Type Identifier Functions

A *data type* is the generic name assigned to any data sharing a common set of characteristics.

3.4.1 Scalar Data Types

- *Scalar* data types ***are used to represent a single value***.
- Several data types fall under this category, including ***Boolean, integer, float, and string***.

Boolean:

- A Boolean represents two possible states: TRUE or FALSE.
- Alternatively, you can use zero to represent FALSE, and any nonzero value to represent TRUE.

```
$alive = false; // $alive is false.
$alive = 1; // $alive is true.
$alive = -1; // $alive is true.
$alive = 5; // $alive is true.
$alive = 0; // $alive is false.
```

Integer:

- An *integer* is representative of any whole number or, in other words, a number that does not contain fractional parts.

- PHP supports integer values represented in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal) numbering systems

```
42          // decimal
-678900     // decimal
0755        // octal
0xC4E       // hexadecimal
```

Float:

- Floating-point numbers, also referred to as *floats*, *doubles*, or *real numbers*, allow you to specify numbers that contain fractional parts.
- Floats are used to represent monetary values, weights, distances etc ..
- PHP's floats can be specified in a variety of ways, several of which are demonstrated here:

```
4.5678
4.0
8.7e4
1.23E+11
```

String:

- Simply put, a string is a sequence of characters treated as a contiguous group.
- *Strings* are delimited by single or double quotes.
- The following are all examples of valid strings:

```
"PHP is a great language"
"Whoop-de-do"
'*9subway\n'
"123$%^789"
```

3.4.2 Compound Data Types

- *Compound data types* allow for multiple items of the same type to be aggregated under a single representative entity.
- The ***array and the object*** fall into this category.

Object:

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First class of object must be declared
- For this, use the class keyword.
- A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
   function Car() {
      $this->model = "VW";
   }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

Array:

- An Array is a variable which holds collection of homogeneous values.
- In the following example $cars is an array.
- The PHP var_dump() function returns the data type and value:

Example

```
<html>
    <body>
    <?php
        $cars = array("Volvo","BMW","Toyota");
        var_dump($cars);
        echo("<br>");
        $str = "hello";
        var_dump($str);
    ?>
    </body>
</html>
```

Output:
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
string(5) "hello"

3.4.3 Converting Between Data Types Using Type Casting

- Converting values from one datatype to another is known as *type casting*.
- A variable can be evaluated once as a different type by casting it to another.
- This is accomplished by placing the intended type in front of the variable to be cast.
- A type can be cast by inserting one of the operators shown in below Table in front of the variable.

Type Casting Operators

(array)	Array
(bool) or (boolean)	Boolean
(int) or (integer)	Integer
(object)	Object
(real) or (double) or (float)	Float
(string)	String

Suppose to cast an integer as a double:

```
$score = (double) 13; // $score = 13.0
```

Type casting a double to an integer will result in the integer value being rounded down, regardless of the decimal value.

```
$score = (int) 14.8; // $score = 14
```

3.4.4 Type-Related Functions

- A few functions are available for both verifying and converting data types.

1) Retrieving Types

- The gettype() function returns the type of the provided variable.
- In total, eight possible return values are available: array, boolean, double, integer, object, resource, string, and unknown type.
- Its prototype follows:

string gettype(mixed var)

```
<html>
<body>
 <?php
        $str="hello";
        $n=10;
        echo "type of str is ",gettype($str);
        echo"</br>";
        echo "type of n is ",gettype($n);
```

```
?>
</body>
</html>
```

Output:

type of str is string
type of n is integer

2) Converting Types

- The settype() function converts a variable to the type specified by type.
- Seven possible type values are available: array, boolean, float, integer, null, object, and string.
- If the conversion is successful, TRUE is returned; otherwise, FALSE is returned.
- Its prototype follows:
  boolean settype(mixed var, string type)

```
<html>
<body>
<?php
        $str="hello";
        $n=10;
        echo "type of str is ",gettype($str);
        echo"</br>";
        echo "type of n is ",gettype($n);
        settype($n,string);
        echo"</br>";
        echo "type of n is ",gettype($n);
?>
</body>
</html>
```

Output:

Type of str is string
Type of n is integer
Type of n is string

3.4.5 Type Identifier Functions

- A number of functions are available for determining a variable's type, including is_array(), is_bool(), is_float(), is_integer(), is_null(), is_numeric(), is_object(), is_resource(), is_scalar(), and is_string().
- Because all of these functions follow the same naming convention, arguments, and return values, their introduction is consolidated into a single example.
- The generalized prototype follows:
  boolean is_name(mixed var)
- All of these functions are grouped ultimately accomplishes the same task.
- Each determines whether a variable, specified by var, satisfies a particular condition specified by the function name.
- If var is indeed of the type tested by the function name, TRUE is returned; otherwise,FALSE is returned.
- An example follows:

```
<? php
$item = 43;
printf("The variable \$item is of type array: %d <br />", is_array($item));
printf("The variable \$item is of type integer: %d <br />", is_integer($item));
```

```
printf("The variable \$item is numeric: %d <br />", is_numeric($item));
?>
```

This code returns the following:

```
The variable $item is of type array: 0
The variable $item is of type integer: 1
The variable $item is numeric: 1
```

3.5 Identifiers

- *Identifier* is a general term applied to variables, functions, and various other user-defined objects.
- There are several properties that PHP identifiers must abide by:
  - An identifier can consist of one or more characters and must begin with a letter or an underscore. Furthermore, identifiers can consist of only letters, numbers, underscore characters, and other ASCII characters from 127 through 255.
  - ***Valid and Invalid Identifiers***

Valid	**Invalid**
my_function	This&that
Size	!counter
_someword	4ward

  - Identifiers are case sensitive. Therefore, a variable named $recipe is different from a variable named $Recipe, $rEciPe, or $recipE.
  - Identifiers can be any length. This is advantageous because it enables a programmer to accurately describe the identifier's purpose via the identifier name.
  - An identifier name can't be identical to any of PHP's predefined keywords.

3.6 Variables

3.6.1 Variable Declaration
3.6.2 Variable scope
3.6.3 Variable variables

3.6.1 Variable Declaration:

- A variable is a named memory location that contains data and may be manipulated throughout the execution of the program.
- A variable always begins with a dollar sign, $, which is then followed by the variable name.
- Variable names follow the same naming rules as identifiers.
- The following are all valid variables:

```
$color
$operating_system
$_some_variable
$model
```

- Note that variables are case senCsitive.
- For instance, the following variables bear no relation to one another:

```
$color
$Color
$COLOR
```

- Interestingly, variables do not have to be explicitly declared in PHP as they do in a language such as C.
- Rather, variables can be declared and assigned values simultaneously.
- Good programming practice dictates that all variables should be declared prior to use.
- After declaring the variables,values can be assigned to them.
- **Two methodologies are available for variable assignment:**
  - **by value and**
  - **by reference.**

Value Assignment

- Assignment by value simply involves copying the value of the assigned expression to the variable assignee.
- This is the most common type of assignment.
- A few examples follow:

```
$color = "red";
$number = 12;
$age = 12;
$sum = 12 + "15"; // $sum = 27
```

- Here, each of these variables possesses a copy of the expression assigned to it
- For example, $number and $age each possesses their own unique copy of the value 12.
- If you prefer that two variables point to the same copy of a value, you need to assign by reference.

Reference Assignment

- PHP 4 introduced the ability to assign variables by reference, which essentially means that you can Create a variable that refers to the same content as another variable does.
- Therefore, a change to any variable referencing a particular item of variable content will be reflected among all other variables referencing that same content.
- You can assign variables by reference by appending an ampersand (&) to the equal sign. Let's consider an example:

```
<? php
        $value1 = "Hello";
        $value2 =& $value1; // $value1 and $value2 both equal "Hello"
```

3.6.2 Variable Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced / used.
- PHP variables can be one of four scope types:
  - **3.6.2.1 Local variables**
  - **3.6.2.2 Function parameters**
  - **3.6.2.3 Global variables**
  - **3.6.2.4 Static variables**

3.6.2.1 Local Variables

A variable declared in a function is considered *local*. That is, it can be referenced only in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function. Note that when you exit the function in which a local variable has been declared, that variable and its corresponding value are destroyed

```
$x = 4;
function assignx ()
{
        $x = 0;
        printf("\$x inside function is %d <br />", $x);
}
assignx();
        printf("\$x outside of function is %d <br />", $x);
```

$x inside function is 0
$x outside of function is 4

3.6.2.2 Function Parameters

In PHP, as in many other programming languages, any function that accepts arguments must declare those arguments in the function header. Although those arguments accept values that come from outside of the function, they are no longer accessible once the function has exited.

```
// multiply a value by 10 and return it to the caller
    function x10 ($value)
    {
    $value = $value * 10;
    return $value;
    }
```

Keep in mind that although you can access and manipulate any function parameter in the function in which it is declared, it is destroyed when the function execution ends.

3.6.2.3 Global Variables

In contrast to local variables, a *global* variable can be accessed in any part of the program. To modify a global variable, however, it must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword global in front of the variable that should be recognized as global.

```
<html>
    <?php
        $x = 5;
        $y = 10;

    function myTest()
    {
        global $x,$y;
         $y = $x + $y;
    }
    myTest();
        echo $y; // outputs 15
    ?>
</html>
```

3.6.2.4 PHP The static Keyword

Normally, when a function is completed / executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.
To do this, use the **static** keyword when you first declare the variable:

Example

```
<? Php
    function myTest()
    {
    static $x = 0;
    echo $x;
     $x++;
    }
    myTest();
    myTest();
    myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

3.6.3 Variable Variables

- On occasion, you may want to use a variable whose content can be treated dynamically as a variable in itself. Consider this typical variable assignment:
  $recipe = "spaghetti";
- Interestingly, you can treat the value spaghetti as a variable by placing a second dollar sign in front of the original variable name and again assigning another value:
  $$recipe = "& meatballs";
- This in effect assigns *& meatballs* to a variable named spaghetti. Therefore, the following two snippets of code produce the same result:
  echo $recipe $spaghetti;
- The result is the string *spaghetti & meatballs*.

3.7 Constants

- A *constant* is a value that cannot be modified throughout the execution of a program.
- Constants are particularly useful when working with values that definitely will not require modification, such as Pi (3.141592).
- Once a constant has been defined, it cannot be changed (or redefined) at any other point of the program.
- Constants are defined using the define() function.

Defining a Constant

- The define() function defines a constant by assigning a value to a name.
- boolean define(string name, mixed value)
  Eg : define("PI", 3.141592);
- The constant is subsequently used in the following listing:

```
printf("The value of Pi is %f", PI);
$pi2 = 2 * PI;
printf("Pi doubled equals %f", $pi2);
```

- This code produces the following results:
  The value of pi is 3.141592.
  Pi doubled equals 6.283184.
- constants are global; they can be referenced anywhere in your script

3.8 Expressions

An *expression* is a phrase representing a particular action in a program. All expressions consist of at least one operand and one or more operators. A few examples follow:

```
$a = 5; // assign integer value 5 to the variable $a
$a = "5"; // assign string value "5" to the variable $a
$sum = 50 + $some_int; // assign sum of 50 + $some_int to $sum
$wine = "Zinfandel"; // assign "Zinfandel" to the variable $wine
$inventory++; // increment the variable $inventory by 1
```

3.8.1 Operands

Operands are the inputs of an expression

```
$a++; // $a is the operand
$sum = $val1 + val2; // $sum, $val1 and $val2 are operands.
```

3.8.2 Operators

An *operator* is a symbol that specifies a particular action in an expression. complete listing of all operators, ordered from highest to lowest precedence.

Operator	Purpose
new	Object instantiation
( )	Expression subgrouping
[ ]	Index enclosure
! ~ ++ --	Boolean NOT, bitwise NOT, increment, decrement

@	Error suppression
/ * %	Division, multiplication, modulus
+ - .	Addition, subtraction, concatenation
<< >>	Shift left, shift right (bitwise)
< <= > >=	Less than, less than or equal to, greater than, greater than or equal to
== != === <>	Is equal to, is not equal to, is identical to, is not equal to
& ^ |	Bitwise AND, bitwise XOR, bitwise OR
&& ||	Boolean AND, Boolean OR
?:	Ternary operator
= += *= /= .= %=&=|= ^= <<= >>=	Assignment operators
AND XOR OR	Boolean AND, Boolean XOR, Boolean OR
,	Expression separation

Operator Precedence

Operator precedence is a characteristic of operators that determines the order in which they evaluate the operands surrounding them.

PHP follows the standard precedence rules used in elementary school math class. Consider a few examples:

```
$total_cost = $cost + $cost * 0.06;
```

This is the same as writing

```
$total_cost = $cost + ($cost * 0.06);
```

because the multiplication operator has higher precedence than the addition operator.

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

3.9 String Interpolation

3.9.1 Double quotes
3.9.2 Escape sequences
3.9.3 Single quotes
3.9.4 Curly braces
3.9.5 Heredoc

To offer developers the maximum flexibility when working with string values, PHP offers a means for both literal and figurative interpretation

3.9.1 Double Quotes

Strings enclosed in double quotes are the most commonly used in PHP scripts because they offer the most flexibility. This is because both variables and escape sequences will be parsed accordingly.

Consider the following example:

```
<?php
$sport = "boxing";
echo "Jason's favorite sport is $sport.";
?>
```

This example returns the following:

Jason's favorite sport is boxing.

3.9.2 Escape Sequences

Escape sequences are also parsed. Consider this example:

```
<?php
$output = "This is one line.\nAnd this is another line.";
echo $output;
?>
```

Output:

This is one line. And this is another line.

In addition to the newline character, PHP recognizes a number of special escape sequences, all of which are

Recognized Escape Sequences

Sequence	Description
\n	Newline character
\r	Carriage return
\t	Horizontal tab
\\	Backslash
\$	Dollar sign
\"	Double quote
\[0-7]{1,3}	Octal notation
\x[0-9A-Fa-f]{1,2}	Hexadecimal notation

3.9.3 Single Quotes

Enclosing a string within single quotes is useful when the string should be interpreted exactly as stated. This means that both variables and escape sequences will not be interpreted when the string is parsed.

For example, consider the following single-quoted string:

print 'This string will $print exactly as it\'s \n declared.';

This produces the following:

This string will $print exactly as it's \n declared.

3.9.4 Curly Braces

While PHP is perfectly capable of interpolating variables representing scalar data types, you'll find that variables representing complex data types such as arrays or objects cannot be so easily parsed when embedded in an echo() or print() string. You can solve this issue by delimiting the variable in curly braces, like this:

echo "The capital of Ohio is {$capitals['ohio']}.";

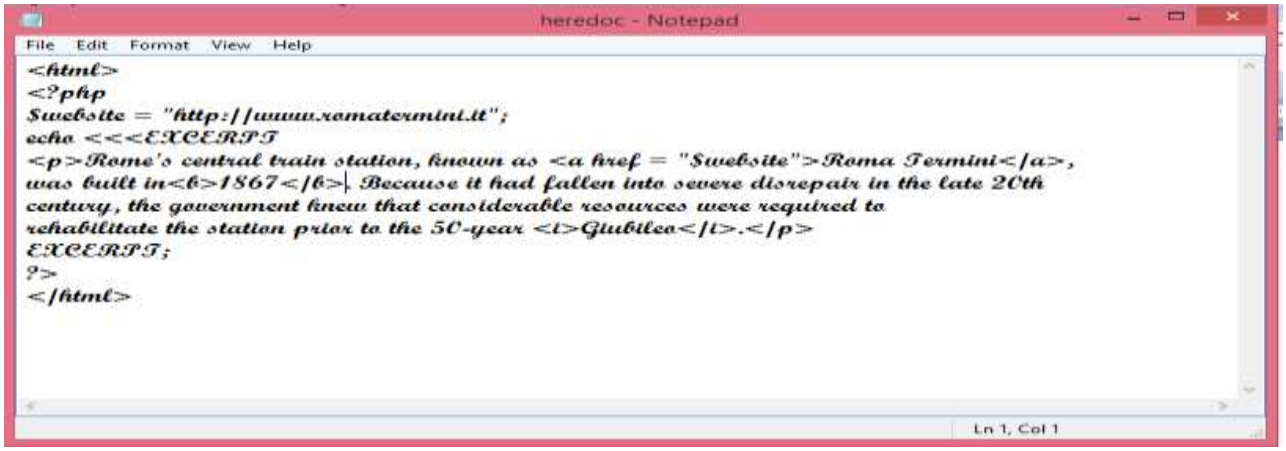
3.9.5 Heredoc

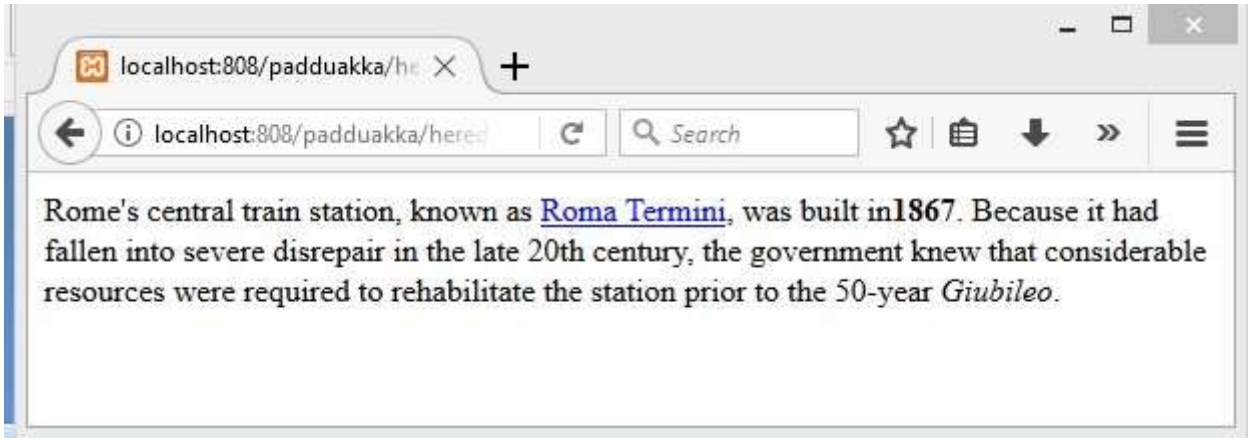
Heredoc syntax offers a convenient means for outputting large amounts of text. Rather than delimiting strings with double or single quotes, two identical identifiers are employed. An example follows:

```
<? php
        $website = "http://www.romatermini.it";
        echo <<<EXCERPT
        <p>Rome's central train station, known as <a href = "$website">Roma Termini</a>,
        was built in 1867. Because it had fallen into severe disrepair in the late 20th
        century, the government knew that considerable resources were required to
        rehabilitate the station prior to the 50-year <i>Giubileo</i>.</p>
        EXCERPT;
?>
```

Several points are worth noting regarding this example:

- The opening and closing identifiers (in the case of this example, EXCERPT) must be Identical. You can choose any identifier you please, but they must exactly match. The only constraint is that the identifier must consist of solely alphanumeric characters and underscores and must not begin with a digit or an underscore.
- The opening identifier must be preceded with three left-angle brackets (<<<).
- Heredoc syntax follows the same parsing rules as strings enclosed in double quotes. That is, both variables and escape sequences are parsed. The only difference is that double quotes do not need to be escaped.
- The closing identifier must begin at the very beginning of a line. It cannot be preceded with spaces or any other extraneous character. Furthermore, the presence of any spaces following the opening or closing identifier will produce a syntax error.





3.10 ARRAYs

3.10.1 What is an Array?
3.10.2 Creating an Array
3.10.3 Outputting an Array
3.10.4 Adding and Removing Array Elements
3.10.5 Locating Array Elements
3.10.6 Traversing Arrays
3.10.7 Determining Array Size and Uniqueness
3.10.8 Sorting Arrays
3.10.9 Merging, Slicing, Splicing, and Dissecting Arrays
3.10.10 Other useful Array Functions

3.10.1 What is an Array?

- Array is a variable that represents collection of homogeneous Data items.

3.10.2 Creating an Array

- Two ways of Creating Arrays are
  1) **To Create an Empty Array**
     $variable = array (); // to create an empty array
     Eg: $a = array ();
  2) **To Create and Initialize an Array**
     $variable = array (item1, item2,…,itemN);
     Eg: $a = array (10,20,30);
- **Associative arrays are arrays that use named keys**

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

```
<?php
$age= array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter   is   " .   $age['Peter']   . "   years   old.";
?>
```

Output:

Peter is 35 years old.

3.10.3 Outputting an Array

3.10.3.1 Printing Arrays for Testing Purposes

- The most common way to output an array's contents is by iterating over each key and echoing the corresponding value.
- foreach statement does the trick nicely:

```
<html>
    <? php
            $a = array ();
            $a[0] = "10";
            $a[1]  = "20";
            $a[2] = "30";
            foreach ($a AS $i)
            {
                    echo "{$i}<br/>";
            }
    ?>
</html>
```

OUTPUT:

10
20
30

3.10.3.1 Printing Arrays for Testing Purposes

- print_r() function is used to easily output their contents to the screen for testing purposes.
- Its prototype follows:

  boolean print_r(mixed *variable* [, boolean *return*])
- The print_r() function accepts a variable and sends its contents to standard output, returning TRUE on success and FALSE otherwise.
- The optional parameter return modifies the function's behavior, causing it to return the output to the caller, rather than send it to standard output.
- Therefore, to return the contents of the preceding $a array, just set return to TRUE:

Program :

```
<html>
        <? php
```

```
                $a = array();
                $a[0] = "10";
                $a[1]  = "20";
                $a[2] = "30";
                print_r($a);
        ?>
</html>
```

Output

Array ([0] => 10 [1] => 20 [2] => 30)

program

```
 <html>
        <?php
                $a = array();
                $a[0] = "10";
                $a[1]  = "20";
                $a[2] = "30";
                $res = print_r($a, TRUE);
                echo $res;
        ?>
</html>
```

Output

Array ([0] => 10 [1] => 20 [2] => 30)

10.3.4 Adding and Removing Array Elements

- **10.3.4.1 Adding a Value to the Front of an Array**
- **10.3.4.2 Adding a Value to the End of an Array**
- **10.3.4.3 Removing a Value from the Front of an Array**
- **10.3.4.4 Removing a Value from the End of an Array**

10.3.4.1 Adding a Value to the Front of an Array

- Adding a Value to the Front of an Array
- The array_unshift() function adds elements to the front of the array.
- All preexisting numerical keys are modified to reflect their new position in the array.
- Its prototype follows:

  int array_unshift(array *array*, mixed *variable* [, mixed *variable*...])

- The following example adds two states to the front of the $states array:

```
$states = array("Ohio", "New York");
array_unshift($states, "California", "Texas");
// $states = array("California", "Texas", "Ohio", "New York");
```

10.3.4.2 Adding a Value to the End of an Array

- The **array_push()** function adds a value to the end of an array, returning the total count of elements in the array after the new value has been added.
- You can push multiple variables onto the array simultaneously by passing these variables into the function as input parameters.
- Its prototype follows:

  int array_push(array *array*, mixed *variable* [, mixed *variable*...])

- The following example adds two more states onto the $states array:

```
$states = array("Ohio", "New York");
array_push($states, "California", "Texas");
// $states = array("Ohio", "New York", "California", "Texas");
```

10.3.4.3 Removing a Value from the Front of an Array

- The **array_shift()** function removes and returns the first item found in an array.
- Its prototype follows:

  mixed array_shift(array *array*)

- The following example removes the first state from the $states array:

```
$states = array("Ohio", "New York", "California", "Texas");
$state = array_shift($states);
// $states = array("New York", "California", "Texas")
// $state = "Ohio“
```

10.3.4.4 Removing a Value from the End of an Array

- The **array_pop()** function removes and returns the last element from an array.
- Its prototype follows:/

mixed array_pop(array *array*)

- The following example removes the last state from the $states array:

```
$states = array("Ohio", "New York", "California", "Texas");
$state = array_pop($states);
// $states = array("Ohio", "New York", "California"
// $state = "Texas".
```

10.3.5 Locating Array Elements

- Several functions are available to search arrays in order to locate items of interest

10.3.5.1 Searching an Array
10.3.5.2 Searching Associative Array Keys
10.3.5.3 Searching Associative Array Values
10.3.5.4 Retrieving Array Keys
10.3.5.5 Retrieving Array values

10.3.5.1 Searching an Array

- The **in_array()** function searches an array for a specific value, returning TRUE if the value is found and FALSE otherwise.
- Its prototype follows:

boolean in_array(mixed *needle*,)

- **I**n the following example, a message is output if a specified state (Ohio) is found in an array

```
$state = "Ohio";
$states = array("California", "Hawaii", "Ohio", "New York");
if(in_array($state, $states))
echo "Not to worry, $state is smoke-free!";
```

10.3.5.2 Searching Associative Array Keys

- The function **array_key_exists()** returns TRUE if a specified key is found in an array and FALSE otherwise.
- Its prototype follows:

boolean array_key_exists(mixed *key*, array *array*)

- The following example will search an array's keys for Ohio, and if found, will output information Ohio joined the Union on March 1, 1803

```
$state["Delaware"] = "December 7, 1787";
$state["Pennsylvania"] = "December 12, 1787";
$state["Ohio"] = "March 1, 1803";
if (array_key_exists("Ohio", $state))
printf("Ohio joined the Union on %s", $state["Ohio"]);
```

- The following is the result:

Ohio joined the Union on March 1, 1803

10.3.5.3 Searching Associative Array Values

- The array_search() function searches an array for a specified value, returning its key if located and FALSE otherwise.
- Its prototype follows:

mixed array_search(value, array)

- The following example searches $state for a particular date (December 7), returning information about the corresponding state if located:

```
$state["Ohio"] = "March 1";
```

```
$state["Delaware"] = "December 7";
$state["Pennsylvania"] = "December 12";
$founded = array_search("December 7", $state);
if ($founded) printf("%s was founded on %s.", $founded,  $state[$founded]);
```

- The output follows:

-Delaware was founded on December 7.

10.3.5.4 Retrieving Array Keys

- The **array_keys()** function returns an array consisting of all keys located in an array. Its prototype follows:

array array_keys(array *array*)

- The following example outputs all of the key values found in the $state array:

```
$state["Delaware"] = "December 7, 1787";
$state["Pennsylvania"] = "December 12, 1787";
$state["New Jersey"] = "December 18, 1787";
$keys = array_keys($state);
print_r($keys);
```

- The output follows:

Array ([0] => Delaware [1] => Pennsylvania [2] => New Jersey)

10.3.5.5 Retrieving Array values

- The **array_values()** function returns all values located in an array, automatically providing numeric indexes for the returned array.
- Its prototype follows:

array array_values(array *array*)

- The following example will retrieve the population numbers for all of the states found in $population:

```
$population = array("Ohio" => "11,421,267", "Iowa" => "2,936,760");
print_r(array_values($population));
```

- This example will output the following:

Array ([0] => 11,421,267 [1] => 2,936,760)

10.3.6 Traversing Arrays

10.3.6.1 Retrieving the Current Array Key
10.3.6.2 Retrieving the Current Array Value
10.3.6.3 Retrieving the Current Array Key and Value
10.3.6.4 Moving the Array Pointer
10.3.6.4.1 Moving the Pointer to the Next Array Position
10.3.6.4.2 Moving the Pointer to the First Array Position
10.3.6.4.3 Moving the Pointer to the Last Array Position

10.3.6.1 Retrieving the Current Array Key

- The **key()** function returns the key located at the current pointer position of the provided array. Its prototype follows:

mixed key(array *array*)

- Example :

```
<html>
    <body>
        <? php
            $num = array (10=>"apple", 20=>"mango",30=>"orange");
            $key = key($num);
            print "Current key is $key";
        ?>
    </body>
```

```
</html>
```

- This returns the following: The Current key is 10

10.3.6.2 Retrieving the Current Array Value

- The **current()** function returns the array value residing at the current pointer position of the array. Its prototype follows: **mixed current(array *array*)**

```
<html>
    <body>
        <?php
            $num = array(10=>"apple",20=>"mango",30=>"orange");
            $val =current($num);
            print "Current key is $val";
        ?>
    </body>
</html>
```

This returns the following: The Current value is apple

10.3.6.3 Retrieving the Current Array Key and Value'

- **each () function returns the** current key/value pair from the array and advances the pointer one position.
- Its prototype follows:

array each(array *array*)

PROGRAM

```
<html>
    <body>
        <?php
            $num = array(10=>"apple", 20=>"mango",30=>"orange");
            $key_val =each($num);
            print_r($key_val);
        ?>
    </body>
</html>
```

OUTPUT

Array ([1] => apple [value] => apple [0] => 10 [key] => 10)

10.3.6.4 Moving the Array Pointer

- Several functions are available for moving the array pointer.

10.3.6.4.1 Moving the Pointer to the Next Array Position

- The next() function returns the array value residing at the position immediately following that of the current array pointer. Its prototype follows:

mixed next(array *array*)

10.3.6.4.2 Moving the Pointer to the First Array Position

- The **reset()** function serves to set an array pointer back to the beginning of the array. Its prototype follows: **mixed reset(array *array*)**

```
<?php
    $num = array("apple","mango","orange");
    $n = next($num);
    echo $n;//displays mango;
    $r = reset($num);//brings cursor to apple
    $c=current($num);//displays current value
    echo $c;
?>
```

10.3.6.4.3 Moving the Pointer to the Last Array Position

- The **end()** function moves the pointer to the last position of an array, returning the last element. Its prototype follows:
  mixed end(array *array*)
- The following example demonstrates retrieving the first and last array values:

```
$fruits = array("apple", "orange", "banana");
$fruit = reset(fruits); // returns "apple"
$fruit = end($fruits); // returns "banana"
```

10..3.7 Determining Array Size and Uniqueness

- A few functions are available for determining the number of total and unique array values.

10.3.7.1 Determining the Size of an Array
10.3.7.2 Counting Array Value Frequency
10.3.7.3 Determining Unique Array Values

10.3.7.1 Determining the Size of an Array

- The count() function returns the total number of values found in an array. Its prototype follows:
  integer count(array *array)*
- number of vegetables found in the $garden array:

```
$garden = array("cabbage", "peppers", "turnips", "carrots");
echo count($garden);
```

- This returns the following: **4**

10.3.7.2 Counting Array Value Frequency

- The array_count_values() function returns an array consisting of associative key/value pairs. Its prototype follows:
  array array_count_values(array *array*)
- Each key represents a value found in the input_array, and its corresponding value denotes the frequency of that key's appearance (as a value) in the input_array. An example follows:

```
$states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
$stateFrequency = array_count_values($states);
print_r($stateFrequency);
```

- This returns the following:
  Array ([Ohio] => 2 [Iowa] => 2 [Arizona] => 1)

10.3.7.3 Determining Unique Array Values

- The **array_unique()** function removes all duplicate values found in an array, returning an array consisting of solely unique values.
- Its prototype follows:
  array array_unique(array *array)*
- An example follows:

```
$states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
$uniqueStates = array_unique($states);
print_r($uniqueStates);
```

- This returns the following:
  Array ([0] => Ohio [1] => Iowa [2] => Arizona)

3.10.8 Sorting Arrays

3.10.8.1 Reversing Array Element Order
3.10.8.2 Flipping Array Keys and Values
3.10.8.3 Sorting an Array
3.10.8.4 Sorting an Array in Reverse Order

3.10.8.1 Reversing Array Element Order

- The array_reverse() function reverses an array's element order. Its prototype follows:
  array array_reverse(array *array)
- An example follows:
  $states = array("Delaware", "Pennsylvania", "New Jersey");
  print_r(array_reverse($states));
- This example returns the following:

Array ([0] => New Jersey [1] => Pennsylvania [2] => Delaware)

3.10.8.2 Flipping Array Keys and Values

- The array_flip() function reverses the roles of the keys and their corresponding values in an array. Its prototype follows:
  array array_flip(array *array*)
- An example follows:
  $state = array("Delaware", "Pennsylvania", "New Jersey");
  $state = array_flip($state);
  print_r($state);
- This example returns the following:
  Array ([Delaware] => 0 [Pennsylvania] => 1 [New Jersey] => 2)

3.10.8.3 Sorting an Array

- The sort() function sorts an array, ordering elements from lowest to highest value. Its prototype follows:
  void sort(array *array)
- Consider an example. Suppose you want to sort exam grades from lowest to highest:
  $grades = array(42, 98, 100, 100, 43, 12);
  sort($grades);
  print_r($grades);
- The outcome looks like this:

Array ([0] => 12 [1] => 42 [2] => 43 [3] => 98 [4] => 100 [5] => 100)

3.10.8.4 Sorting an Array in Reverse Order

- The rsort() function is identical to sort(), except that it sorts array items in reverse (descending) order. Its prototype follows:
  void rsort(array *array* [, int *sort_flags*])
- An example follows:
  $states = array("Ohio", "Florida", "Massachusetts", "Montana");
  rsort($states);
  print_r($states);
- It returns the following:
  Array ([0] => Ohio [1] => Montana [2] => Massachusetts [3] => Florida)

3.10.9 Merging, Slicing, Splicing, and Dissecting Arrays

3.10.9.1 Merging Arrays

- The array_merge() function merges arrays together, returning a single, unified array.
- The resulting array will begin with the first input array parameter, appending each subsequent array parameter in the order of appearance.
- Its prototype follows:

array array_merge(array *array1*, array *array2* [, array *arrayN*])

```
<? php
        $face = array("J", "Q", "K", "A");
        $numbered = array("2", "3", "4", "5", "6", "7", "8", "9");
        $cards = array_merge($face, $numbered);
        foreach($cards AS $i)
                echo "$i</br>";
?>
```

3.10.9.2 Combining Two Arrays

- The array_combine() function produces a new array consisting of a submitted set of keys and corresponding values.
- Its prototype follows:
  array array_combine(array *keys*, array *values*)
- Both input arrays must be of equal size, and neither can be empty. An example follows:

```
$abbreviations = array("AL", "AK", "AZ", "AR");
$states = array("Alabama", "Alaska", "Arizona", "Arkansas");
$stateMap = array_combine($abbreviations,$states);
print_r($stateMap);
```

- This returns the following:
  Array ([AL] => Alabama [AK] => Alaska [AZ] => Arizona [AR] => Arkansas)

3.10.9.3 Slicing an Array

- The array_slice() function returns a section of an array based on a starting and ending offset value. Its prototype follows:
  array array_slice(array *array*, int *offset* [, int *length*)

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut");
$subset = array_slice($states, 4);
print_r($subset);
```

- This returns the following:

Array ([0] => California [1] => Colorado [2] => Connecticut)

3.10.9.4 Splicing an Array

- The array_splice() function removes all elements of an array found within a specified range, returning those removed elements in the form of an array.
- Its prototype follows:

array array_splice(array *array*, int *offset* [, int *length* [, array *replacement*]])

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Connecticut");
$subset = array_splice($states, 4);
print_r($states);
print_r($subset);
```

- This produces the following (formatted for readability):

Array ([0] => Alabama [1] => Alaska [2] => Arizona [3] => Arkansas)
Array ([0] => California [1] => Connecticut)

3.10.9.5 Calculating an Array Intersection

- The **array_intersect()** function returns a key-preserved array consisting only of those values present in the first array that are also present in each of the other input arrays. Its prototype follows:

array array_intersect(array *array1*, array *array2* [, *arrayN*])

- The following example will return all states found in the $array1 that also appear in $array2 and $array

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$intersection = array_intersect($array1, $array2, $array3);
print_r($intersection);
```

- This returns the following:

```
Array ( [0] => OH [3] => H
```

3.10.9.6 Calculating an Array Difference

- Essentially the opposite of array_intersect(), the function array_diff() returns those values located in the first array that are not located in any of the subsequent arrays:

array_diff(array *array1*, array *array2* [, arrayN])

- An example follows:

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$diff = array_diff($array1, $array2, $array3);
print_r($intersection);
```

- This returns the following:

```
Array ( [0] => CT )
```

3.10.10 Other Useful Array Function

3.10.10.1 Returning a Random Set of Keys

3.10.10.2 Shuffling Array Elements

3.10.10.1 Returning a Random Set of Keys

- The array_rand() function will return a random number of keys found in an array. Its prototype follows:

mixed array_rand(array *array* [, int *num_entries*])

- Omitting the optional num_entries parameter, only one random value will be returned.
- tweak the number of returned random values by setting num_entries accordingly. An example follows:

```
$states = array("Ohio" => "Columbus", "Iowa" => "Des Moines",
"Arizona" => "Phoenix");
$randomStates = array_rand($states, 2);
print_r($randomStates);
```

- This returns the following (your output may vary):

```
Array ( [0] => Arizona [1] => Ohio )
```

3.10.10.2 Shuffling Array Elements

- Shuffling Array Elements
- The shuffle() function randomly reorders an array. Its prototype follows:

void shuffle(array *input_array*)

- Consider an array containing values representing playing cards:

```
$cards = array("jh", "js", "jd", "jc", "qh", "qs", "qd", "qc",
"kh", "ks", "kd", "kc", "ah", "as", "ad", "ac");
shuffle($cards);
print_r($positions);
```

- This returns something along the lines of the following (your results will vary because of the shuffle):

```
Array ( [0] => js [1] => ks [2] => kh [3] => jd
[4] => ad [5] => qd [6] => qc [7] => ah
```

[8] => kc [9] => qh [10] => kd [11] => as
[12] => ac [13] => jc [14] => jh [15] => qs)

UNIT -IV

1) Object-Oriented PHP
2) Object Cloning
3) Interfaces
4) Inheritance
5) Namespaces
6) Error and Exception handling
7) Working with Files and Operating System
8) Date and Time in PHP

1) Object-Oriented PHP

1.1) Benefits of OOP

1.1.1) Encapsulation
1.1.2) Inheritance
1.1.3) Polymorphism

1.2.1) Classes

- Our everyday environment consists of countless entities: plants, people, vehicles, food...I could go on for hours just listing them.
- Each entity is defined by a particular set of characteristics and behaviors that ultimately serves to define the entity for what it is.
- For example, a vehicle might be defined as having characteristics such as color, number of tires, make, model, and capacity, and having behaviors such as stop, go, turn, and honk horn.
- In the vocabulary of OOP, such an embodiment of an entity's defining attributes and behaviors is known as a *class*.
- Classes are intended to represent those real-life items that you'd like to manipulate within an application.
- For example, if you want to create an application for managing a public library, you'd probably want to include classes representing books, magazines, employees, special events, patrons, anything else that would require oversight.
- Each of these entities embodies a certain set of characteristics and behaviors, better known in OOP as properties and methods, respectively, that define the entity as what it is.
- ***Class Defines Data and Functions that manipulate the data***
- **PHP's generalized class creation syntax follows:**

```
class ClassName
{
// Property declarations defined here
// Method declarations defined here
}
```

- **Example:**

```
class Employee
    {
        Public $name;
        public $age;
        Public function setDetails($x,$y);
        {
            $this->$name = $x;
            $this->$age = $y;
        }
        Public function getDetails()
        {
            Echo "$this->$name";
            Echo "$this->$age";
        }
    }
```

1.2.2) Objects

- Instances of Class are called Objects. For example, an employee management application may include an Employee class. You can then call upon this class to create and maintain specific instances, Sally and Jim,
- for example: Objects are created using the new keyword, like this:
  $sally = new Employee();
  $Jim = new Employee();
- Once the object is created, all of the characteristics and behaviors defined within the class are made available to the newly instantiated object.

1.2.3)Properties

Properties are attributes that are intended to describe some aspect of a class.

Declaring Properties

The rules regarding property declaration are quite similar to those in place for variable declaration;. Because PHP is a loosely typed language, properties don't even necessarily need to be declared; they can simply be created and assigned simultaneously by a class object Instead, common practice is to declare properties at the beginning of the class. Optionally, you can assign them initial values at this time. An example follows:

```
class Employee
{
public $name = "John";
private $age;
}
```

Invoking Properties

Properties are referred to using the -> operator and, unlike variables, are not prefaced with a dollar sign. Furthermore, because a property's value typically is specific to a given object, it is correlated to that object like this:

```
$object->property
```

For example, the Employee class includes the properties name, title, and wage. If you create an object of type Employee, you would refer to its properties like this:

```
$Jim->name
$Jim->age
```

Property Scopes

PHP supports 4 class property scopes: *public, private, protected* and *static*.

Public

Properties can be declared as public by prefacing the property with the keyword public. An example follows:

```
class Employee
{
public $name;
// Other property and method declarations follow...
}
```

Public properties can then be accessed and manipulated directly via the corresponding object, like so:

```
$employee = new Employee();
$employee->name = "Mary Swanson";
$name = $employee->name;
echo "New employee: $name";
Executing this code produces the following:
New employee: Mary Swanson
```

Private

Private properties are only accessible from within the class in which they are defined. An example follows:

```
class Employee
{
private $name;
private $telephone;
}
```

Properties designated as private are not directly accessible by an instantiated object, nor are they available to child classes. private properties must be accessed via publicly exposed interfaces, which satisfies one of OOP's main concept Encapsulation.

Consider the following example, in which a private property is manipulated by a public method:

```
class  Employee
{
private $name;
public function setName($x)
{
        $this->name = $x;
}
}
$emp = new Employee;
$emp->setName("Mary");//works

$emp->name = "Sita";//fatal error,cant access private property outside the class
$e1 = $emp->name;
Echo $e1;
```

Protected

Protected properties are also made available to inherited classes for access and manipulation.

```
class Employee
{
protected $wage;
}
```

1.2.4) Constants

Constants who value does not intended to change within a class. These values will remain unchanged throughout the lifetime of any object instantiated from that class. Class constants are created like so:

```
const NAME = 'VALUE';
class mathFunctions
{
const PI = '3.14159265';
const E = '2.7182818284';
const EULER = '0.5772156649';
// Define other constants and methods here...
}
Class constants can then be called like this:
echo  mathFunctions::PI;
// Define other constants and methods here...
}
```

1.2.5) Declaring Methods

Methods are created in exactly the same fashion as functions, using identical syntax. The only difference between methods and normal functions is that the method declaration is typically prefaced with a scope descriptor. The generalized syntax follows:

```
scope function functionName()
{
// Function body goes here
}
```

For example, a public method titled calculateSalary() might look like this:

```
public function calculateSalary()
{
return $this->wage * $this->hours;
}
```

Invoking Methods

Methods are invoked in almost exactly the same fashion as functions.

```
$employee = new Employee("Janie");
```

```
$salary = $employee->calculateSalary();
```

Method Scopes

PHP supports six method scopes: *public, private, protected, abstract, final*, and *static*.

Public

Public methods can be accessed from anywhere at any time. You declare a public method by prefacing it with the keyword public or by forgoing any prefacing whatsoever.

```
<?php
class Visitors
{
        public function greetVisitor()
        {
                echo "Hello<br />";
        }
        function sayGoodbye()
        {
                echo "Goodbye<br />";
        }
}
Visitors::greetVisitor();// accessing without object because method is public
$visitor = new Visitors();
$visitor->sayGoodbye();
?>
```

The following is the result:

```
Hello
Goodbye
```

Private

Methods marked as *private* are available for use only within the originating class and cannot be called by the instantiated object, nor by any of the originating class's child classes.

Protected

Class methods marked as *protected* are available only to the originating class and its child classes.

Abstract

Abstract methods are special in that they are declared only within a parent class but are implemented in child classes. Only classes declared as *abstract* can contain abstract methods. You might declare an abstract method if you want to define an application programming interface (API) that can later be used as a model for implementation.Abstract methods are declared like this:

```
abstract function methodName();
```

Suppose that you want to create an abstract Employee class, which would then serve as the base class for a variety of employee types (manager, clerk, cashier, etc.):

```
abstract class Employee
{
abstract function hire();
abstract function fire();
abstract function promote();
abstract demote();
}
```

This class could then be extended by the respective employee classes, such as Manager, Clerk, and Cashier.

Final

Marking a method as *final* prevents it from being overridden by a subclass. A finalized method is declared like this:

```
class Employee
{
final function getName() {
```

```
...
}
}
```

1.3 Constructors and Destructors

1.3.1 Constructors

- Constructors are used to initialize instance Variables.
- Constructors are used to set values to instance variables , either to default or used defined values.
- Quite simply, a constructor is defined as a block of code that automatically executes at the time of object instantiation.
- OOP constructors offer a number of advantages:
  - Constructors can call class methods or other functions.
  - Class constructors can call on other constructors, including those from the class parent.
- PHP recognizes constructors by the name __construct (a double underscore precedes the constructor keyword).
- The general syntax for constructor declaration follows:

```
function __construct([argument1, argument2, ..., argumentN])
{
// Class initialization code
}
```

1.3.1.1 Invoking Parent Constructors

- PHP does not automatically call the parent constructor; it must be explicitly called using the parent keyword.
- An example follows:

```
<?php
class Employee
{
    protected $name;
    protected $title;

    function __construct()
    {
        echo "<p>Employee constructor called!</p>";
    }
}
class Manager extends Employee
{
    function __construct()
    {
        parent::__construct();
        echo "<p>Manager constructor called!</p>";
    }
}
```

This results in the following:

```
Employee constructor called!
Manager constructor called!
```

1.3.1.2 Invoking Unrelated Constructor

- It is possible to invoke class constructors that don't have any relation to the instantiated object simply by prefacing __construct with the class name, like

```
classname::__construct()
```

- As an example, assume that the Manager and Employee classes used in the previous example bear no hierarchical relationship; instead, they are simply two classes located within the same library.
- The Employee constructor could still be invoked within Manager's constructor, like this:

```
Employee::__construct();
```

1.3.2 Destructors

- PHP 5 introduces a destructor concept similar to that of other object-oriented languages, such as C++.
- The destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence.
- Destructors are created like any other method but must be titled __destruct(). An example follows:

```
<?php
    class Book
    {
        private $title;
        private $isbn;
        private $copies;
        function  __construct($isbn)
        {
            echo "<p>Book class instance created.</p>";
        }
        Function  __destructor()
        {
            echo "<p>Book class instance destroyed.</p>";
        }
    }
    $book = new Book("0615303889");
?>
```

- Here's the result:

```
Book class instance created.
Book class instance destroyed.
```

1.4 Static Class Members

- Sometimes it's useful to create properties and methods that are not invoked by any particular object but rather are pertinent to and are shared by all class instances.
- For example, suppose that you are writing a class that tracks the number of web page visitors.
- You wouldn't want the visitor count to reset to zero every time the class is instantiated, so you would set the property to be of the static scope:

```
<?php
    class Visitor
    {
        private static $visitors = 0;
        function __construct()
        {
            self::$visitors++;
        }
        static function getVisitors()
        {
            return self::$visitors;
        }
    }
    // Instantiate the Visitor class.
        $visits = new Visitor();
        echo Visitor::getVisitors()."<br />";
    // Instantiate another Visitor class.
        $visits2 = new Visitor();
        echo Visitor::getVisitors()."<br />";
?>
```

The results are as follows:

```
1
```

2

- Because the $visitors property was declared as static, any changes made to its value (in this case via the class constructor) are reflected across all instantiated objects.
- Also note that static properties and methods are referred to using the self keyword and class name, rather than via $this and arrow operators.

1.5 The instanceof Keyword

- The instanceof keyword was introduced with PHP 5. With it you can determine whether an object is an instance of a class, is a subclass of a class, or implements a particular interface, and do something accordingly.
- For example, suppose you want to learn whether $manager is derived from the class Employee:

```
$manager = new Employee();
...
if ($manager instanceof Employee) echo "Yes";
```

1.6 Helper Functions

- A number of functions are available to help the developer manage and use class libraries.

1) Creating a Class Alias

The **class_alias()** function creates a class alias, allowing the class to be referred to by more than one
name. Its prototype follows:

boolean class_alias(string *originalClassName*, string *aliasName*)

2) Determining Whether a Class Exists

The class_exists() function returns TRUE if the class specified by class_name exists within the currently executing script context and returns FALSE otherwise. Its prototype follows:

boolean class_exists(string *class_name*)

3) Determining Object Context

The get_class() function returns the name of the class to which object belongs and returns FALSE if object is not an object. Its prototype follows:

string get_class(object object)

4) Learning about Class Methods

The get_class_methods() function returns an array containing all method names defined by the class class_name. Its prototype follows:

array get_class_methods(mixed *class_name*)

5) Learning about Class Properties

The get_class_vars() function returns an associative array containing the names of all properties and their corresponding values defined within the class specified by class_name. Its prototype follows:

array get_class_vars(string *class_name*)

6) Learning about Object Properties

The function get_object_vars() returns an associative array containing the defined properties available to object and their corresponding values. Those properties that don't possess a value will be assigned NULL within the associative array. Its prototype follows:

array get_object_vars(object *object*)

7) Determining an Object's Parent Class

The get_parent_class() function returns the name of the parent of the class to which object belongs. If object's class is a base class, that class name will be returned. Its prototype follows:

string get_parent_class(mixed *object*)

8) Determining Interface Existence

The interface_exists() function determines whether an interface exists, returning TRUE if it does and FALSE otherwise. Its prototype follows:

boolean interface_exists(string *interface_name* [, boolean *autoload*])

9) Determining Object Type

The is_a() function returns TRUE if object belongs to a class of type class_name or if it belongs to a class that is a child of class_name. If object bears no relation to the class_name type, FALSE is returned. Its prototype follows:

boolean is_a(object *object*, string *class_name*)

10) Determining Object Subclass Type

The is_subclass_of() function returns TRUE if object (which can be passed in as type string or object) belongs to a class inherited from class_name and returns FALSE otherwise. Its prototype follows:

boolean is_subclass_of(mixed *object*, string *class_name*)

11) Determining Method Existence

The method_exists() function returns TRUE if a method named method_name is available to object and returns FALSE otherwise. Its prototype follows:

boolean method_exists(object *object*, string *method_name*)

```
class Employee
{
        Public $name;
        public $age;
        Public function setDetails($x,$y);
        {
                $this->$name = $x;
                $this->$age = $y;
        }
        Public function getDetails()
        {
                Echo "$this->$name";
                Echo "$this->$age";
        }
}
$emp = new Employee();
$emp->setDetails("smith",30);
$emp->getDetails();

class Employee
{
        Public $name;
        private $age;
}
$emp = new Employee()
$emp->name = "sita";
$stu = $emp->name;
Echo "$stu";//displays sita
$emp->$age = 30;
$num = $emp->age;//error because age is a private property , cant be accessed
Echo "$num";
```

2) Object Cloning

Object cloning is the act of making a copy of an object. In any complex object-oriented PHP program, there are situations that require copies of objects. By assigning an object instance to a new variable,, one creates only a new reference and the object's state information is shared by both reference variables.

```
$object1 = new Account();
$object2 = $object1;
```

Sometimes it is necessary to copy of the object rather than a copy of the reference. So, PHP provides an intrinsic operation using the keyword clone:

```
$account = new Account();
$clonedAccount = clone $account;
```

```
<?php
class Corporate_Drone
{
        private $employeeid
        private $tiecolor;
        function setEmployeeID($employeeid)
         {
                $this->employeeid = $employeeid;
        }
        function getEmployeeID()
        {
                return $this->employeeid;
        }
        function setTieColor($tiecolor)
         {
                $this->tiecolor = $tiecolor;
        }
        function getTieColor()
        {
                return $this->tiecolor;
        }
}
$drone1 = new Corporate_Drone();
$drone1->setEmployeeID("12345");
$drone1->setTieColor("red");
$drone2 = clone $drone1;
$drone2->setEmployeeID("67890");
printf("Drone1 employeeID: %d <br />", $drone1->getEmployeeID());
printf("Drone1 tie color: %s <br />", $drone1->getTieColor());
printf("Drone2 employeeID: %d <br />", $drone2->getEmployeeID());
printf("        Drone2 tie color: %s <br />", $drone2->getTieColor());
```

Executing this code returns the following output:

```
        Drone1 employeeID: 12345
        Drone1 tie color: red
        Drone2 employeeID: 67890
        Drone2 tie color: red
```

The __clone() Method

Any code in this method will execute directly following PHP's native cloning behavior, By adding the following method in Corporate_Drone class

```
        function __clone()
         {
        $this->tiecolor = "blue";
        }
```

let's create a new Corporate_Drone object, add the employeeid property value, clone it, and then output some data to show that the cloned object's tiecolor was indeed set through the __clone() method.

Extending clone's Capabilities with the __clone() Method

```
        // Create new Corporate_Drone object
        $drone1 = new Corporate_Drone();
        // Set the $drone1 employeeid property
        $drone1->setEmployeeID("12345");
        // Clone the $drone1 object
        $drone2 = clone $drone1;
        // Set the $drone2 employeeid property
        $drone2->setEmployeeID("67890");
        // Output the $drone1 and $drone2  employeeid properties
        printf("Drone1 employeeID: %d <br />", $drone1->getEmployeeID());
```

```
printf("Drone2 employeeID: %d <br />", $drone2->getEmployeeID());
printf("Drone2 tie color: %s <br />", $drone2->getTieColor());
```

OUTPUT

Drone1 employeeID: 12345
Drone2 employeeID: 67890
Drone2 tie color: blue

3. Interfaces

3.1 Implementing a Single Interface
3.2 Implementing Multiple Interfaces

- Interface is similar to Class.
- Class defines instance variables and Methods that manipulate the Data whereas Interface defines static and final variables and abstract methods.
- However, an interface is different from a class in several ways, including –
  - You cannot instantiate an interface.
  - An interface does not contain any constructors.
  - All of the methods in an interface are abstract.
  - An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - An interface is not extended by a class; it is implemented by a class.
  - An interface can extend multiple interfaces.
- We declare an interface with the **interface** keyword and, the class that inherits from an interface with the **implements**keyword.
- Let's see the general case:

```
interface interfaceName
{
  // abstract methods
}

class Child implements interfaceName
{
  // defines the interface methods and may have its own code
}
```

3.1 Implementing a Single Interface

In the simple example given below, we will create an interface for the classes that handle cars, which commits all its child classes to setModel() and getModel() methods.

```
interface Car
{
        public function setModel($name);
        public function getModel();
}
```

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
class miniCar implements Car
{
        private $model;
        public function setModel($name)
        {
                $this -> model = $name;
        }
        public function getModel()
        {
```

```
                return $this -> model;
            }
        }
```

3.2 Implementing Multiple Interfaces

- We can implement a number of interfaces in the same class, and so circumvent the law that prohibits the inheritance from more than one parent class.
- In order to demonstrate multiple inheritance from different interfaces, we create another interface, Vehicle

```
interface Vehicle
{
        public function setHasWheels($bool);
        public function getHasWheels();
}
```

- **Now, our child class can implement the two interfaces.**

```
class miniCar implements Car, Vehicle
{
        private $model;
        private $hasWheels;
        public function setModel($name)
        {
                $this -> model = $name;
        }
        public function getModel()
        {
                return $this -> model;
        }
        public function setHasWheels($bool)
        {
                $this -> hasWheels = $bool;
        }
        public function getHasWheels()
        {
                return ($this -> hasWheels)? "has wheels" : "no wheels";
        }
}
```

4. Inheritance

4.1 Class Inheritance
4.2 Inheritance and constructors

4.1 Class Inheritance

- **Inheritance is** a mechanism in which one class acquires all the properties and behaviors of parent class.
- The idea behind inheritance is that
  - New classes can be built on existing classes.
  - Child classes can reuse methods and fields of parent class and can add new methods and fields.
- Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship
- Class inheritance in PHP is accomplished by using the extends keyword

```
<html>
<body>
    <?php
        class Parent1
        {
```

```
                function par_method()
                {
                        echo "Parent Class method invoked</br>";
                }
        }
        class Child extends Parent1
        {
                function child_method()
                {
                        echo "Child Class method invoked</br>";
                }
        }
        class SubChild extends Child
        {
                function subChild_method()
                {
                        echo "SubChild Class method invoked";
                }
        }
        $obj = new SubChild();
        $obj->par_method();
        $obj->child_method();
        $obj->subChild_method();

    ?>
</body>
</html>
```

Output

Parent Class method invoked
Child Class method invoked
SubChild Class method invoked

4.2 Inheritance and Constructors

Inheritance is very powerful and useful OOP concept that gives flexibility and re-usability to the code in an efficient way. A common question pertinent to class inheritance has to do with the use of constructors. Does a parent class constructor execute when a child is instantiated? If so, what happens if the child class also has its own constructor? Does it execute in addition to the parent constructor, or does it override the parent?

If a parent class offers a constructor, it does execute when the child class is instantiated, provided parent constructor must be invoked in the child constructor as

Parentclassname::__construct()

```
<html>
    <body>
        <?php
            class Parent1
            {
                function __construct()
                {
                    echo"parent constructor invoked</br>";
                }
            }
            class Child extends Parent1
            {
                function __construct()
```

```
                {
                        Parent1::_construct();
                        echo"child constructor invoked</br>";
                }
        }

        $obj = new Child();

    ?>
  </body>
</html>
```

output

parent constructor invoked
child constructor invoked

5) Namespaces

- As the size of the PHP code library increases, sometimes it becomes necessary to accidentally reuse a function or class name that has been declared before.
- Name collision problems can be solved with namespaces. PHP constants, classes, and functions can be grouped into namespaced libraries.

How are Namespaces Defined?

- By default, all constant, class and function names are placed in a global space Namespaced code is defined using a single namespace keyword at the top of your PHP file.
- It **must** be the first command (with the exception of declare) and no non-PHP code or white-space can precede the command, e.g.

```
<?php
// define this code in the MyProject namespace
namespace MyProject;
```

- The code following this line will belong to the MyProject namespace.
- Different namespaced code can be defined in the same file

```
<?php
namespace MyProject1;
// PHP code for the MyProject1 namespace
namespace MyProject2;
// PHP code for the MyProject2 namespace
// Alternative syntax
namespace MyProject3 {
  // PHP code for the MyProject3 namespace
}
?>
```

- It is advisable to define a single namespace per file

Calling Namespaced Code

- In a file named lib1.php, we will define a constant, function, and class with the App\Lib1 namespace:

lib1.php

```
<?php
// application library 1
namespace App\Lib1;
```

```
const MYCONST  = 'App\Lib1\MYCONST';
function MyFunction()
{
        return __FUNCTION__;
}
class MyClass
{
        static function WhoAmI()
        {
                return __METHOD__;
        }
}
?>
```

- To call this code, we can use PHP code such as:

myapp.php

```
<?php
header('Content-type: text/plain');
require_once('lib1.php');

echo \App\Lib1\MYCONST . "\n";
echo \App\Lib1\MyFunction() . "\n";
echo \App\Lib1\MyClass::WhoAmI() . "\n";
?>
```

- No namespace is defined for myapp.php so the code exists in the global space.
- Any direct reference to MYCONST, MyFunction() or MyClass will fail because they exist in the App\Lib1 namespace.
- We must therefore add a prefix of \App\Lib1 to create a fully-qualified name.
- The following result is output when we load myapp.php:

```
App\Lib1\MYCONST
App\Lib1\MyFunction
App\Lib1\MyClass::WhoAmI
```

- Fully-qualified names can obviously become quite long .

Namespace Importing

- Namespaces can be imported with the use operator, e.g.

myapp2.php:

```
<?php
use App\Lib1;
require_once('lib1.php');
header('Content-type: text/plain');
echo Lib1\MYCONST  .  "\n";
echo Lib1\MyFunction() . "\n";
echo Lib1\MyClass::WhoAmI() . "\n";
?>
```

Namespace Aliases

- Namespace aliases are perhaps the most useful construct. Aliases allow us to reference long namespaces using a shorter name.
- For the purposes of this example, we will define two almost identical code blocks; the only difference is their namespace:

lib1.php:

```
<?php
// application library 1
namespace App\Lib1;

const MYCONST = 'App\Lib1\MYCONST';
```

```
function  MyFunction() {
        return __FUNCTION_;
}

class  MyClass {
        static function WhoAmI() {
                return __METHOD__;
        }
}
?>
```

lib2.php:

```
<?php
// application library 2
namespace  App\Lib2;

const MYCONST = 'App\Lib2\MYCONST';

function MyFunction() {
        return __FUNCTION__;
}

class  MyClass {
        static function WhoAmI() {
                return __METHOD__;
        }
}
?>
```

myapp3.php:

```
<?php
use App\Lib1 as L;
use App\Lib2\MyClass as Obj;

header('Content-type: text/plain');
require_once('lib1.php');
require_once('lib2.php');

echo L\MYCONST . "\n";
echo L\MyFunction() . "\n";
echo L\MyClass::WhoAmI() . "\n";
echo Obj::WhoAmI() . "\n";
?>
```

The first use statement defines AppLib1 as 'L'. Any qualified names using 'L' will be translated to 'AppLib1' at compile-time. We can therefore refer to LMYCONST and LMyFunction rather than the fully-qualified name.

The second use statement is more interesting. It defines 'Obj' as an alias for the **class** 'MyClass' within the AppLib2 namespace. This is only possible for classes — not constants or functions

7) Working with file and Operating System

- **7.1 Learning About Files and Directories**
  - **7.1.1 Parsing Directory Paths**
  - **7.1.2 Calculating File, Directory, and Disk Sizes**
  - **7.1.3 Determining Access and Modification Times**
- **7.2 Working with Files**
  - **7.2.1 The Concept of a Resource**

7.1 Learning About Files and Directories

- A file is the common storage unit in a computer, and all programs and data are "written" into a file and "read" from a file.
- A Directory holds one or more files, and a Directory can be empty until it is filled. A folder can also contain other folders (subfolders).
- This section introduces many of PHP's built-in functions for obtaining the important details about files and directories such as location, size, last modification time, last access time, and other defining information.

7.1.1 Parsing Directory Paths

- It's often useful to parse directory paths for various attributes such as the tailing extension name, directory component, and base name.
- Several functions are available for performing such tasks.

7.1.1.1 Retrieving a Path's Filename

- The basename() function returns the filename component of a path.

string basename(string *path*)

```
<?php
$path = '/home/www/data/users.txt';
printf("Filename: %s <br />", basename($path));
?>
```

- Executing this example produces the following output:

Filename: users.txt

7.1.1.2 Retrieving a Path's Directory

- The dirname() function is essentially the counterpart to basename(), providing the directory component of a path. Its prototype follows:

string dirname(string *path*)

- The following code will retrieve the path leading up to the file name users.txt:

```
<?php
$path = '/home/www/data/users.txt';
printf("Directory path: %s", dirname($path));
?>
```

- This returns the following:

Directory path: /home/www/data

7.1.1.3 Learning More about a Path

- The pathinfo() function creates an associative array containing three components of a path, namely the directory name, the base name, and the extension. Its prototype follows:

array pathinfo(string *path*)

- Consider the following path:

/home/www/htdocs/book/chapter10/index.html

- The pathinfo() function can be used to parse this path into the following four components:
  - Directory name: /home/www/htdocs/book/chapter10
  - Base name: index.html
  - File extension: html
  - File name: index

- You can use pathinfo() like this to retrieve this information:

```
<?php
$pathinfo = pathinfo('/home/www/htdocs/book/chapter10/index.html');
printf("Dir name: %s <br />", $pathinfo['dirname']);
printf("Base name: %s <br />", $pathinfo['basename']);
printf("Extension: %s <br />", $pathinfo['extension']);
printf("Filename: %s <br />", $pathinfo['filename']);
?>
```

- This produces the following output:

```
Dir name: /home/www/htdocs/book/chapter10
Base name: index.html
Extension: html
Filename: index
```

7.1.1.4 Identifying the Absolute Path

- The realpath() function converts all symbolic links and relative path references located in path to their absolute counterparts. Its prototype follows:

  string realpath(string *path*)

- For example, suppose your directory structure assumes the following path:

  /home/www/htdocs/book/images/

- You can use realpath() to resolve any local path references:

```
<?php
$imgPath = '../../images/cover.gif';
$absolutePath = realpath($imgPath);
        // Returns /www/htdocs/book/images/cover.gif
?>
```

7.1.2 Calculating File, Directory, and Disk Sizes

- Number of Standard PHP Functions to Calculate file, directory, and disk sizes are.

7.1.2.1 Determining a File's Size

- The filesize() function returns the size, in bytes, of a specified file. Its prototype follows:

  int filesize(string *filename*)

- An example follows:

```
<?php
$file = '/www/htdocs/book/chapter1.pdf';
$bytes = filesize($file);
$kilobytes = round($bytes/1024, 2);
printf("File %s is $bytes bytes, or %.2f kilobytes",
    basename($file), $kilobytes);
?>
```

- This returns the following:

  File chapter1.pdf is 91815 bytes, or 89.66 kilobytes

7.1.2.2 Calculating a Disk's Free Space

- The function disk_free_space() returns the available space, in bytes, allocated to the disk partition housing a specified directory. Its prototype follows:

  float disk_free_space(string *directory*)

- An example follows:

```
<?php
$drive = '/usr';
printf("Remaining MB on %s: %.2f", $drive,
round((disk_free_space($drive) / 1048576), 2));
?>
```

- This returns the following:

  Remaining MB on /usr: 2141.29

7.1.2.2 Calculating Total Disk Size

- The disk_total_space() function returns the total size, in bytes, consumed by the disk partition housing a specified directory. Its prototype follows:
  float disk_total_space(string *directory*)
- An example follows:

```
<?php
$partition = '/usr';
// Determine total partition space
$totalSpace = disk_total_space($partition) / 1048576;
?>
```

7.1.3 Determining Access and Modification Times

- PHP offers three functions for determining a file's access, creation, and last modificationtime.

7.1.3.1 Determining a File's Last Access Time

- The fileatime() function returns a file's last access time in Unix timestamp format or FALSE on error.Its prototype follows:
  int fileatime(string *filename*)
- An example follows:

```
<?php
$file = '/var/www/htdocs/book/chapter10/stat.php';
printf("File last accessed: %s", date("m-d-y g:i:sa",
fileatime($file)));
?>
```

- This returns the following:
  File last accessed: 06-09-10 1:26:14pm

7.1.3.2 Determining a File's Last Changed Time

- The filectime() function returns a file's last changed time in Unix timestamp format or FALSE on error.Its prototype follows:
  int filectime(string *filename*)
- An example follows:

```
<?php
$file = '/var/www/htdocs/book/chapter10/stat.php';
printf("File inode last changed: %s", date("m-d-y g:i:sa",
filectime($file)));
?>
```

- This returns the following:
  File inode last changed: 06-09-10 1:26:14pm

7.1.3.3 Determining a File's Last Modified Time

- The filemtime() function returns a file's last modification time in Unix timestamp format or FALSE otherwise. Its prototype follows:
  int filemtime(string *filename*)
- The following code demonstrates how to place a "last modified" timestamp on a web page:

```
<?php
$file = '/var/www/htdocs/book/chapter10/stat.php';
echo "File last updated: ".date("m-d-y g:i:sa",
filemtime($file));
?>
```

- This returns the following:
  File last updated: 06-09-10 1:26:14pm

7.2 Working with Files

7.2.1 The Concept of a Resource

- The term *resource* is commonly used to refer to any entity from which an input or output stream can be initiated.
- Standard input or output, files, and network sockets are all examples of resources.
- Resource handling Functions are

7.2.2 Recognizing Newline Characters

- The newline character, represented by the \n character sequence (\r\n on Windows), denotes the end of a line within a file.
- Several functionsare there working with the newline character. Some of these functions include file(), fgetcsv(), and fgets().

7.2.3 Recognizing the End-of-File Character

- EOF, represents End of file Character.
- In the case of PHP, this function is feof(). The feof() function determines
- whether a resource's EOF has been reached. It is used quite commonly in file I/O operations. Its prototype follows:

int feof(string *resource*)

- An example follows:

```
<?php
// Open a text file for reading purposes
$fh = fopen('/home/www/data/users.txt', 'r');
// While the end-of-file hasn't been reached, retrieve the next line
while (!feof($fh)) echo fgets($fh);
// Close the file
fclose($fh);
?>
```

7.2.4 Opening a File & Closing File

- The fopen() function binds a file to a handle.Its prototype follows:

resource fopen(string *resource*, string *mode)*

```
$fh = fopen('/var/www/users.txt', 'r');
$fh = fopen('/var/www/docs/summary.html', 'w');
```

- The fclose() function handles this for you, closing the previously opened file pointer specified by a file handle, returning TRUE on success and FALSE otherwise. Its prototype follows:

boolean fclose(resource *filehandle*)

7.2.5 Reading from a file

Reading a File into an Array

- The file() function is capable of reading a file into an array, separating each element by the newline character, with the newline still attached to the end of each element. Its prototype follows:

array file(string *filename)*

Reading File Contents into a String Variable

- The file_get_contents() function reads the contents of a file into a string. Its prototype follows:

string file_get_contents(string *filename)*

Reading a Specific Number of Characters

- The fgets() function returns a certain number of characters read in through the opened resource handle, or everything it has read up to the point when a newline or an EOF character is encountered. Itsprototype follows:

string fgets(resource *handle* [, int *length*])

Reading a File One Character at a Time

- The fgetc() function reads a single character from the open resource stream specified by handle. If theEOF is encountered, a value of FALSE is returned. Its prototype follows:

string fgetc(resource *handle*)

Reading in an Entire File

- The readfile() function reads an entire file specified by filename and immediately outputs it to the output buffer, returning the number of bytes read. Its prototype follows:

int readfile(string *filename)*

7.2.6 Writing a String to a File

- The fwrite() function outputs the contents of a string variable to the specified resource. Its prototype follows:

int fwrite(resource *handle*, string *string* [, int *length*])

7.2.7 Moving the File Pointer

- Several PHP functions are available to jump around within a file, reading from and writing to various locations.

Moving the File Pointer to a Specific Offset

- The fseek() function moves the pointer to the location specified by a provided offset value. Its prototype follows:

int fseek(resource *handle*, int *offset* [, int *whence*])

If the optional parameter whence is omitted, the position is set offset bytes from the beginning of the file. Otherwise, *whence* can be set to one of three possible values, which affect the pointer's position:

SEEK_CUR: Sets the pointer position to the current position plus offset bytes.
SEEK_END: Sets the pointer position to the EOF plus offset bytes. In this case, offset must be set to a negative value.
SEEK_SET: Sets the pointer position to offset bytes. This has the same effect as omitting whence.

Moving the File Pointer Back to the Beginning of the File

- The rewind() function moves the file pointer back to the beginning of the resource. Its prototype follows:

int rewind(resource *handle*)

7.2.8 Reading Directory Contents

Opening a Directory Handle

- Just as fopen() opens a file pointer to a given file, opendir() opens a directory stream specified by a path. Its prototype follows:

resource opendir(string *path* [, resource *context*])

Closing a Directory Handle

- The closedir() function closes the directory stream. Its prototype follows:

void closedir(resource *directory_handle*)

Parsing Directory Contents

- The readdir() function returns each element in the directory. Its prototype follows:

string readdir([resource *directory_handle*])

```
<?php
$dh = opendir('/usr/local/apache2/htdocs/');
while ($file = readdir($dh))
echo "$file <br />";
closedir($dh);
?>
```

Sample output follows:

```
.
..
articles
images
news
```

Reading a Directory into an Array

- The scandir() function, introduced in PHP 5, returns an array consisting of files and directories found in directory or returns FALSE on error. Its prototype follows

array scandir(string *directory* [,int *sorting_order*])

- Setting the optional sorting_order parameter to 1 sorts the contents in descending order, overriding the default of ascending order. Executing this example (from the previous section)

```
<?php
print_r(scandir('/usr/local/apache2/htdocs'));
?>
```

returns the following

```
Array ( [0] => . [1] => .. [2] => articles [3] => images
[4] => news [5] => test.php )
```

7.3 Executing Shell Commands

PHP has Several functions to interact with the underlying operating systems

Removing a Directory

- The rmdir() function attempts to remove the specified directory, returning TRUE on success and FALSE
- otherwise. Its prototype follows:

  int rmdir(string *dirname*)

Renaming a File

- The rename() function renames a file, returning TRUE on success and FALSE otherwise. Its prototype follows:

  boolean rename(string *oldname*, string *newname*)

Touching a File

- The touch() function sets the file filename's last-modified and last-accessed times, returning TRUE on

success or FALSE on error. Its prototype follows:

int touch(string *filename* [, int *time* [, int *atime*]])

8. Date and Time

8.1 The Unix Timestamp

- Simply put, the Unix timestamp is a way to track time as a running total of seconds.
- This count starts at the Unix Epoch on January 1st, 1970 at UTC.
- Therefore, the **Unix timestamp is merely the number of seconds between a particular date and the Unix**
- **Epoch**.
- It should also be pointed out that this point in time technically does not change no matter where you are located on the globe.
- This is very useful to computer systems for tracking and sorting dated information in dynamic and distributed applications both online and client side.
- The reason why Unix timestamps are used by many webmasters is because they can represent all time zones at once.
- strtotime() is just so you can convert a normalized date into an actual unix timestamp for cross language/platform compatibility.
- For Example : $date = "2012-12-29";

  $date=strtotime($date);

  Echo $date//displays 1356739200
- You mean to say from January 1st, 1970 to 2012-12-29 number of second is 1356739200

8.2 PHP's Date and Time Library

- Php offers many Date and Time Related functions for Validating Dates,Formatting Dates and Times, Converting a Timestamp to User-Friendly Values and working with timestamps.

8.2.1 Validating Dates

- checkdate() function accomplishes the task of validating dates, returns TRUE if the supplied date is valid and FALSE otherwise. Its prototype follows:

  Boolean checkdate(int *month*, int *day*, int *year*)

- For Example

```
echo "April 31, 2010: ".(checkdate(4, 31, 2010) ? 'Valid' : 'Invalid');
// Returns false, because April only has 30 days
```

8.2.2 Formatting Dates and Times

- The date() function returns a string representation of the current date and/or time formatted according to the instructions specified by a predefined format. Its prototype follows:

Parameter	Description	Example
A	Lowercase ante meridiem and post meridiem	am or pm
A	Uppercase ante meridiem and post meridiem	AM or PM

string date(string *format* [, int *timestamp*])

The date() Function's Format Parameters

D	Day of month, with leading zero	01 to 31
D	Three-letter text representation of day	Mon through Sun
E	Timezone identifier	America/New_York
F	Complete text representation of month	January through December
G	12-hour format, without zeros	1 through 12
G	24-hour format, without zeros	0 through 23
H	12-hour format, with zeros	01 through 12
H	24-hour format, with zeros	00 through 23
i	Minutes, with zeros	01 through 60
I	Daylight saving time	0 if no, 1 if yes
j	Day of month, without zeros	1 through 31
L	Text representation of day	Monday through Sunday
L	Leap year	0 if no, 1 if yes
m	Numeric representation of month, with zeros	01 through 12
M	Three-letter text representation of month	Jan through Dec
n	Numeric representation of month, without	zeros 1 through 12
O	Difference to Greenwich Mean Time (GMT)	–0500
r	Date formatted according to RFC 2822	Tue, 19 Apr 2010 22:37:00 –0500
S	Seconds, with zeros	00 through 59
S	Ordinal suffix of day	st, nd, rd, th
t	Total number of days in month	28 through 31
T	Time zone	PST, MST, CST, EST, etc.
U	Seconds since Unix epoch (timestamp)	1172347916
w	representation of weekday	0 for Sunday through 6 for Saturday
W	ISO 8601 week number of year	1 through 52 or 1 through 53, depending on the day in which the week ends. See ISO 8601 standard for more information.
Y	Four-digit representation of year	1901 through 2038
z	Day of year	0 through 364
Z	Time zone offset in seconds	–43200 through 50400

Examples

```
echo "Today is ".date("F d, Y");
        // Today is August 22, 2010

echo "Today is ".date("l");
                // Today is Wednesday
```

Working with Time

- The date() function can also produce time-related values.

```
echo "The time is ".date("h:i:s");
// The time is 07:44:53
```

8.2.3 Converting a Timestamp to User-Friendly Values

- The getdate() function accepts a timestamp and returns an associative array consisting of its components. Its prototype follows:

array getdate([int *timestamp*])

In total, 11 array elements are returned, including the following:

hours: Numeric representation of the hours. The range is 0 through 23.
mday: Numeric representation of the day of the month. The range is 1 through 31.
minutes: Numeric representation of the minutes. The range is 0 through 59.
mon: Numeric representation of the month. The range is 1 through 12.
month: Complete text representation of the month, e.g., July.
seconds: Numeric representation of the seconds. The range is 0 through 59.
wday: Numeric representation of the day of the week, e.g., 0 for Sunday.

weekday: Complete text representation of the day of the week, e.g., Friday.
yday: Numeric offset of the day of the year. The range is 0 through 364.
year: Four-digit numeric representation of the year, e.g., 2010.
0: Number of seconds since the Unix epoch (timestamp).

Consider the timestamp 1274729324 (May 24, 2010 15:28:44 EDT). Let's pass it to getdate() and review the array elements:

```
Array (
[seconds] => 44
[minutes] => 28
[hours] => 15
[mday] => 24
[wday] => 1
[mon] => 5
[year] => 2010
[yday] => 143
[weekday] => Monday
[month] => May
[0] => 1274729324
)
```

8.2.4 Working with Timestamps

- PHP offers two functions for working with timestamps: time() and mktime().

Determining the Current Timestamp

- The time() function is useful for retrieving the present Unix timestamp. Its prototype follows:

  int time()
- The following example was executed at 15:31:22 EDT on May 24, 2010:

  echo time();
- This produces a corresponding timestamp:

  1274729482

Creating a Timestamp Based on a Specific Date and Time

- The mktime() function is useful for producing a timestamp based on a given date and time. If no date

  and time is provided, the timestamp for the current date and time is returned. Its prototype follows:

  int mktime([int *hour* [, int *minute* [, int *second* [, int *month* [, int *day* [, int *year*]]]]])
- As an example,

  if you want to know the timestamp for May 24, 2010 3:35 p.m., all you have to do is plug in the

  appropriate values:

  echo mktime(15,35,00,5,24,2010);

  This returns the following:

  1274729700

8.3 Date Fu

8.3.1 Displaying the Web Page's Most Recent Modification Date

The getlastmod() function returns the value of the page's Last Modified header or FALSE

in the case of an error. Its prototype follows:

int getlastmod()

Example

```
$lastmod = date("F d, Y h:i:sa", getlastmod());
echo "Page last modified on $lastmod";
```

This returns output similar to the following:

Page last modified on February 26, 2010 07:59:34pm

8.3.2 Determining the Number of Days in the Current Month

To determine the number of days in the current month, use the date() function's t parameter. Consider the following code:

```
printf("There are %d days in %s.", date("t"), date("F"));
```

If this is executed in April, the following result will be output:

There are 30 days in April.

8.3.3 Calculating the Date X Days from the Present Date

It's often useful to determine the precise date of some specific number of days into the future or past. Suppose you want to know what the date will be 45 days into the future, based on today's date of May 24, 2010:

```
$futuredate = strtotime("+45 days");
echo date("F d, Y", $futuredate);
```

This returns the following:

July 08, 2010

UNIT - V

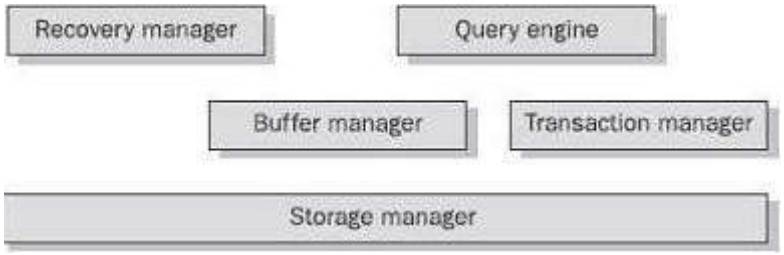
1. **An Overview of MySQL Architecture**
   - 1.1 Primary Subsystems
   - 1.2 Support Components
   - 1.3 Subsystem/Component Interaction and Control Flow
2. **The MySQL Engine**
   - 2.1 Connectivity
   - 2.2 SQL
   - 2.3 Data Integrity
   - 2.4 Transactions

1. An Overview Of MySQL Architecture

- MySQL consists of both ***primary Subsystems*** and ***support components*** that interact with each other to read, parse, execute and return query results.

1.1 Primary SubSystems

- The MySQL Architecture consists of five primary subsystems, which work together to respond to the request made to the MySQL database server.
  - 1.1.1) The Query Engine
  - 1.1.2) The Storage Manager
  - 1.1.3) The Buffer Manager
  - 1.1.4) The Transaction Manager
  - 1.1.5) The Recovery Manager



MySQL subsystems

1.1.1) The Query Engine

This subsystem contains 3 interrelated components

The Syntax Parser
The Query Optimizer
The Execution Component

The ***Syntax Parser*** decomposes the SQL commands it receives from calling programs into a form that can be understood by the MYSQL engine.

The ***Query Optimizer*** , streamlines the syntax and then prepares the most efficient plan of query execution.

The ***Execution Component*** then interprets the execution plan , and based on the information it has received, makes request of the other components to retrieve the record.

1.1.2) The Storage Manager

The Storage Manager interfaces with the operating System(OS) to write data to the disk efficiently.

The Query Cache

If a query returns a given set of records, repeating the same query should return the same set of records unless the underlying data has somehow changed.

The MySQL engine uses an extremely efficient result set caching mechanism,known as the query cache, that dramatically enhances response times for queries that are called upon to retrieve the exact same data as a previous query.

Because of this mechanism, MySQL queries are faster than Oracle and SQL Server.

1.1.3) The Buffer Manager

This Subsystem handles all memory management issues between requests for data by the Query Engine and the Storage Manager.

MySQL makes aggressive use of memory to cache result sets that can be returned as rather than making duplicate requests to the storage Manager. This cache is maintained in the Buffer Manager.

1.1.4) The Transaction Manager

The function of the Transaction Manager is to facilitate concurrency in Data access.This subsystem provides a locking facility to ensure that multiple simultaneous users access the data in a consistent way, without corrupting or damaging the data in any way.

1.1.5) The Recovery Manager

The Recovery Manager's job is to keep copies of data for retrieval later, in case of a loss of data. It also logs commands that modify the data and other significant events inside the Database.

1.2) Support Components

In Addition to the five primary subsystems, the MySQL Architecture contains the following two support components.

- The Process Manager
- Function Libraries

The Process Manager

This component performs two functions in the system. First, it manages user connections via network connection management. Second, it it synchronizes competing tasks and processes via multithreading modules.

Function Libraries

This component contains general-purpose routines that are used by all the other subsystems. It includes String manipulation, sorting etc.. functions

1.3) Subsystem/Component Interaction and Control Flow

- The Query engine requests the data to be read or written to buffer manager to satisfy a users query.
- It depend on the transaction manager for locking of the data so that the concurrency is ensured.
- To perform table creation and drop operations, the query engine accesses the storage manager directly, bypassing the buffer manager ,to create or delete the files in the file system.
- The buffer manager caches data from the storage manager for efficient retrieval by the query engine. It depends on the transaction manager to check the locking status of data before it performs any modification action.
- The Transaction manager depends on the Query cache and storage manager to place locks on data in memory and in the file system.
- The recovery manager uses the Storage manager to store command/event logs and backups of the data in the file system. It depends on the transaction manager to obtain locks on the log files being written. The recovery manager also needs to use the buffer information.manager during recovery from crashes.
- The storage manager depends on the operating system file system for persistent storage and retrieval of data. It depends on the transaction manager to obtain locking status

2) The MySQL Engine

MySQL engine has been designed for Maximum Scalability, Maximum resource efficiency and easy portability to various platforms and Architectures.The Important Characteristics of the SQL Engine are

2.1) Connectivity

MySQL is designed on the assumption that the vast majority of its applications will be running on a TCP/IP network. This is a fairly good assumption , given that TCP/IP is not only highlg robust and secure but is also common to unix , windows and almost any other seriours operating .

2,2) SQL

Its fair to say that SQL is today one of the most widely used cross-vendor language.
As with other Implementation , such as SQL Server's T-SQL and Oracle's SQL, MySQL has its own variations of the SQL standard that add power beyond what is available within the standard.

2.4) Data Integrity

MySQL supports engine-level data integrity through the use of Primary key and foreign key constraints. Columns can be defined so that explicit NULL values cannot be entered into them.

2.5) Transactions

A Transaction-safe database system must pass what is known as the ACID test to qualify for compliance. An ACID –compliant database must support the following characteristics

- Atomicity
- Consistency
- Isolation
- Durability

2.6) Extensibility

In most RDBMS Products, we can extend the capabilities of the database by using stored procedures. The programmability is usually further extended by enhancements to SQL that contains control-of-flow statements and conditional logic, as SQL Server does with T-SQL and oracle with PL/SQL.

As of yet, MySQL includes no support for stored procedures, but one of the great benefits of this RDBMS is its extensibility . Since it is an open source code, it permits the developers to add new functions and features that are compiled into the ngine as part of the core product.

2.7) Symmetric Multiprocessing with MySQL

To Take advantage of multiprocessor architecture, MySQL is built using a multi-threaded design, which allows threads to be allocated between processors to achieve a higher degree of parallelism.

2.8) Security

The process of accessing a MySQL database can be broken down into two tasks

1) Connecting to the MySQL server itself
2) And Accessing individual objects, such as tables or columns in a database.

2.9) Replication

Replication is a Data Distribution mechanism that allows to place copies of tables and databases in remote locations so that users can more easily access them.
For Example, a national company might have a common product database that is updated centrally but that is used locally by each office. Rather than forcing applications to query this table remotely every time it's needed, it is more cost effective to distribute a copy to everyone, thus incurring the transmission overhead only once for each office.

2.10) Application Programming Interfaces

For application developers, MySQL provides set of APIs that provide an understandable set of rules by which host languages can connect to MySQL and send commands. Currently, MySQL APIs are available for Perl, PHP, C ,C++, Java, Visual Basic,Python, Ruby and .NET

2.11) Add-On Tools

List of Software Programs that work with MySQL are

- MySQL CC, is an excellent front-end query and Database Management tool for MySQL
- DBTools Manager Professional, is a graphcal client used to manage MySQL Databases, tables etc. This Tool also helps to import data from other RDBMS and provide a point-and-click interface to query and report design.

3) MySQL Data Types

Properly defining the fields in a table is important to the overall optimization of the database. use only the type and size of field which really needs. For example, do not define a field 10 characters wide, if it requires only 2 characters. These type of fields (or columns) are also referred to as **data types**

MySQL uses many different data types broken into three categories –

- Numeric
- Date and Time

- String Types.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions −

- **INT** − A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** − A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** − A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** − A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** − A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** − A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE(M,D)** − A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** − An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Types

The MySQL date and time datatypes are as follows −

- **DATE** − A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** − A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** − A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** − Stores the time in a HH:MM:SS format.

- **YEAR(M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

String Types

Although the numeric and date types are fun, most data you'll store will be in a string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT** – A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBs and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.
- **TINYBLOB or TINYTEXT** – A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** – A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** – A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
- **ENUM** – An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

4. MySQL Operators

MySQL comes with more than 25 operators , each designed for a specific task.These operators can be classified, according to their function, into the following four categories

4.1) Arithmetic Operators
4.2) Comparision Operators
4.3) Logical Operators
4.4) Bit Operators

4.1) Arithmetic Operators

MySQL supports the following Arithmetic Operators

Operator	What it Does
+	Addition
_	Subtraction
*	Multiplication
/	Division : returns Quotient
%	Division : returns Remainder

Examples :

Select 10+5, 1000000+64763 // ***returns 15 and 1064763***
Select 5-1 // ***returns 4***
Select 7 *10 , 10/2, 10%2 //***returns 70 and 5 and 0***

4.2) Comparision Operators

MySQL provides Numerous Comparisions operators, which allows to compare the left side of an expression with its right side. The result of such a comparision operation is always 1(true), 0(false), or NULL(Could not be determined)

Operator	What it Does
=	Equal To
<> or !=	Not Equal TO
< = >	NULL-safe equal to
<	Less Than
<=	Less Than or Equal To
>	Greater Than
>=	Greater Than or Equal TO
BETWEEN	Exists in Specified Range
IN	Exists in Specified set
IS NULL	Is NULL
IS NOT NULL	Is not NULL
LIKE	Wildcard match
REGEXP or RLIKE	Regular Expression match

These comparision operators can be used to compare both numbers and strings. Numbers are compared as floating-point values, while strings are compared in a case insensitive manner

Examples

2) SELECT 100 > 100, 100 >10, 3.14 > 3.144;

100 > 100	100 > 10	3.14 > 3.144
0	1	0

3) SELECT 'a' > 'a', 'x' <= 'x', 'e' > 'f';

'a' > 'a'	'x' <= 'x'	'e' > 'f'
0	1	0

4) SELECT 99 BETWEEN 10 AND 100, 'f' NOT BETWEEN 'a' AND 'z'

99 BETWEEN 10 AND 100	'f' NOT BETWEEN 'a' AND 'Z'
1	0

5) SELECT 'red' IN ('red','green','blue')

99 BETWEEN 10 AND 100	'f' NOT BETWEEN 'a' AND 'Z'
1	0

6) SELECT 3 NOT IN (1,2)

3 NOT IN(1,2)
0

7) SELECT 'red' REGEXP 'red|blue|green';

'red' REGEXP 'red|blue|green'
1

4.3) Logical Operators

MySQL also comes with 4 Logical operators

MySQL also comes with four logical operators, which make it possible to test the logical validity of one or more expressions.The result of an operation involving these operators is always 1(true), 0(false), or NULL.Logical operators supported by MySQL are

Operator	What it Does
NOT	Logical NOT
AND or &&	Logical ADD
OR & ||	Logical OR
XOR	Logical XOR

Examples:

1) SELECT fname, lname, age from employee where gender = "F" and age >30;
2) SELECT name,age from employee where desg ="manager" or desg = "analyst";
3) SELECT fname,lname from employee where age >25 AND NOT (country = 'us');

4.4) Bit Operators

MySQL also includes 6 operators designed specifically for bit manipulation. They are

Operator	What it Does
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bit inversion
>>	Bitwise right shift
<<	Bitwise left shift

5. MySQL Functions

MySQL Built-in functions are broadly classified into the following groups

5.1) Math Functions
5.2) Aggregate Functions
5.3) String Manipulation Functions
5.4) Date and Time Manipulation Functions
5.5) Data Encryption Functions
5.6) Control Flow Functions
5.7) Formatting Functions
5.8) Type Conversion Functions
5.9) System Information Functions

5.1) Math Functions

MySQL supports the following Math Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is greater than or equal to a number
COUNT	Returns the number of records in a select query

DEGREES	Converts a radian value into degrees
DIV	Used for integer division
EXP	Returns e raised to the power of number
FLOOR	Returns the largest integer value that is less than or equal to a number
GREATEST	Returns the greatest value in a list of expressions
LEAST	Returns the smallest value in a list of expressions
LN	Returns the natural logarithm of a number
MAX	Returns the maximum value of an expression
MIN	Returns the minimum value of an expression
MOD	Returns the remainder of n divided by m
PI	Returns the value of PI displayed with 6 decimal places
POW	Returns m raised to the nth power
POWER	Returns m raised to the nth power
ROUND	Returns a number rounded to a certain number of decimal places
SQRT	Returns the square root of a number
SUM	Returns the summed value of an expression
TRUNCATE	Returns a number truncated to a certain number of decimal places

Examples

1) SELECT GREATEST(100,99,158,63); // Returns 158
2) SELECT LEAST(-2,5,99); //Returns -2
3) SELECT FLOOR(75.1),CEILING(75.1); //Returns 75,76
4) SELECT ROUND(10.33),TRUNCATE(1.78509765,3) //Returns 10,1.785
5) SELECT POW(4,4); //Returns 256.000
6) SELECT SQRT(49), //Returns 7

5.2) Aggregate Functions

Aggregate **functions are most commonly used with SELECT queries containing GROUP BY** clauses. MySQL supports the following Aggregate Functions

Function	What it Does
AVG(col)	Returns the average of the values in the named columns
COUNT(col)	Returns a count of the number of non-NULL records in the named columns
MIN(col)	Returns the minimum value in the named column
MAX(col)	Returns the maximum value in the named column
SUM(col)	Returns the total of the values in the named column
STDDEV(col)	Returns the standard deviation of the values in the named column
VARIANCE(col)	Returns the statistical variance of the values in the named column
GROUP_CONCAT(col)	Returns a concatenated set of column values belonging to a group

Examples

1) SELECT COUNT(*) FROM employee; // Returns 30
2) SELECT MIN(age),MAX(sal) FROM employee; // Returns 25, 50000
3) SELECT SUM(sal) FROM employee; //Returns 1,50,000

5.3) String Manipulation Functions

Since MySQL Database usually contain string values as well as numeric data, MySQL comes with a variety of functions designed to aid in String Manipulation. The important functions are

Function	Description
ASCII	Returns the number code that represents the specific character
CHAR_LENGTH	Returns the length of the specified string (in characters)
CONCAT	Concatenates two or more expressions together
FIELD	Returns the position of a value in a list of values
FORMAT	Formats a number as a format of "#,###.##", rounding it to a certain number of decimal places
INSERT	Inserts a substring into a string at a specified position for a certain number of characters
INSTR	Returns the position of the first occurrence of a string in another string
LCASE	Converts a string to lower-case
LEFT	Extracts a substring from a string (starting from left)
LENGTH	Returns the length of the specified string (in bytes)
LOCATE	Returns the position of the first occurrence of a substring in a string
LOWER	Converts a string to lower-case
LPAD	Returns a string that is left-padded with a specified string to a certain length
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string a specified number of times
REPLACE	Replaces all occurrences of a specified string
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a substring from a string (starting from right)
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string with a specified number of spaces
STRCMP	Tests whether two strings are the same
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

Examples

1) SELECT LENGTH('Rama'); // Returns 4
2) SELECT SUBSTRING('market',2,3) //Returns ark
3) SELECT CONCAT('sita','rama'); //Returns sitarama
4) SELECT REPLACE('I am Joe Cool','Cool','Camel'); //Returns I am Joe Camel
5) SELECT UCASE('rama'),LCASE('SITA'); //Returns RAMA, sita

5.4) Data and Time Functions

Given the large number of Date and Time Data types in MySQL, its no surprise that the RDBMS comes with an equally large number of functions to manipulate data and time values.

Function	Description
ADDDATE	Returns a date after a certain time/date interval has been added
ADDTIME	Returns a time/datetime after a certain time interval has been added
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURTIME	Returns the current time
DATE	Extracts the date value from a date or datetime expression
DATEDIFF	Returns the difference in days between two date values
DATE_ADD	Returns a date after a certain time/date interval has been added
DATE_FORMAT	Formats a date as specified by a format mask
DATE_SUB	Returns a date after a certain time/date interval has been subtracted
DAY	Returns the day portion of a date value
DAYNAME	Returns the weekday name for a date
DAYOFMONTH	Returns the day portion of a date value
DAYOFWEEK	Returns the weekday index for a date value
DAYOFYEAR	Returns the day of the year for a date value
EXTRACT	Extracts parts from a date
FROM_DAYS	Returns a date value from a numeric representation of the day
HOUR	Returns the hour portion of a date value
LAST_DAY	Returns the last day of the month for a given date
MAKEDATE	Returns the date for a certain year and day-of-year value
MAKETIME	Returns the time for a certain hour, minute, second combination
MINUTE	Returns the minute portion of a date value
MONTH	Returns the month portion of a date value
MONTHNAME	Returns the full month name for a date

NOW	Returns the current date and time
PERIOD_ADD	Takes a period and adds a specified number of months to it
PERIOD_DIFF	Returns the difference in months between two periods
SYSDATE	Returns the current date and time
TIME	Extracts the time value from a time/datetime expression
TIME_FORMAT	Formats a time as specified by a format mask
TIME_TO_SEC	Converts a time value into numeric seconds
TIMEDIFF	Returns the difference between two time/datetime values
TIMESTAMP	Converts an expression to a datetime value and if specified adds an optional time interval to the value
TO_DAYS	Converts a date into numeric days
WEEK	Returns the week portion of a date value
WEEKDAY	Returns the weekday index for a date value
WEEKOFYEAR	Returns the week of the year for a date value
YEAR	Returns the year portion of a date value
YEARWEEK	Returns the year and week for a date value

Examples

1) SELECT NOW(); //Returns the current date
2) SELECT CURTIME(),CURDATE(); //Returns Current Time &Date
3) SELECT MONTHNAME('1967-09-21'); //Returns September
4) SELECT DAYNAME('2003-02-14'); //Returns Friday

5.5) Encryption Functions

MySQL comes with a few specialized functions designed specially to perform data encryption. They are

Function	What it Does
AES_ENCRYPT(str,key)	Returns an Advanced Encryption standard(AES)-encrypted version of *str* using secret key *key*
AES_DECRYPT(str,key)	Decrypts an AES_encrypted string *str* using secret key *key*
DECODE(str,key)	Decodes encrypted string *str* with *key*
ENCRYPT(str,key)	Returns an encrypt version of *str* with *salt* using *the UNIX crypt*() function
ENCODE(str,key)	Encodes *str* with *key*
MD5()	Returns MD5 checksum for str
PASSWORD(str)	Returns an encrypted version of str
SHA()	Retuens a secure Hash Algorithm checksum for str

Examples

1) SELECT PASSWORD("secret"); // Returns 428567f408994404
2) SELECT ENCRYPT('open sesame','abc'); // Returns ab/G8gtZdMwak

5.6) Control Flow Functions

MySQL also comes with four functions that can be used to perform conditional operations. These functions make it possible to implement conditional logic within SQL itself, allowing developers to shift some of the applications business logic to the database back end.

Function	What it Does
CASE WHEN [test1] THEN [result1]… ELSE [default] END	Returns resultN if testN evaluates as true, else returns default
CASE[test] WHEN [vall] THEN [result]… ELSE [default] END	Returns resultN if test evaluates to valN,else returns default
IF(test,t,f)	Returns t if test evaluates as true;else returns f
IFNULL(arg1,arg2)	Returns arg1 if arg1 is not NULL;else returns arg2
NULLIF(arg1,arg2)	Returns NULL if arg1 equals arg2;else returns arg2.

Examples

1) SELECT fname,lname,(math + sci + lit) as total,
   CASE WHEN (math + sci + lit) < 50 THEN 'D'
   WHEN (math + sci + lit) BETWEEN 50 AND 150 THEN 'C'
   WHEN (math + sci + lit) BETWEEN 151 AND 250 THEN 'B'
   ELSE 'A' END AS grade FROM marks;

Fname	Lname	Total	Grade
John	Doe	143	C
Sarah	Short	145	C
Joe	Cool	246	B
Mark	Wumba	145	C
Tom	Thumb	12	D

5.7) Formatting Functions

MySQL also comes with a few functions specially designed to assist in formatting data. The simplest of these is the FORMAT() function, intended to format large numeric values into more readable comma-separated sequences.

The First argument to FORMAT() is always the number to be formatted; the second is the number of decimal places in the end value. Formatting functions are

Function	**What it Does**
DATE_FORMAT(date,fmt)	Formats date as per format specified in fmt
FORMAT(x,y)	Formats x as a comma-separated number sequence, rounded off to y decimal places
INET_ATON(ip)	Returns a numeric representation of IP address ip
INET_NTOA(num)	Returns the IP address representation of num
TIME_FORMAT(time,fmt)	Formats time as per format specified in fmt

Example

1) SELECT FORMAT(999999.546789,2); //Returns 999999.54

Function	**What it Does**
%a	Short weekday name(sun,mon…)
%b	Short month name(Jan,Feb…)
%d	Day of the month
%H	Hour(01,02,..)
%I	Minute(00,01,…)
%j	Day of the year(001,002…)
%m	Two-digit month(00,01…)
%M	Long month name(January,February…)
%p	AM/PM
%r	Time in 12-hour format
%s	Second(00,01…)

%T	Time in 24-hour format
%w	Day of the week(0,1…)
%W	Long weekday name(Sunday,Monday…)
%y	Four-digit year

Examples

1) SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');// Returns Thursday , 15th May 2003 01:38:44 PM
2) SELECT DATA_FORMAT(19980317,'%d%m%y');//Returns 17/03/1998

5.8) Type Conversion Functions

To assist in Data type conversion, MySQL offers the CAST() function, which can be used to cast a value to a specific data type. MySQL supports the following types.

- BINARY
- CHAR
- DATE
- TIME
- DATETIME
- SIGNED
- UNSIGNED

Examples

1) SELECT 1 + '99'; //Returns 100
2) SELECT 1 + CAST('99' AS SIGNED); // Returns 100

5.9) System Information Functions

MySQL also comes with specialized functions that can be used to obtain information about the system itself.

The DATABASE(),USER() AND VERSION() functions return information about the currently selected database, current user, and MySQL version, respectively

Function	What it Does
DATABASE()	Returns name of currently selected database
BENCHMARK(count,expr)	Evaluates expr count times
CONNECTION_ID()	Returns client connection ID
FOUND_ROWS()	Returns the number of rows returned by the last SELECT query
GET_LOCK(str,dur)	Obtains a lock named str for dur seconds
IS_FREE_LOCK(str)	Checks to see whether lock named str is free
LAST_INSERT_ID()	Returned the last AUTOINCREMENT ID automatically generated by the system
RELEASE_LOCK(str)	Releases lock named str
USER()	Returns name os currently logged-in user
VERSION()	Returns MySQL server version

6. Accessing a Database in PHP

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

Example

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

```
$dbname = "myDB";

// Create connection
$conn = mysql_connect($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
   // output data of each row
   while($row = $result->fetch_assoc()) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
$conn->close();
?>
```

Run example »

```
id: 1 - Name: John Doe
id: 2 - Name: Mary Moe
id: 3 - Name: Julie Dooley
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The net line of code runs the query and puts the resulting data into a variable called $result.

Then, the function num_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the function fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

You can also put the result in an HTML table:

Example

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

// Create connection
$conn = mysql_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysql_query($conn, $sql);

if (mysql_num_rows($result) > 0) {
   // output data of each row
   while($row = mysql_fetch_assoc($result)) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
```

```
mysqli_close($conn);
?>
```

Run example »

ID	Name
1	John Doe
2	Mary Moe
3	Julie Dooley

7. Updating a Database

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Update Data In a MySQL Table

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Let's look at the "MyGuests" table:

id	Firstname	Lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

Example

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

// Create connection
$conn = mysql_connect($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE  id=2";

if ($conn->query($sql) === TRUE) {
   echo "Record updated successfully";
} else {
```

```
    echo "Error updating record: " . $conn->error;
}

$conn->close();
?>
```

8. Creating a Database

A database consists of one or more tables.The CREATE DATABASE statement is used to create a database in MySQL.

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

// Create connection
$conn = mysql_connect($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

Creating a Table in the Database

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

After the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
    - AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Object-oriented)

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

// Create connection
$conn = new mysql_connect($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
   echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

9. Working with Data

9.1) Inserting, Updating and Deleting Records

9.1.1) Inserting Records

9.1.2) Updating Records

9.1.3) Deleting Records

9.2) Retrieving Records

9.2.1) Retrieving Specific Rows and Columns

9.2.2) Using Built-In Functions

9.2.3) Aliasing Table and column Names

9.2.4) Limiting Query Results

9.2.5) Sorting Query Results

9.2.6) Grouping Query Results

9.2.7) Using Variables

9.2.8) Using Sub queries

9.2.9) Controlling SELECT behavior

9.3) Copying, Importing, and Exporting Records

- **9.3.1) Copying Records**
- **9.3.2) Importing Records**
- **9.3.3) Exporting Records**

10. Joins

- **10.1 What is a join**
- **10.2 Types of joins**
  - **10.2.1.1 Cross Join**
  - **10.2.1.2 Inner Join**
  - **10.2.1.3 Outer Join**
  - **10.2.1.4 Self Join**
  - **10.2.1.5 Unions**