PROGRAM NO: 1

OBJECTIVE: Create web pages in HTML using the following-

a) Basic HTML tags
b) HTML List
d) HTML Table

THEORY:

HTML: HTML stands for Hyper Text Markup Language. HTML is the computer language that is used to create documents for display on the Web. The HTML language consists of a series of HTML tags. Learning HTML involves finding out what tags are used to mark the parts of a document and how these tags are used in creating an HTML document.

A) BASIC HTML TAGS:

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document.
- The <html> element is the root element of an HTML page.
- The <head> element contains meta information about the HTML page.
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading.
- The <p> element defines a paragraph.

SAMPLE CODE:

```
<html>
  <head>
    <title> My first web page </title>
  <head>
  <body>
    <b> <i> WELCOME TO WEB TECHNOLOGY LAB </i></b>
    <h1> web page creation </h1>
  </body>
</html>
```

OUTPUT:

B) HTML List

HTML lists allow web developers to group a set of related items in lists. Types:

- **Unordered HTML List**: An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag.
- **Unordered HTML List**: An ordered list starts with the **<ol>** tag. Each list item starts with the **<li>** tag.
- **Definition List:** To display content as key term and its definition with <dl> tag.

SAMPLE CODE:

```
ORDERED LIST
<ol type="1" start="4">
  <li>ABC</li>
  <li>PQR</li>
  <li>XYZ</li>
</ol>

UNORDERED LIST
<ul type="disc">
  <li>ABC</li>
  <li>PQR</li>
  <li>XYZ</li>
</ul>
```

OUTPUT:

C) HTML Table

- The **<table>** tag defines an HTML table.
- An HTML table consists of one **<table>** element and one or more <tr>, <th>, and <td> elements.
- An HTML table may also include <caption>, <thead>, <tfoot>, and <tbody> elements.

SAMPLE CODE:

```
<table border="5" cellspacing ="50" cellpadding ="20" >
  <tr>
    <td> A </td>
    <td> B </td>
    <td> C </td>
  </tr>
  <tr>
    <td> 1 </td>
    <td> 2 </td>
    <td> 3 </td>
  </tr>
</table>
```

OUTPUT:

PROGRAM NO: 2

OBJECTIVE: Write HTML code to display your CV in navigator, your Institute website, Department Website and Tutorial website for specific subject and use CSS (Inline, Internal, External) at appropriate places.

THEORY:

HTML: HTML stands for Hyper Text Markup Language. HTML is the computer language that is used to create documents for display on the Web. The HTML language consists of a series of HTML tags. Learning HTML involves finding out what tags are used to mark the parts of a document and how these tags are used in creating an HTML document.

Tags are instructions that tell our browser what to show on a Web page. They break up our document into basic sections. All tags start with a < (left bracket) and end with a > (right bracket).

BASIC HTML TAGS:

1. <HTML> </HTML>

This tag tells your browser that the file contains HTML-coded information. All html tags must be placed between the open <HTML> tag and the closed tag </HTML>. The file extension .html also indicates the document is an HTML document. All html documents MUST be saved with the .html file extension.

2. <HEAD> </HEAD>

The head tag identifies the first part of your HTML-coded document. The title tag (explained below) must be places between the open <HEAD> tag and the closed </HEAD> tag.

3. <BODY> </BODY>

The largest part of your HTML document is the body, which contains the content of your document (displayed within the text area of your browser window). All HTML tags that pertain to the body of your HTML document must be places between the open <BODY> tag and the closed </BODY> tag. The tag has attributes which you can use to set the colors_of your background, text, links, and also to include your own background image. They are as follows:

- BGCOLOR="white" Sets the background color (other color names: red, black, blue etc)
- TEXT="black" Sets the body text color
- LINK="blue" Sets the unvisited hypertext links
- VLINK ="purple" Sets the visited hypertext links
- ALINK="red" Sets the active hypertext links (the color of the hypertext link when you have your mouse button depressed)

- BACKGROUND Let you use an image as the background <background= Body attributes are used as part of the open <body> tag. For example:
  <BODY BGCOLOR = "white" TEXT = "black" LINK = "blue" VLINK = "purple" ALINK = "red">

SOME OTHER TAGS USEFUL IN WEBPAGE DESIGNING:

A) HTML LISTS:

1. ORDERED HTML LIST:

- An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.
- The type attribute of the <ol> tag, defines the type of the list item marker:

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

SAMPLE CODE:

```
<ol start ="1" type="1">
 <li> ABC
   <ol start="1" type="a">
    <li>AB
      <ol  start ="5" type= "a">
        <li>A</li>
        <li>B</li>
      </ol>
    </li>
    <li>CD
      <ol  start ="1" type= "a">
        <li>C</li>
        <li>D</li>
      </ol>
    </li>
   </ol>
</ol>
```

OUTPUT:

2. UNORDERED HTML LIST:

- An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag.
- The list items will be marked with bullets (small black circles) by default.

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

SAMPLE CODE:

```
<ul type="square">
  <li> ABC
    <ul type="circle">
      <li>AB</li>
    </ul>
  </li>
  <li>BCD
    <ul type= "disc">
       <li>D</li>
    </ul>
    <ul type="square">
  </li>
</ul>
```

OUTPUT:

3. DEFINITION LIST:

- To display content as key term and its definition.

Tag	Description
dl	description list
dt	definition term
dd	definition description

SAMPLE CODE:

```
<dl>
  <dt> Computer : </dt>
    <dd>  It is an electronic device </dd>
  <dt> Java :  </dt>
    <dd>  It is a programming language </dd>
</dl>
```

OUTPUT:

B) HTML TABLES:

- An HTML table is defined with the <table> tag.
- Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered.
- A table data/cell is defined with the <td> tag.

ATTRIBUTES:

1. Cell padding specifies the space between the cell content and its borders.
2. To left-align the table headings, use the CSS **text-align** property
3. Border spacing specifies the space between the cells. **border-spacing**
4. To make a cell span more than one column, use the **colspan** attribute:
5. To make a cell span more than one row, use the **rowspan** attribute
6. The border is set using the CSS **border** property
7. Width, height, Background , bgcolor
8. Cell spacing – space between 2 cells.

SAMPLE CODE:

```
<table border="3">
        <tr>
          <th> A </th>
          <th> B </th>
          <th> C </th>
        </tr>
        <tr>
          <td> 1 </td>
          <td> 2 </td>
          <td> 3 </td>
        </tr>
 </table>
```

OUTPUT:

C) FRAMES SET:

- HTML frames allows us to display no. of web pages.
- The <frameset> element holds one or more <frame> elements.

SAMPLE CODE:

```
<Frameset cols = "30 %, 40%, 30 %">
  <Frameset rows = "50%, 50 %">
    <Frame>
    <Frame>
  </Frameset>
  <Frameset rows="50 %,*">
    <Frame>
    <Frame>
  </Frameset>
  <Frameset rows = "50 %,*">
    <Frame>
    <Frame>
  </Frameset>
</Frameset>
```

D) CSS (CASCADING STYLE SHEET)

- It is used to enhance the functionality of in built tags.
- Describe how elements are to be displayed on screen.
- CSS is of 3 types –
    1. Inline
    2. Internal
    3. External

1. INLINE CSS:

- Uses style attribute
- Used to apply unique style in a single HTML element i.e. implemented on a particular tag.

SYNTAX: Hello

2. INTERNAL CSS:

- An internal CSS is used to define a style for a single HTML page i.e tag specific.
- Used in <style> in <head>

SYNTAX :

```
<head>
        <style>
            { …….}
        </style>
</head>
```

3. EXTERNAL CSS:

- An external style sheet is used to define the style for many HTML pages.
- With an external style sheet, we can change the look of an entire web site, by changing one file (.css).

SYNTAX:

```
<Head>
        <link href= “a.css” rel =Stylesheet”>  </link>
</Head>
```

SYNTAX OF SELECTORS:

```
Selector
{
    Property: value;
}
```

TYPES OF SELECTORS:

1. Inbuilt selectors: All inbuilt tags eg. <b> , <u>, < ul>, <i> etc.

2. Class selectors (.): User defined. It is a universal selector. i.e. accepted over any Web browser.

3. Id selectors (#): It is only implemented over the inbuilt Web browser.

SOME PROPERTIES OF CSS:

1. Change color
Color: green;
Background-color: red;

2. Change font type

Font-family: times new roman;

3. Change size of text
Font-size: 12px;

4. Type scope
Text-weight: bold;
Font-style: bold

E) CSS NAVIGATION BAR:

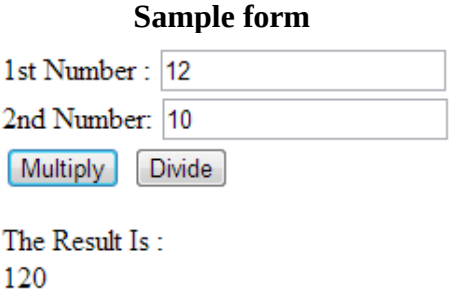
- Having easy-to-use navigation is important for any web site.
- With CSS you can transform boring HTML menus into good-looking navigation bars.

SAMPLE CODE:

```
<div class="topnav">
  <a class="active" href="#home">Home</a>
  <a href="#news">News</a>
  <a href="#contact">Contact</a>
  <a href="#about">About</a>
</div>
```

PROGRAM NO: 3

OBJECTIVE: Use JavaScript to develop the following program:
a) to calculate multiplication and division of two numbers (input from user).



b. to accept a string as a parameter and counts the number of vowels within the string.
c. using function to validate whether a given value in the text box is number or not.

THEORY:

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

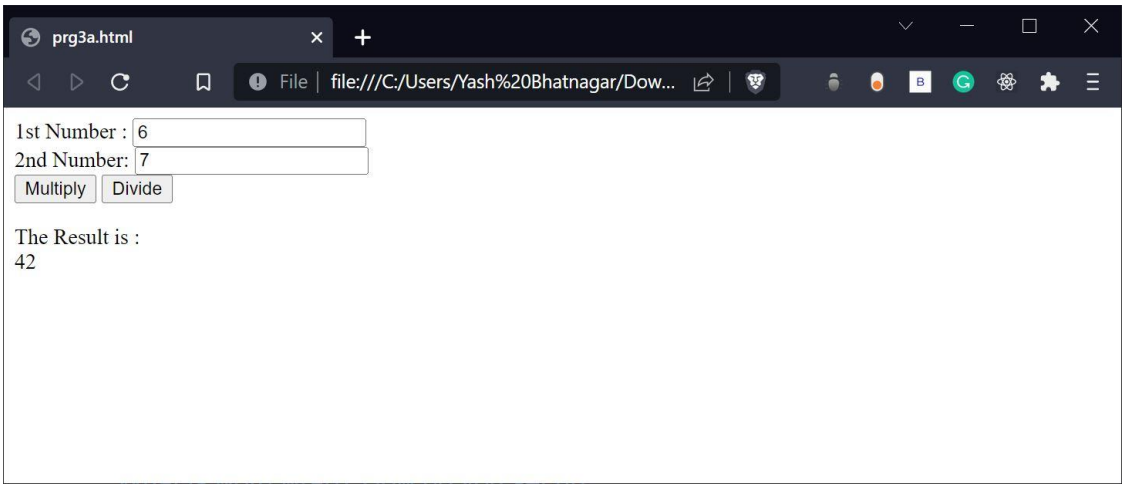
A) to calculate multiplication and division of two numbers

SAMPLE CODE:

```
<body>
  <form>
    1st Number : <input type="text" id="first" /><br>
    2nd Number: <input type="text" id="second" /><br>
    <input type="button" onClick="multiplyBy()" Value="Multiply" />
    <input type="button" onClick="divideBy()" Value="Divide" />
  </form>
  <p>The Result is : <br>
    <span id = "result"></span>
  </p>
  <script>
    function multiplyBy()
      {
        num1 = document.getElementById("first").value;
        num2 = document.getElementById("second").value;
        document.getElementById("result").innerHTML = num1 * num2;
      }
    function divideBy()
      {
        num1 = document.getElementById("first").value;
```

```
      num2 = document.getElementById("second").value;
      document.getElementById("result").innerHTML = num1 / num2;
    }
  </script>
</body>
```

OUTPUT:

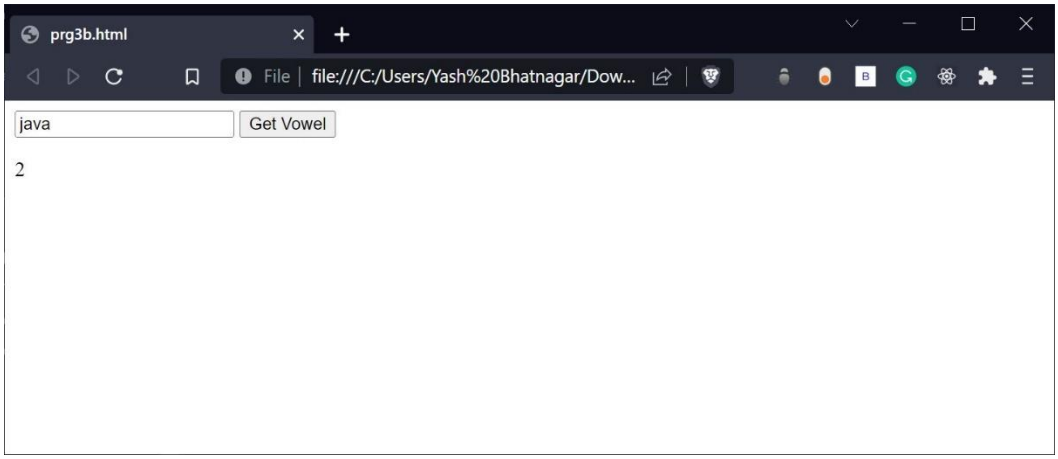


B) to accept a string as a parameter and counts the number of vowels within the string.

SAMPLE CODE:

```
<body>
 <script> function a()
  {
    var v1=document.getElementById ("txt").value;
    var c=0;
    for(var i=0;i<v1.length;i++)
      {
                if(v1[i]==='a'||v1[i]==='e'||v1[i]==='i'||v1[i]==='o'||v1[i] ==='u'
                  ||v1[i]==='A'||v1[i]==='E'||v1[i]==='I'||v1[i]==='O'||v1[i]==='U')
                 {
                   c=c+1;
                }
      }
    document.getElementById('Result').innerHTML = c;
   }
 </script>
 <input type="text" placeholder="Enter the text" id="txt">
 <button type="button" onclick=" a();">Get Vowel </button>
 <p id="Result"></p>
</body>
```

OUTPUT:



C) using function to validate whether a given value in the text box is number or not.

SAMPLE CODE:

```
<body>
  <input type="text" placeholder="Enter the Text" id="Txt">
  <button type="button" onclick="check();">validate</button>
  <p id="result"></p>
  <script>
    function check()
      {
        var v1=document.getElementById("Txt").value;
        if(isNaN(v1)) // Check not a number
          {
            document.getElementById('result').innerHTML = "False";
          }
        else
          {
            document.getElementById('result').innerHTML = "True";
          }
      }
  </script>
</body>
```

OUTPUT:

PROGRAM NO: 4

OBJECTIVE: Write an HTML program to design an entry form of student details and validate it using javascript.

THEORY:

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

HTML FORMS: HTML Forms are required to collect different kinds of user inputs, such as contact details like name, email address, phone numbers, or details like credit card information, etc. Users generally complete a form by modifying its controls e.g. entering text, selecting items, etc. and submitting this form to a web server for processing.

SYNTAX:

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

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, etc.

INPUT ELEMENT: An input element can be of type text field, checkbox, password field, radio button, submit button, reset button, etc. and several new input types. The <input> element can be displayed in several ways, depending on the **type** attribute. Eg.

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)
<input type="password">	Defines a password field
<input type="reset">	Defines a reset button that reset all form values to default values
<input type="checkbox">	Defines checkbox

<input type="button">	Defines a button

THE ACTION ATTRIBUTE: The action attribute defines the action to be performed when the form is submitted.

<form action="path">

THE METHOD ATTRIBUTE: The method attribute specifies the HTTP method (GET or POST) to be used when submitting the form data:

<form action="path" method="get/post">

THE <SELECT> ELEMENT:

- The <select> element defines a drop-down list**.**
- The <option> elements defines an option that can be selected.
- By default, first item in drop-down list is selected. To define a pre-selected option, use selected attribute.

THE <TEXTAREA> ELEMENT: The <textarea> element defines a multi-line input field (a text area**).** It has three main attributes name, rows and cols.

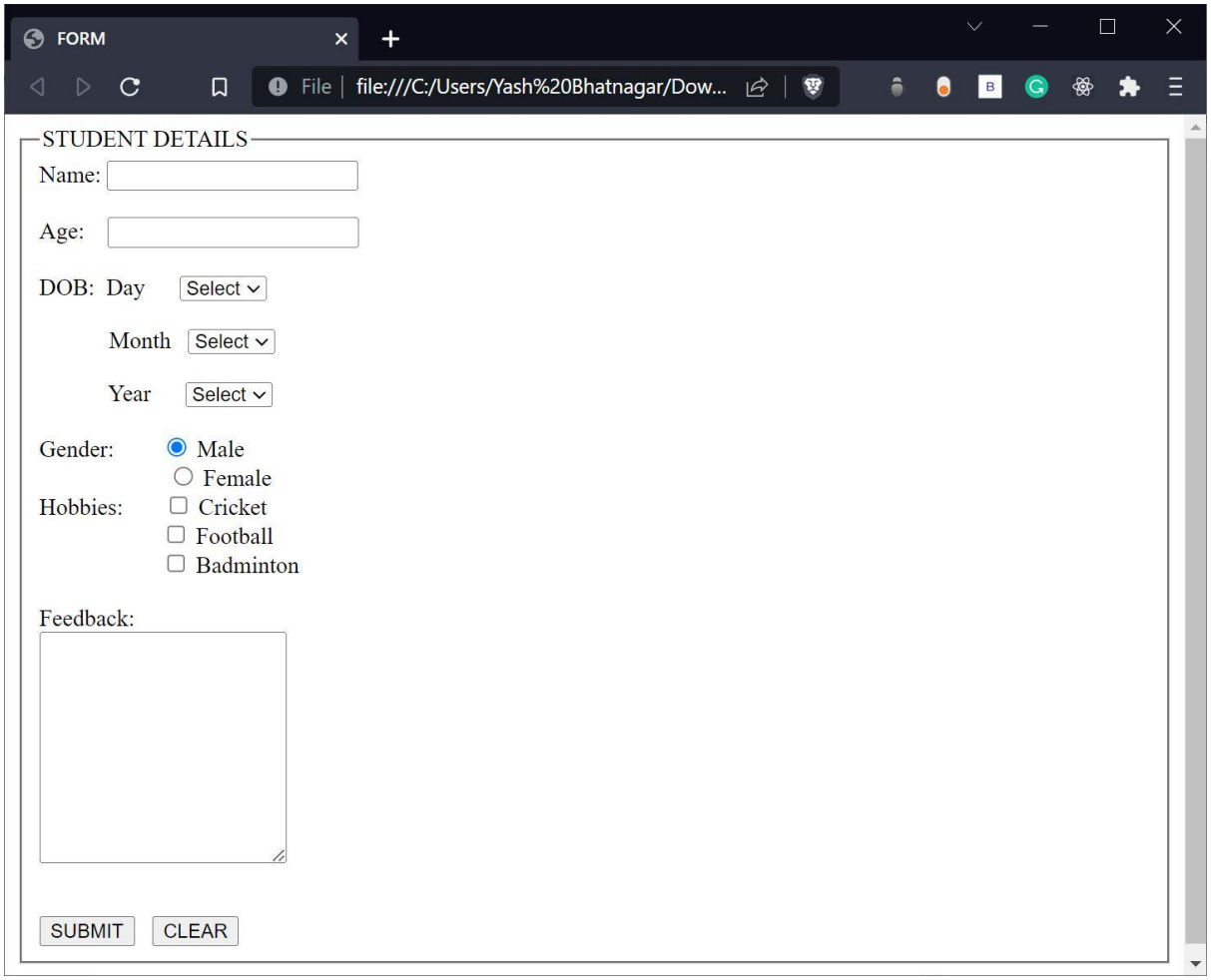
THE <BUTTON> ELEMENT: The <button> element defines a clickable button.

SAMPLE CODE:

```
<html>
 <head>
  <title> FORM </title>
 </head>
 <body>
  <form action= "abc.html" method = "post">
   <fieldset>
      <legend>STUDENT DETAILS</legend>
      Name: <input type="text">            <br><br>
      Age:    <input type="text">          <br>   <br>
      DOB:  Day        <Select>
                  <option>Select</option>
                  <option>1</option>
                 <option>2</option>
                  <option>3</option>
                    :
                  <option>31</option>
                 </Select><br><br>        
                        
         Month     <Select>
                   <option>Select </option>
                <option>Feb
```

```
                <option>March
            <option >Dec
                </Select><br><br>       
                       
          Year         <Select>
                    <option>Select</option>
                    <option>1982</option>
                <option>1983</option>
                    :
                    <option>2016</option>
                </Select>
    <br> <br>
      Gender:       
                        <input type="Radio" name="a" checked> Male  <br>
                           

                   
        <input type="Radio" name="a"> Female <br>
      Hobbies:      
        <input type="Checkbox"> Cricket
<br>            &n
bsp;       
        <input type="Checkbox"> Football
<br>          
                                 
        <input type="Checkbox"> Badminton<br><br>
      Feedback: <br>
        <textarea rows ="10" cols = "20"> </textarea>
      <br><br><br>
      <input type="submit" value="SUBMIT">
       
      <input type="reset" value="CLEAR">

    </fieldset>
  </form> </html>
```

OUTPUT:


FORM
File | file:///C:/Users/Yash%20Bhatnagar/Dow...
STUDENT DETAILS
Name:
Age:
DOB: Day Select
Month Select
Year Select
Gender: Male
Female
Hobbies: Cricket
Football
Badminton
Feedback:
SUBMIT
CLEAR

PROGRAM NO: 5

OBJECTIVE: Write a java program to show:

a) multi-level inheritance in java
b) multi-threading

THEORY:

JAVA: Java is one of the most popular and widely used programming language.

- Java has been one of the most popular programming language for many years.
- Java is Object Oriented. However it is not considered as pure object oriented as it provides support for primitive data types (like int, char, etc)

A) Multilevel Inheritance: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.

SAMPLE CODE:

```
import java.io.*;
import java.lang.*;
import java.util.*;

class one
{
   public void first()
   {
      System.out.println("WEB");
   }
}

class two extends one
{
   public void second()
      {
       System.out.println("Technology");
      }
}

class three extends two
 {
```

```
    public void third()
    {
        System.out.println("Lab");
    }
}

public class Main
{
    public static void main(String[] args)
    {
        three g = new three();
        g.first();
        g.second();
        g.third();
    }
}
```

OUTPUT:

```
Web
Technology
Lab
```

B) Multithreading: Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

SAMPLE CODE:

```
class abc extends Thread
 {
    public void run()
    {
        try
         {
            System.out.println(“Thread " + Thread.currentThread().getId()+ " is running");
        }
        catch (Exception e)
        {
            System.out.println("Exception is caught");
        }
```

```
    }
}

// Main Class
public class Multithread
{
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 1; i < n; i++)
          {
            abc ob = new abc();
            ob.start();
          }
    }
}
```

OUTPUT:

```
Thread 5 is running
Thread 4 is running
Thread 6 is running
Thread 2 is running
Thread 1 is running
Thread 3 is running
Thread 7 is running
```

PROGRAM NO: 6

OBJECTIVE: Write a Java applet to display the Application Program screen i.e. calculator and other.

THEORY:
APPLET: Applet is a special type of program that is embedded in the web page to generate the dynamic content. It runs inside the browser and works at client side.

APPLET LIFE CYCLE STAGES: It has the following stages-
1. Born or initialization
2. Running
3. Idle or stopped
4. Dead or terminate

1. APPLET BORN STATE: It is the first state of applet lifecycle. Applet enters in a born/ initialization state when it is first loaded. Applet moves in a born state when init() method of Applet class executes.

SYNTAX OF INIT() METHOD IN JAVA APPLET:
```
public void init()
{ }
```

2. APPLET RUNNING STATE: Applet enters in a running state from born state when a start() method of Applet class executes.

SYNTAX OF START() :
```
public void start()
{ }
```

3. APPLET IDLE STATE: A running appletenters in a idle sate from running state when execution of start complete.

SYNTAX OF STOP():
```
public void stop()
{ }
```

4. APPLET DEAD STATE: **T**he Applet will be terminate when destroy () method calls. In a applet lifecycle, applet terminates/destroys ones.

SYNTAX OF DESTROY():
```
public void destroy()
{ }
```

5. APPLET DISPLAY STATE: The Display state executes immediately after when applet enters in a running state. In this state we display the information on the output screen using paint() method.

SYNTAX OF DISPLAY STATE:

```
public void paint(Graphics  g)
{ }
```

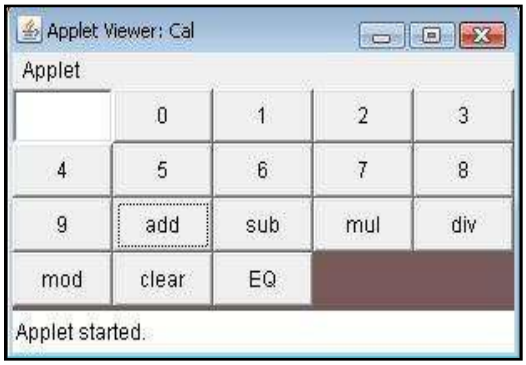
SAMPLE CODE:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="Cal" width=300 height=300>
</applet>
*/
public class Cal extends Applet implements ActionListener
{
    String msg=" ";
    int v1,v2,result;
    TextField t1;
    Button b[]=new Button[10];
    Button add,sub,mul,div,clear,mod,EQ;
    char OP;

    public void init()
     {
          Color k=new Color(120,89,90);
          setBackground(k);
          t1=new TextField(10);
          GridLayout gl=new GridLayout(4,5);
          setLayout(gl);
          for(int i=0;i<10;i++)
            {
                b[i]=new Button(""+i);
             }
          add=new Button("add");
          sub=new Button("sub");
          mul=new Button("mul");
          div=new Button("div");
          mod=new Button("mod");
          clear=new Button("clear");
          EQ=new Button("EQ");
          t1.addActionListener(this);
          add(t1);
```

```
        for(int i=0;i<10;i++)
          {
          add(b[i]);
          }
        add(add);
        add(sub);
        add(mul);
        add(div);
        add(mod);
        add(clear);
        add(EQ);
        for(int i=0;i<10;i++)
          {
          b[i].addActionListener(this);
          }
        add.addActionListener(this);
        sub.addActionListener(this);
        mul.addActionListener(this);
        div.addActionListener(this);
        mod.addActionListener(this);
        clear.addActionListener(this);
        EQ.addActionListener(this);
  }
public void actionPerformed(ActionEvent ae)
 {
        String str=ae.getActionCommand();
        char ch=str.charAt(0);
        if ( Character.isDigit(ch))
        t1.setText(t1.getText()+str);
        else
        if(str.equals("add"))
         {
             v1=Integer.parseInt(t1.getText());
             OP='+';
             t1.setText("");
         }
        else if(str.equals("sub"))
         {
             v1=Integer.parseInt(t1.getText());
             OP='-';
             t1.setText("");
         }
        else if(str.equals("mul"))
         {
             v1=Integer.parseInt(t1.getText());
             OP='*';
             t1.setText("");
```

```
        }
       else if(str.equals("div"))
        {
            v1=Integer.parseInt(t1.getText());
            OP='/';
            t1.setText("");
        }
       else if(str.equals("mod"))
        {
            v1=Integer.parseInt(t1.getText());
            OP='%';
            t1.setText("");
        }
      if(str.equals("EQ"))
        {
            v2=Integer.parseInt(t1.getText());
            if(OP=='+')
            result=v1+v2;
            else if(OP=='-')
            result=v1-v2;
            else if(OP=='*')
            result=v1*v2;
            else if(OP=='/')
            result=v1/v2;
            else if(OP=='%')
            result=v1%v2;
            t1.setText(""+result);
          }
       if(str.equals("clear"))
          {
            t1.setText("");
          }
    }
}
```

OUTPUT:



PROGRAM NO: 7

OBJECTIVE: Writing program in XML for creation of DTD, which specifies set of rules. Create a style sheet in CSS/ XSL & display the document in internet explorer.

THEORY:

XML: XML stands for **E**xtensible **M**arkup **L**anguage. It is a dynamic markup language. It is used to transform data from one form to another form.

SYNTAX:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML DTD: XML Document Type Declaration, commonly known as DTD, is a way to describe precisely the XML language. DTDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

SYNTAX:

```
<!DOCTYPE element DTD identifier
  [
  declaration1
  declaration2
  ........
  ]>
```

In the above syntax

- DTD starts with <!DOCTYPE> delimiter.
- An element tells the parser to parse the document from the specified root element.
- DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called external subset.
- The square brackets [ ] enclose an optional list of entity declarations called internal subset.

TYPES OF DTD: DTD can be classified on its declaration basis in the XML document, such as:

- Internal DTD
- External DTD

INTERNAL DTD: A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, standalone attribute in XML declaration must be set to yes. This means the declaration works independent of external source.

SYNTAX:
<!DOCTYPE root-element [element-declarations]>
where root-element is the name of root element and element-declarations is where you declare the elements.

EXTERNAL DTD: In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To reference it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

SYNTAX:
<!DOCTYPE root-element SYSTEM "file-name">
where file-namen is the file with .dtd extension.

An XML file can be displayed using two ways. These are as follows:-

1. Cascading Style Sheet (CSS)
2. Extensible Stylesheet Language Transformation (XSL)

DISPLAYING XML FILE USING CSS: CSS can be used to display the contents of the XML document in a clear and precise manner. It gives the design and style to whole XML document. In order to display the XML file using CSS, link XML file with CSS.

SYNTAX:
<?xml-stylesheet type="text/css" href="name_of_css_file.css"?>

DISPLAYING XML FILE USING XSLT: E**X**tensible **S**tylesheet **L**anguage **T**ransformation commonly known as XSLT is a way to transform XML document into other formats such as XHTML.

1. <xsl:template> defines a way to reuse templates in order to generate the desired output for nodes of a particular type/context.various template are used with the template like name, match, mode.

2. <xsl:value-of> tag puts the value of the selected node as per XPath expression, as text.

3. <xsl:for-each> tag applies a template repeatedly for each node.

4.<xsl:sort> tag specifies a sort criteria on the nodes.

5. <xsl:if> tag specifies a conditional test against the content of nodes.

SAMPLE CODE:

XML file: Books.xml-

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Rule.css"?>
<books>
   <heading>Welcome</heading>
   <book>
      <title>Title -: Web Programming</title>
      <author>Author -: Chrisbates</author>
      <publisher>Publisher -: Wiley</publisher>
      <edition>Edition -: 3</edition>
      <price> Price -: 300</price>
   </book>
   <book>
      <title>Title -: Internet world-wide-web</title>
      <author>Author -: Ditel</author>
      <publisher>Publisher -: Pearson</publisher>
      <edition>Edition -: 3</edition>
      <price>Price -: 400</price>
   </book>
   <book>
      <title>Title -: Computer Networks</title>
      <author>Author -: Foruouzan</author>
      <publisher>Publisher -: Mc Graw Hill</publisher>
      <edition>Edition -: 5</edition>
      <price>Price -: 700</price>
   </book>
   <book>
      <title>Title -: DBMS Concepts</title>
      <author>Author -: Navath</author>
      <publisher>Publisher -: Oxford</publisher>
      <edition>Edition -: 5</edition>
      <price>Price -: 600</price>
   </book>
   <book>
      <title>Title -: Linux Programming</title>
      <author>Author -: Subhitab Das</author>
      <publisher>Publisher -: Oxford</publisher>
      <edition>Edition -: 8</edition>
      <price>Price -: 300</price>
   </book>
</books>
```

In the above example, Books.xml is linked with Rule.css which contains the corresponding style sheet rules.

CSS FILE: Rule.css-

```
books {
    color: white;
    background-color : gray;
    width: 100%;
}
 heading {
    color: green;
    font-size : 40px;
    background-color : powderblue;
}
 heading, title, author, publisher, edition, price {
    display : block;
}
 title {
    font-size : 25px;
    font-weight : bold;
}
```

OUTPUT:

PROGRAM NO: 8

OBJECTIVE: Write programs using Java script for Web Page to display browsers information.

THEORY:

The navigator object contains information about the browser.

NAVIGATOR OBJECT PROPERTIES:

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
geolocation	Returns a Geolocation object that can be used to locate the user's position
language	Returns the language of the browser
onLine	Determines whether the browser is online
platform	Returns for which platform the browser is compiled
product	Returns the engine name of the browser
userAgent	Returns the user-agent header sent by the browser to the server

SAMPLE CODE:

```
<html>
  <head>
    <title>Browser Information</title>
  </head>
  <body>
    <h1>Browser Information</h1>
    <hr>
    <p>
      The <b>navigator</b> object contains the following information about the browser you   are
      using.
    </p>
    <ul>
      <script LANGUAGE="JavaScript" type="text/javascript">
         document.write("<li><b>Code Name:</b> " + navigator.appCodeName);
         document.write("<li><b>App Name:</b> " + navigator.appName);
          document.write("<li><b>App Version:</b> " + navigator.appVersion);
          document.write("<li><b>User Agent:</b> " + navigator.userAgent);
          document.write("<li><b>Language:</b> " + navigator.language);
          document.write("<li><b>Platform:</b> " + navigator.platform);
```

```
    </script>
   </ul>
   <hr>
  </body>
</html>
```

OUTPUT:

Browser Information

The **navigator** object contains the following information about the browser you are using.

- **Code Name:** Mozilla
- **App Name:** Netscape
- **App Version:** 5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
- **User Agent:** Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
- **Language:** en-US
- **Platform:** Win32

PROGRAM NO: 9

OBJECTIVE: Demonstrate client-server communication using socket programming.

THEORY: To connect to another machine we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port. The java.net.Socket class represents a Socket.

To open a socket:
Socket socket = new Socket ("127.0.0.1", 4999)

- First argument – **IP address of Server**. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

- Second argument – **TCP Port**. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

SAMPLE CODE:

File name as Client.java-

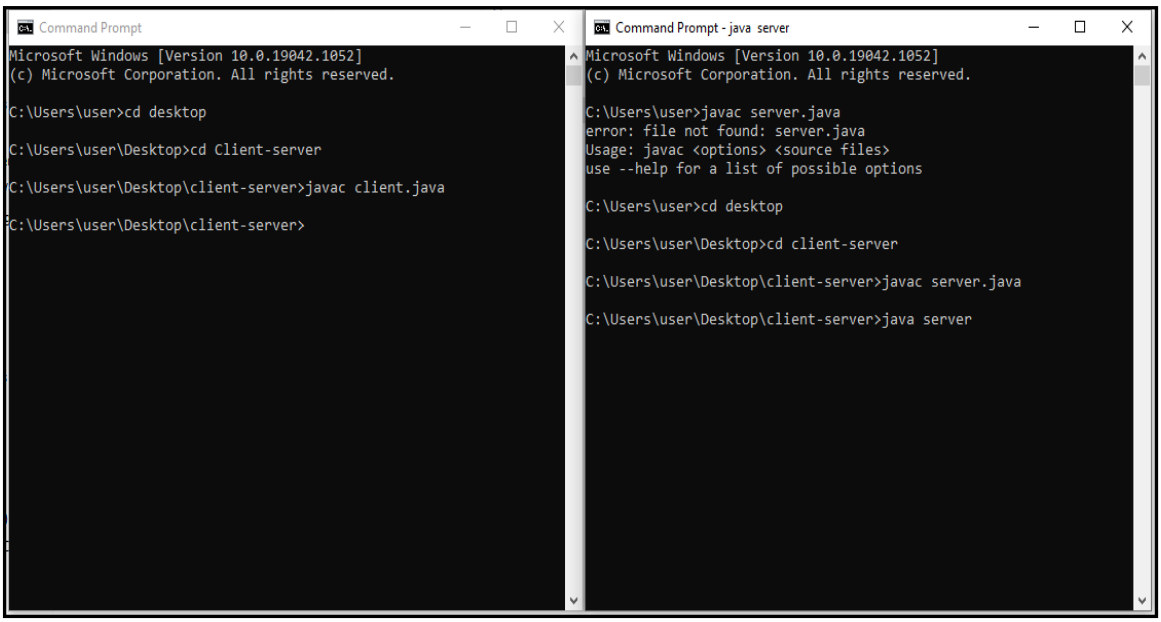
```
import java.net.*;
import java.io.*;
public class client
 {
    public static void main (String[] args) throws IOException
      {
        Socket s=new Socket("localhost",4999);
      }
 }
```

File Name as Server.java

```
import java.net.*;
import java.io.*;
public class server
 {
    public static void main(String[] args) throws IOException
       {
            ServerSocket ss = new ServerSocket (4999);
           Socket s=ss.accept();
           System.out.println("client connection");
       }
```

}

OUTPUT:


Command Prompt
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
C:\Users\user>cd desktop
C:\Users\user\Desktop>cd Client-server
C:\Users\user\Desktop\client-server>javac client.java
C:\Users\user\Desktop\client-server>
Command Prompt - java server
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
C:\Users\user>javac server.java
error: file not found: server.java
Usage: javac <options> <source files>
use --help for a list of possible options
C:\Users\user>cd desktop
C:\Users\user\Desktop>cd client-server
C:\Users\user\Desktop\client-server>javac server.java
C:\Users\user\Desktop\client-server>java server

PROGRAM NO: 10

OBJECTIVE: Install TOMCAT web server and APACHE. Access the above developed staticweb pages for books web site, using these servers by putting the web pages developed **.**

THEORY:

Introduction to servlets:

Servlets are java programs that run on web or application servers, acting as middle layer between request coming from the web browsers or other HTTP clients and database servers. Servlets process user request, produce the results and sends the results as a response to the user.

Life cycle of Servlet

Loading: The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute <load-on-startup> of web.xml file. If the attribute <load-on-startup> has a positive value then the servlet is load with loading of the container otherwise it load when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.

Initialization: After creating the instances, the servlet container calls the init() method and passes the servlet initialization parameters to the init() method. The init() must be called by the servlet container before the servlet can service any request. The initialization parameters persist untill the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet. The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.

Servicing the Request: After successfully completing the initialization process, the servlet will be available for service. Servlet creates seperate threads for each request. The sevlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost()) for handling the request and sends response to the client using the methods of the response object.

Destroying the Servlet: If the servlet is no longer needed for servicing any request, the servlet container calls the destroy() method . Like the init() method this method is also called only once throughout the life cycle of the servlet. Calling the destroy() method indicates to the servlet container not to sent the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.

Servlet API Provides the following two packages

Javax.servlet

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

Javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

PROCESS TO CREATE ENVIRONMENT FOR EXCECTION OF A SERVLET SAMPLE CODE:ME

Set the JAVA_HOME Variable

You must set the JAVA_HOME environment variable to tell Tomcat where to find Java. Failingto properly set this variable prevents Tomcat from handling JSP pages. This variable should listthe base JDK installation directory, not the bin subdirectory.
We could do this by click on the Start menu, select Control Panel, choose System, click on the advanced tab, press the Environment Variables button at the bottom, and enter the JAVA_HOME variable and value directly as:
Name: JAVA_HOME
Value: C:\jdk

Set the CLASSPATH

Since servlets and JSP are not part of the Java 2 platform, standard edition, youhave to identify the servlet classes to the compiler. The server already knows about theservlet classes, but the compiler (i.e., javac) you use for development probably doesn't.So, if you don't set your CLASSPATH, attempts to compile servlets, tag libraries, or other classes that use the servlet and JSP APIs will fail with error messages about unknownclasses.
Name: JAVA_HOME
Value: *install_dir/common/lib/servlet-api.jar*

Turn on Servlet Reloading

The next step is to tell Tomcat to check the modification dates of the class files of requested servlets and reload ones that have changed since they were loaded into the server's memory. This slightly degrades performance in deployment situations, so is turned off by default. However, if you fail to turn it on for your development server, you'll have to restart the server every time you recompile a servlet that has already been loaded into the server's memory.

To turn on servlet reloading, edit install_dir/conf/server.xml and add a Default Context sub element to the main Host element and supply true for the reloadable attribute. For example, in Tomcat 5.0.27, search for this entry:
<Host name="localhost" debug="0" appBase="webapps" ...> and then insert the following immediately below it:
<Default Context reloadable="true"/>
Be sure to make a backup copy of server.xml before making the above change.

Enable the Invoker Servlet

The invoker servlet lets you run servlets without first making changes to yourWeb application's deployment descriptor. Instead, you just drop your servlet into WEB-INF/classes and use the URL http://host/servlet/ServletName . The invoker servlet isextremely convenient when you arelearning and even when you are doing your initialdevelopment.

To enable the invoker servlet, uncomment the following servlet and servlet-mapping elements in install_dir/conf/web.xml. Finally, remember to make a backup copyof the original version of this file before you make the changes.

```
<servlet>

        <servlet-name>invoker</servlet-name>

        <servlet-class> org.apache.catalina.servlets.InvokerServlet
        </servlet-class>
        …
</servlet>
…
<servlet-mapping>

        <servlet-name>invoker</servlet-name>

        <url-pattern>/servlet/*</url-pattern>

</servlet-mapping>
```

PROGRAM NO: 11

OBJECTIVE: Write a JSP which insert the details of the 3 or 4 users who register with the the web site by using registration form. Authenticate the user when he submits the login form using the username and password.

THEORY:

JSP Scripting Elements

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

1. Expressions of the form **<%= expression %>** that are evaluated and inserted into the output,
2. Scriptlets of the form **<% code %>** that are inserted into the servlet's service method, and
3. Declarations of the form **<%! code %>** that are inserted into the body of the servlet class, outside of any existing methods.

Each of these is described in more detail below.

JSP Expressions

A JSP expression is used to insert Java values directly into the output. It has the following form:

```
<%= Java Expression %>
```

The Java expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at run-time (when the page is requested), and thus has full access to information about the request. For example, the following shows the date/time that the page was requested:

```
Current time: <%= new java.util.Date() %>
```

To simplify these expressions, there are a number of predefined variables that you can use.

These implicit objects are discussed in more detail later, but for the purpose of expressions, the most important ones are:

- request, the HttpServletRequest;
- response, the HttpServletResponse;
- session, the HttpSession associated with the request (if any); and
- out, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.

JSP Scriptlets

If you want to do something more complex than insert a simple expression, JSP scriptlets let you insert arbitrary code into the servlet method that will be built to generate the page. Scriptlets have the following form:

```
<% Java Code %>
```

Scriptlets have access to the same automatically defined variables as expressions. So, for example, if you want output to appear in the resultant page, you would use the out variable.

```
<%
        String queryData = request.getQueryString();
        out.println("Attached GET data: " + queryData);
%>
```

Note that code inside a scriptlet gets inserted exactly as written, and any static HTML (template text) before or after a scriptlet gets converted to print statements. This means that scriptlets need not contain complete Java statements, and blocks left open can affect the static HTML outside of the scriptlets.

JSP Declarations

A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (outside of the service method processing the request). It has the following form:

```
<%! Java Code %>
```

Since declarations do not generate any output, they are normally used in conjunction with JSP expressions or scriptlets. For example, here is a JSP fragment that prints out the number of times the current page has been requested since the server booted (or the servlet class was changed and reloaded):

```
<%! private int accessCount = 0; %>
```

SAMPLE CODE:

Login.html:

```
<html>
 <body>
<center><h1>XYZ Company Ltd.</h1></center>
<table border="1" width="100%" height="100%">
<tr>
 <td valign="top" align="center"><br/>
 <form action="auth.jsp"><table>
```

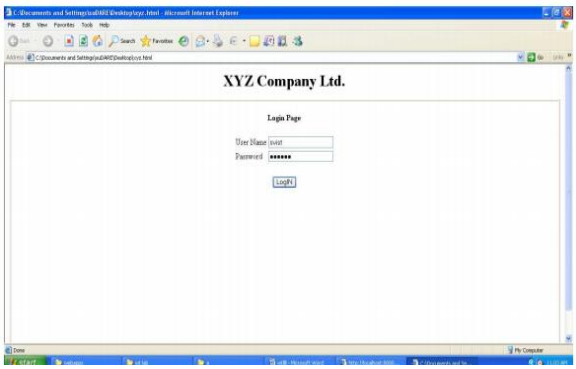
```
<tr>
<td colspan="2" align="center"><b>Login Page</b></td>
</tr>
<tr>
<td colspan="2" align="center"><b> </td>
</tr>
<tr>
<td>User Name</td>
<td><input type="text" name="user"/></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="pwd"/></td>
</tr>
<tr>
<td> </td>
<td> </td>
</tr>
<tr>
<td colspan="2" align="center"><input type="submit" value="LogIN"/></td>
</tr>
</table>
</form>
</td>
</tr>
</table>
</body>
</html>
```

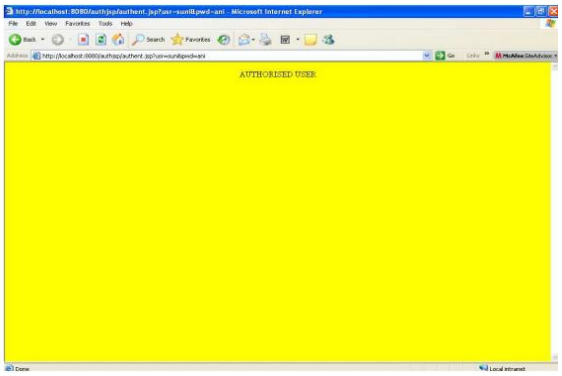
Auth.jsp:

```
<%@page import="java.sql.*;"%>
<html>
<head>
<title>
This is simple data base example in JSP</title>
</title>
</head>
<body bgcolor="yellow">
<%!String uname,pwd;%>
<%
uname=request.getParameter("user");
pwd=request.getParameter("pwd");
 try
{
 Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
 Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@195.100.101.158:1521:CCLAB",
"scott","tiger ");
 Statement st=con.createStatement();
 ResultSet rs=st.executeQuery("select name,password from personal where
name='"+uname+"' and password='"+pwd+"'");
if(rs.next())
    {
    out.println("Authorized person");
    }
    else
    {
    out.println("UnAuthorized person");
    }
  con.close();
    }
    catch(Exception e){out.println(""+e);}
    %>
</body>
</html>
```

OUTPUT





PROGRAM NO: 12

OBJECTIVE: Install a database (Mysql or Oracle). Create a table which should contain at least the following fields: name, password, email-id, phone number Write a java program/servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page.

THEORY:

JDBC Driver Types

There are four types of JDBC drivers in use:

Type 1: JDBC-ODBC Bridge

A Type 1 JDBC-ODBC Bridge provides application developers with a way to access JDBC drivers via the JDBC API. Type 1 JDBC drivers translate the JDBC calls into ODBC calls and then send the calls to the ODBC driver. Type 1 JDBC drivers are generally used when the database client libraries need to be loaded on every client machine.

Type 2: Native API/Partly Java Driver

A Type 2 Native API/Partly Java Driver is a partial Java driver because it converts JDBC calls into database specific calls. Type 2 Native API/Partly Java Driver communicates directly with the database server.

Type 3: Pure Java Driver

A Type 3 Pure Java Driver works in a three tiered architecture. The JDBC calls are passed via the network to the middle tier server. This server translates the calls to the database specific native interface to further request the server. JDBC drivers available from Simba are Type 3 drivers.

Type 4: Native Protocol Java Driver

The type 4 driver is written completely in Java and is hence platform independent. It is installed inside the Java Virtual Machine of the client. It provides better performance over the type 1 and 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 3 drivers, it does not need associated software to work.A Type 4 Native

Protocol Java Driver converts JDBC calls into the database specific calls so that the client applications can communicate directly with the server.

SAMPLE CODE:

Registration.html:

```
<html>
<head>
<title>Registration page</title>
</head>
<body bgcolor="#00FFFf">
<form METHOD="POST" ACTION="register">
<CENTER>
<table>
<center>
<tr> <td> Username </td>
<td><input type="text" name="usr"> </td> </tr>
<tr><td> Password </td>
<td><input type="password" name="pwd"> </td> </tr>
<tr><td>Age</td>
<td><input type="text" name="age"> </td> </tr>
<tr> <td>Address</td>
<td> <input type="text" name="add"> </td> </tr>
<tr> <td>email</td>
<td> <input type="text" name="mail"> </td> </tr>
<tr> <td>Phone</td>
<td> <input type="text" name="phone"> </td> </tr>
<tr> <td colspan=2 align=center> <input type="submit" value="submit"> </td> </tr>
</center>
</table>          </form>          </body>
```

Login.html

```
<html>
<head>
<title>Registration page</title>
</head>
<body bgcolor=pink> <center> <table>
<form METHOD="POST" ACTION="authent">
<tr> <td> Username </td>
<td><input type="text" name="usr"></td> </tr>
```

```
<tr> <td> Password </td>
<td> <input type="password" name="pwd"> </td> </tr>
<tr> <td align=center colspan="2"><input type="submit" value="submit"></td> </tr>
</table> </center>
</form>          </body>          </html>
```

Ini.java:

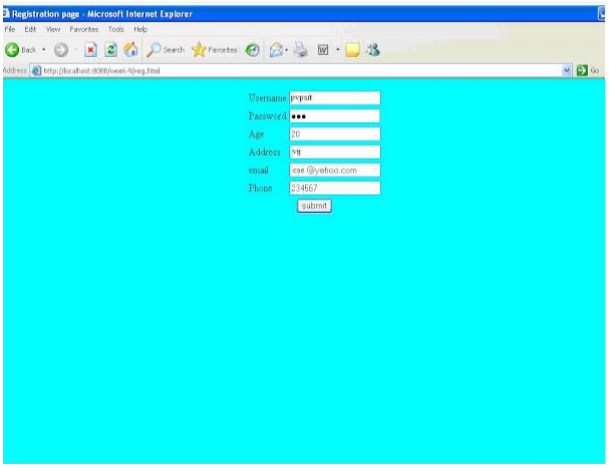
```
import javax.servlet.*;
import java.sql.*;
import java.io.*;
public class Ini extends GenericServlet
{
private String user1,pwd1,email1;
public void service(ServletRequest req,ServletResponse res) throws
ServletException,IOException
{
user1=req.getParameter("user");
pwd1=req.getParameter("pwd");
email1=req.getParameter("email");
res.setContentType("text/html");
PrintWriter out=res.getWriter();
try
  {
   Class.forName("oracle.jdbc.driver.OracleDriver");
   Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@195.100.101.158:1521:cclab","scott","tige
r");
  PreparedStatement st=con.prepareStatement("insert into personal values(?,?,?,?,?,?)");
 st.setString(1,user1);
st.setString(2,pwd1);
st.setString(3,"25");
st.setString(4,"hyd");
st.setString(5,email1);
st.setString(6,"21234");
st.executeUpdate();
con.close();  }
catch(SQLException s)
{  out.println("not found "+s);
}
catch(ClassNotFoundException c)
```

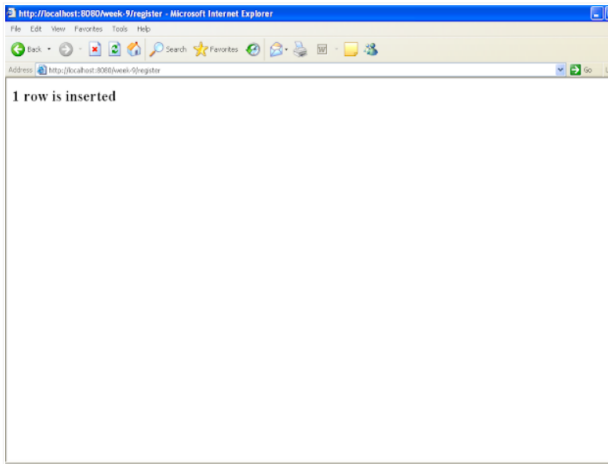
{ out.println("not found "+c); } } }

web.xml:

```
<web-app>
<servlet>
<servlet-name>init1</servlet-name>
<servlet-class>Ini</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>init1</servlet-name>
<url-pattern>/regis</url-pattern>
</servlet-mapping>
</web-app>
```

OUTPUT:





PROGRAM NO: 13

OBJECTIVE: Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following.

1. Create a Cookie and add these four user id's and passwords to this Cookie.
2. Read the user id and passwords entered in the Login form (week1) and authenticate with the values (user id and passwords) available in the cookies.

THEORY:

Servlet Life cycle:

1. Servlet class loading
2. Servlet Instantiation
3. call the init method
4. call the service method
5. call destroy method

Class loading and instantiation

If you consider a servlet to be just like any other Java program, except that it runs within a servlet container, there has to be a process of loading the class and making it ready for requests. Servlets do not have the exact equivalent of a main method that causes them to start execution. When a web container starts it searches for the deployment descriptor (WEB.XML) for each of its web applications. When it finds a servlet in the descriptor it will create an instance of the servlet class. At this point the class is considered to be loaded (but not initialized).

The init method

The HttpServlet class inherits the init method from GenericServlet. The init method performs a role slightly similar to a constructor in an "ordinary" Java program in that it allows initialization of an instance at start up. It is called automatically by the servlet container and as it causes the application context (WEB.XML) to be parsed and any initialization will be performed. It comes in two versions, one with a zero parameter constructor and one that takes a ServletConfig parameter.

The servlet engine creates a request object and a response object. The servlet engine invokes the servlet service() method, passing the request and response objects. Once the init method returns the servlet is said to be placed into service. The process of using init to initialize servlets means that it is possible to change configuration details by modifying the deployment descriptor without having them hard coded in with your Java source and needing a re-compilation.

void init(ServletConfig sc)

Calling the service method

The service() method gets information about the request from the request object, processes the request, and uses methods of the response object to create the client response. The service method can invoke other methods to process the request, such as doGet(), doPost(), or methodsyou write. The service method is called for each request processed and is not normally overridden by the programmer.
The code that makes a servlet "go" is the.

servlet void service(ServletRequest req,ServletResponse res)

The destroy Method

Two typical reasons for the destroy method being called are if the container is shutting down or if the container is low on resources. This can happen when the container keeps a pool of instances of servlets to ensure adequate performance. If no requests have come in for a particularservlet for a while it may destroy it to ensure resources are available for the servlets that are being requested. The destroy method is called only once, before a servlet is unloaded and thus you cannot be certain when and if it is called.
void destroy()

ServletConfig Class

ServletConfig object is used by the Servlet Container to pass information to the Servlet during it's initialization. Servlet can obtain information regarding initialization parameters and their values using different methods of ServletConfig class initialization parameters are name/value pairs used to provide basic information to the Servlet during it's initialization like JDBC driver name, path to database, username, password etc.

Methods of ServletConfig class

Following are the four methods of this class :

1. **getInitParameter(String paramName)** Returns value of the given parameter. If value of parameter could not be found in web.xml file then a null value is returned.
2. **GetInitParameterNames()** Returns an Enumeration object containing all the names of initialization parameters provided for this Servlet.
3. **GetServletContext()** Returns reference to the ServletContext object for this Servlet. It is similar to getServletContext() method provided by HttpServlet class.
4. **GetServletName()** Returns name of the Servlet as provided in the web.xml file or if noneis provided then returns complete class path to the Servlet.

SAMPLE CODE:

1. Create a Cookie and add these four user id's and passwords to this Cookie.

cologin.html:

```
<html>
<head>
<title> login Page </title>
<p style= "background:yellow; top:100px; left:250px; position:absolute; ">
</head>
<body>
<form  ACTION="clogin">
<label> Login </label>
<input type="text" name="usr" size="20"> <br> <br>
<label> Password </label>
<input type="password" name="pwd" size="20"> <br> <br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

cologin1.html

```
<html>
<head>
<title> login Page </title>
<p style= "background:yellow; top:100px; left:250px; position:absolute; ">
</head>
<body>
<form  ACTION="clogin1">
<label> Login </label>
<input type="text" name="usr" size="20"> <br> <br>
<label> Password </label>
<input type="password" name="pwd" size="20"> <br> <br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

Addcook.java:

```
import javax.servlet.* ;
import javax.servlet.http.*;
```

```
import java.io.*;
public class Addcook extends HttpServlet
{
String user,pas;
public void service(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
Cookie c1=new Cookie("usr1","suni");
Cookie p1=new Cookie("pwd1","ani");
Cookie c2=new Cookie("usr2","abc");
Cookie p2=new Cookie("pwd2","123");
Cookie c3=new Cookie("usr3","def");
Cookie p3=new Cookie("pwd3","456");
Cookie c4=new Cookie("usr4","mno");
Cookie p4=new Cookie("pwd4","789");
res.addCookie(c1);
res.addCookie(p1);
res.addCookie(c2);
res.addCookie(p2);
res.addCookie(c3);
res.addCookie(p3);
res.addCookie(c4);
res.addCookie(p4);
out.println("COOKIE ADDED");
}
}
```

Clogin.java:

```
import javax.servlet.* ;
import javax.servlet.http.*;
import java.io.*;
public class Clogin extends HttpServlet
{
String user,pas;
public void service(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
```

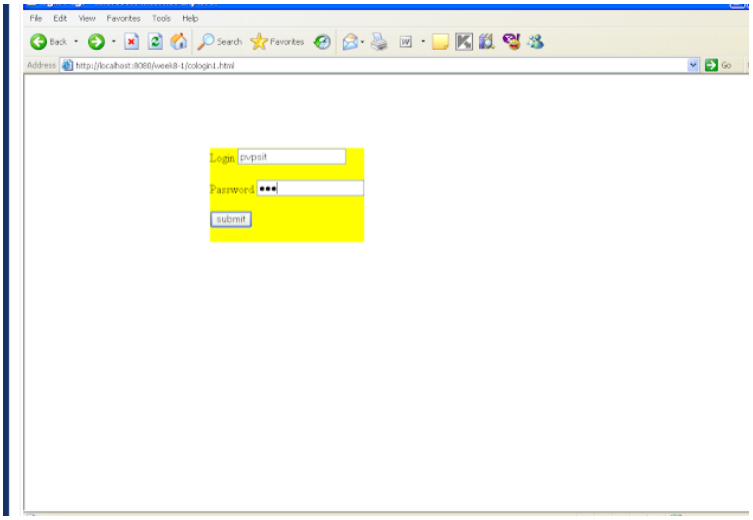
```
user=req.getParameter("usr");
pas=req.getParameter("pwd");
Cookie[] c=req.getCookies();
for(int i=0;i<c.length;i++)
{
if((c[i].getName().equals("usr1")&&c[i+1].getName().equals("pwd1"))||
c[i].getName().equals("usr2")
&&c[i+1].getName().equals("pwd2"))||(c[i].getName().equals("usr3")&&
c[i+1].getName().equals("pwd3"))||(c[i].getName().equals("usr4")&&
c[i+1].getName().equals("pwd4") ))
                {
                     if((user.equals(c[i].getValue()) && pas.equals(c[i+1].getValue())) )
                {
                //RequestDispatcher rd=req.getRequestDispatcher("/cart.html");
                rd.forward(req,res);
                }
                else
                {
                out.println("YOU ARE NOT AUTHORISED USER ");
                //res.sendRedirect("/cookdemo/cologin.html");
                }
                }
}
}
}
```

Web.xml:

```
<web-app>
<servlet>
<servlet-name>him</servlet-name>
<servlet-class>Clogin</servlet-class>
</servlet>
<servlet>
<servlet-name>him1</servlet-name>
<servlet-class>Addcook</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>him</servlet-name>
<url-pattern>/clogin</url-pattern>
</servlet-mapping>
```

<servlet-mapping>
<servlet-name>him1</servlet-name>
<url-pattern>/clogin1</url-pattern>
</servlet-mapping>
</web-app>

OUTPUT



2 Read the user id and passwords entered in the Login form (week1) and authenticate with the values (user id and passwords) available in the cookies.

STEPS:

1. **Crete a web page to get input from user**

2. **Write code for java servelet**
   If the user is a valid user (i.e., user-name and password match) you should welcome the user by name (user-name) else you should display “You are not an authenticated user “. Use init-parameters to do this.

3. **Do required changes in web.xml**

   Store the user-names and passwords in the webinf.xml and access them in the servlet by using the getInitParameters() method.